Preface

We are proud to introduce the proceedings of the Sixth International Conference on Parallel Problem Solving from Nature, PPSN VI, held in Paris, France, on 18–20 September 2000. PPSN VI was organized in association with the Genetic and Evolutionary Computing Conference (GECCO’2000) and the Congress on Evolutionary Computation (CEC’2000), reflecting the beneficial interaction between the conference activities in Europe and in the USA in the field of natural computation.

Starting in 1990 in Dortmund, Germany (Proceedings, LNCS vol. 496, Springer, 1991), this biannual meeting has been held in Brussels, Belgium (Proceedings, Elsevier, 1992), Jerusalem, Israel (Proceedings, LNCS vol. 866, Springer, 1994), Berlin, Germany (Proceedings, LNCS vol. 1141, Springer, 1996), and Amsterdam, The Netherlands (Proceedings, LNCS vol. 1498, Springer, 1998), where it was decided that Paris would be the location of the 2000 conference with Marc Schoenauer as the general chair.

The scientific content of the PPSN conference focuses on problem solving paradigms gleaned from a natural models. Characteristic for Natural Computing is the metaphorical use of concepts, principles and mechanisms underlying natural systems, such as evolutionary processes involving mutation, recombination, and selection in natural evolution, annealing or punctuated equilibrium processes of many-particle systems in physics, growth processes in nature and economics, collective intelligence in biology, DNA-based computing in molecular chemistry, and multi-cellular behavioral processes in neural and immune networks.

Hence the reader will find in these proceedings a variety of sections: Analysis and theory of EAs, Genetic programming, Scheduling, Representations and operators, Co-evolution, Constraint handling techniques, Noisy and non-stationary environments, Evolvable hardware and hardware implementation of EAs, Combinatorial optimization, Applications, Machine learning and classifier systems, New algorithms and metaphors, Multi-objective optimization, EA software.

In total, these proceedings contain 2 invited papers and 86 contributions which were selected from 168 paper submissions to the conference organizers. The members of the program committee (listed on the next pages) had to work on an extremely challenging task in submitting their scientific reviews and facilitating the final decision for the acceptance of the top 51% of all submissions. We are very grateful to these volunteer reviewers who offered their scientific expertise in order to come up with a decision that was as fair as possible. We are aware that, in spite of all our efforts, the review process may not be perfect yet. Moreover, due to limited time and space, some good papers could not be accepted. In any case, we want to thank all authors of submitted papers for their participation, and for their – often involuntary – help in fixing problems with the Web-based submission-and-review procedure.
With respect to that submission procedure, we are deeply indebted to the EvoNet Electronic Communication Committee. We strongly believe that this fully electronic procedure simplified the review process greatly, and facilitated a smooth exchange of papers and reviews. The software used was an improved version of that used for the 1998 edition, which has also been used for various other conferences. One of its main original features was that it allowed the reviewers to mark their preferences among the papers related to their field of interest by browsing through the titles and abstracts. This helped us later to assign the papers and optimize the matching between papers and reviewers. Although some human intervention was needed at the end, we believe that a full autonomous system could be evolved in the near future to provide good support for future events.

As usual, PPSN VI is a poster-only conference; that is, all papers are presented as posters to facilitate personal discussion and the exchange of ideas between the presenter and the audience. Although this might imply a smaller audience than in the case of an oral presentation, the presenter of a paper has a better chance to get in touch with the people most interested in her/his topic. Consequently, posters are not "second-class" papers, as they are usually considered in some other conferences — they are just the means of presenting. The 86 papers presented at the conference were grouped into 7 sessions of about 15 papers each. To simplify the orientation within a poster session and to allow the audience to get a global overview of all sessions, each poster session was introduced by one volunteer who gave a brief overview of all papers presented within a session.

Only the three invited speakers presented a one-hour oral presentation of their research interests, results and scientific views. Aaron Sloman (Birmingham University, UK), Luc Steels (VUB AI Lab, Brussels and Sony Computer Science Laboratory, Paris) and Henrik Hautop Lund (Lego Lab, University of Aarhus, Denmark) all addressed new and emerging topics in natural computation.

Before the technical sessions began on 18 September, two one-day events took place, in cooperation with the adjacent SAB'2000, the Sixth Simulation of Adaptive Behavior conference, organized in Paris from 11 to 16 September (General Chair, Jean-Arcady Meyer). Seven workshops were organized at Collège de France on 16 September, and twelve tutorials were presented at the PPSN venue on 17 September. Both events addressed issues relevant to either conference, or, in most cases, to both conference themes. We would like to thank the corresponding chairs, David W. Corne (Reading University, UK) and Agoston E. Eiben (Vrije Universiteit Amsterdam and Leiden University, The Netherlands). The list of workshops and tutorials, together with their organizers and presenters, can be found on the following introductory pages.

Finally, we would like to thank the sponsors, who helped in one way or another to make this conference possible, with all the organizational hurdles that we had to overcome. These sponsors are the INRIA, who donated the high competence of the Bureau des Relations Extérieures staff that handled the whole local organization, EvoNet European Network of Excellence on Evo-
volutionary Computation (evonet.dcs.napier.ac.uk), the International Society for Genetic and Evolutionary Computation (ISGEC) (www.isgec.org), France Telecom, the Institut Français du Pétrole (www.ifp.fr) and Nutech Solutions, Inc. (www.nutechsolutions.com). We would like to give special mention to the Association Française d'Intelligence Artificielle, AFIA who intellectually supported this conference.

The invaluable help of Jose Carpio Canada and Pedro Castillo (Geneura Team, University of Granada, Spain) for setting up and maintaining the electronic submission procedure, of Benoît Leblanc (Projet Fractales, INRIA, France) our WebMaster, of Amrit Pratap and Sameer Agrawal (Indian Institute of Technology Kanpur, India) and Pierre Collet (CMAP, Ecole Polytechnique, France) during the process of assigning the papers to the reviewers, of Nathalie Gaudechoux (Projet Fractales, INRIA, France) for the preparation of the camera-ready version of these proceedings was highly appreciated and we would like to thank them personally for their efforts to make PPSN VI a success.

We are sure that these proceedings are a witness of the fruitful cooperation and exchange of ideas between natural computation groups from all parts of the world and mark some major progress in the field — and also that the French cultural traditions, represented during the conference by artistic, oenological and gastronomical events, have helped to support this development and to smoothen cultural barriers ...

July 2000

Marc Schoenauer
Juan Julian Merelo
Kalyanmoy Deb, Günter Rudolph and Xin Yao
Évelyne Lutton
Hans-Paul Schwefel
PPSN VI Conference Committee

Marc Schoenauer
CNRS-Ecole Polytechnique, France
Conference Chair

Kalyanmoy Deb
Indian Institute of Technology Kanpur, India
Günter Rudolph
University of Dortmund, Germany
and
Xin Yao
The University of Birmingham, UK
Program Co-Chairs

Juan Julian Merelo
University of Granada, Spain
Electronic Pgm Chair

Hans-Paul Schwefel
University of Dortmund, Germany
Proceedings Chair

Evelyne Lutton
INRIA Rocquencourt, France
Local / Publicity Chair

A.E. Eiben
Free University Amsterdam, and Leiden University, The Netherlands
Tutorial Chair

David Corne
University of Reading, UK
Workshop Chair
PPSN Steering Committee

Yuval Davidor  
*Schema Lrd, Israel*

Kenneth De Jong  
*George Mason University, USA*

Gusz Eiben  
*Leiden University, The Netherlands*

Reinhard Männer  
*Universität Mannheim, Germany*

Marc Schoenauer  
*CNRS-Ecole Polytechnique, France*

Hans-Paul Schwefel  
*Universität Dortmund, Germany*

Hans-Michael Voigt  
*Gesellschaft zur Förderung angewandter Informatik e. V. (GFaI), Germany*
PPSN VI Program Committee

Panagiotis Adamidis, Alexandru Agapie, Sameer Agrawal,
Jarmo Alander, Enrique Alba, Jean-Marc Alliot,
Lee Altenberg, Jarek Arabas, Thomas Bäck,
Wolfgang Banzhaf, Helio Barbosa, David Beasley,
Hugues Bersini, Pierre Bessière, Hans-Georg Beyer,
Philippe Bidaud, Alan Blair, Andrea Bonarini,
Lashon B. Booker, Bertrand Braunschweig, Larry Bull,
E.K. Burke, Pedro Castillo, Nirupam Chakraborti,
Uday Chakraborty, Pierre Collet, Marco Colombetti,
C. Cotta Porras, Peter Cowling, Jason Daida,
Paul James Darwen, Ken DeJong, Daniel Delahaye,
Marco Dorigo, Gerry Dozier, Nicolas Durand,
Javier ElorzaTenreiro, Tom English, Bogdan Fillipic,
Peter Fleming, Cyril Fonlupt, Bernd Freisleben,
Terry Fogarty, Toshio Fukuda, Takeshi Furuhashi,
Roman Galar, Raul Gallard, Attilio Giordana,
Ashish Ghosh, David Goldberg, Ulrich Hammel,
Nikolaus Hansen, Jin-Kao Hao, Inman Harvey,
Tim Hendtlass, Michael Herdy, Francisco Herrera,
Robert Hinterding, Jeff Horn, Hitoshi Iba,
Chen Jong-Chen, Leila Kallel, Lishan Kang,
Chuck Karr, N. Kasabov, Spyros Kazarlis,
Robert E. Keller, JongHwan Kim, Joost N. Kok,
Ibrahim Kuscu, Gary Lamont, Pier-Luca Lanzi,
Rodolphe Leriche, K.S. Leung, Yun Li,
Pierre Liardet, Jiming Liu, Jean Louchet,
Henri Luchian, Elena Marchiori, Worthy Martin,
Keith Mathias, John McDonnell, Zbigniew Michalewicz,
Julian Miller, Claudio Moraga, Heinz Mühlenbein,
Akira Namatame, Martin Oates, Pedro Oliviera,
Una May O'Reilly, Pavel Osmera, Ben Paechter,
Charles Palmer, Zhengjun Pan, Jan Paredis,
Ian Parmee, Alain Pétrowski, Chrisila C. Pettey,
Riccardo Poli, Mitchell Potter, Zhao Qiangfu,
Alain Ratle, Colin Reeves, Robert Reynolds,
Mateen Rizki, Edmund Ronald, Justianin Rosca,
Jonathan Rowe, Grzegorz Rozenberg, Lorenza Saitta,
Ralf Salomon, Jayshree Sarma, Martin Schmidt,
Thorsten Schnier, Michèle Sebag, Sandip Sen,
F. Seredynski, Yuhui Shi, Mark C. Sinclair,
Moshe Sipper, Alice E. Smith, Jim Smith,
Joachim Sprave, Thomas Stidsen, Cho Sung-Bae, El-Ghazali Talbi, Andrea Tettamanzi, Dirk Thierens, Marco Tomassini, Brian Turton, Gilles Venturini, Don Waagen, Peter Whigham, Darrell Whitley, Annie Wu, Xinghuo Yu, Nakao Zensho, Eckart Zitzler.
PPSN VI Tutorials

Pierre Bessiere, CNRS - IMAG, Leibniz laboratory, France
and Emmanuel Mazer, INRIA Rhône-Alpes & IMAG, France
Bayesian Robot Programming

David Wolfe Corne, University of reading, UK
Bioinformatics

Marco Dorigo, IRIDIA, Université Libre de Bruxelles, Belgium
An Introduction to Ant Algorithms and Swarm Intelligence

Kerstin Dautenhahn, University of Hertfordshire, UK
Socially Intelligent Agents - From Animals to Animats

Thomas English, The Tom English Project, USA
No Free Lunch: A Gentle Introduction to Conservation in Optimization and Learning

Dario Floreano, Swiss Federal Institute of Technology, Switzerland
and Stefano Nolfi, Institute of Psychology, C.N.R., Italy
Evolutionary Robotics

Pedro Larrañaga and Jose A. Lozano, University of the Basque Country, Spain
Optimization by learning and simulation of probabilistic graphical models

Zbigniew Michalewicz, University of North Carolina, USA
Modern heuristics and evolutionary computation: principles and current issues

Rolf Pfeifer, University of Zurich, Switzerland
Embodied Cognitive Science – Understanding intelligence
Colin Reeves, Coventry University, UK
Fitness Landscapes: A Guided Tour

Moshe Sipper, Swiss Federal Institute of Technology in Lausanne, Switzerland
Cellular Automata and Self-Replication

Darrell Whitley, Colorado State University, USA
Walsh Analysis, Schemata, Embedded Landscapes and No Free Lunch
PPSN VI Workshops

The CAM-brain Machine
Hugo de Garis, Brain Builder Group, Japan

Learning Classifier Systems
Wolfgang Stolzmann, Universität Würzburg, Germany,
Pier Luca Lanzi, Politecnico di Milano, Italy
and Stewart Wilson, University of Illinois at Urbana-Champaign, USA

Methodology
Jason Daida, in The University of Michigan, USA

Multiobjective Problem Solving from Nature
Joshua Knowles, University of Reading, UK

Parallel and Emergent Computation in Telecommunications
Martin Oates, University of Reading, UK

Real-World Applications
Rajkumar Roy, Cranfield University, UK

Speech as Adaptive Behaviour
Luc Steels, Vrije Universiteit Brussel, Belgium
and Didier Demolin, Université Libre de Bruxelles, Belgium
# Table of Contents

## Invited Papers

Interacting Trajectories in Design Space and Niche Space: A Philosopher Speculates About Evolution ........................................... 3  
  *A. Sloman*

Language as a Complex Adaptive System ..................................... 17  
  *L. Steels*

## Analysis and Theory of EAs

Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio .... 29  
  *E. Alba, J. M. Troya*

Efficiency and Mutation Strength Adaptation of the $(\mu, \mu_I, \lambda)$-ES in a Noisy Environment ......................................................... 39  
  *D. V. Arnold, H.-G. Beyer*

An Analysis of the Configuration Space of the Maximal Constraint Satisfaction Problem .............................................................. 49  
  *M. Belaidouni, J.-K. Hao*

On the Desired Behaviors of Self-Adaptive Evolutionary Algorithms ...... 59  
  *H.-G. Beyer, K. Deb*

Practical Implications of New Results in Conservation of Optimizer Performance ................................................................. 69  
  *T. M. English*

Large Deviations, Evolutionary Computation and Comparisons of Algorithms .............................................................................. 79  
  *O. François*

On the Choice of the Mutation Probability for the (1+1) EA ............... 89  
  *T. Jansen, I. Wegener*

The Genetic Code-Like Transformations and Their Effect on Learning Functions ............................................................................... 99  
  *H. Kargupta*

Perturbation Theory for Evolutionary Algorithms: Towards an Estimation of Convergence Speed .............................................. 109  
  *Y. Landrin-Schweitzer, E. Lutton*
Statistical Characteristics of Evolution Strategies ................. 119
Y. Matsumura, K. Ohkura, K. Ueda

Consensus Sequence Plots and Error Thresholds: Tools for Visualising the
Structure of Fitness Landscapes ........................................ 129
G. Ochoa

Experiments with Tuneable Fitness Landscapes ....................... 139
C. R. Reeves

Introducing a New Persistence Measure .............................. 149
O. Sharpe

An Analysis of Dynamic Severity and Population Size ................ 159
K. Weicker

Functions as Permutations: Regarding No Free Lunch, Walsh Analysis and
Summary Statistics ....................................................... 169
D. Whitley

**Genetic Programming**

Distributed Hybrid Genetic Programming for Learning Boolean Functions. 181
S. Droste, D. Heutelbeck, I. Wegener

Genetic Programming with Dynamic Fitness for a Remote Sensing
Application .............................................................. 191
C. Fonlupt, D. Robilliard

Genetic Programming Bloat without Semantics ....................... 201
W. B. Langdon, W. Banzhaf

Genetic Programming and Domain Knowledge: Beyond the Limitations of
Grammar-Guided Machine Discovery .................................... 211
A. Ratle, M. Sebag

Polymorphy and Hybridization in Genetically Programmed Networks . 221
A. Silva, A. Neves, E. Costa

Building Optimal Committees of Genetic Programs .................. 231
B.-T. Zhang, J.-G. Joung

**Scheduling**

Distributed Simulated Annealing for Job Shop Scheduling .......... 243
A. Albrecht, U. Der, K. Steinhöfel, C.-K. Wong

Anticipation in Dynamic Optimization: The Scheduling Case ........ 253
J. Branke, D. C. Mattfeld
Multirecombinated Evolutionary Algorithms for the Flow Shop Scheduling Problem .......................... 263
   S. C. Esquivel, F. Zuppa, R. H. Gallard

GA Based on the UV-Structure Hypothesis and Its Application to JSP ... 273
   K. Ikeda, S. Kobayashi

Neighbourhood Based Robustness Applied to Tardiness and Total Flowtime Job Shops ........................ 283
   M. T. Jensen

Solving Extended Hybrid-Flow-Shop Problems Using Active Schedule Generation and Genetic Algorithms ......................... 293
   M. Kreutz, D. Hanke, S. Gehlen

A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem ........................................ 303
   M. Vázquez, D. Whitley

Representations and Operators

An Empirical Study on GAs “Without Parameters” .................. 315
   Th. Bäck, A. E. Eiben, N. A. L. van der Vaart

Using Dynastic Exploring Recombination to Promote Diversity in Genetic Search .......................... 325
   C. Cotta, J. M. Troya

Adaptive Control of the Mutation Probability by Fuzzy Logic Controllers 335
   F. Herrera, M. Lozano

A Comparison of Two Representations for the Fixed Charge Transportation Problem .......................... 345
   J. Gottlieb, C. Eckert

Invariance, Self-Adaptation and Correlated Mutations and Evolution Strategies .......................... 355
   N. Hansen

Theoretical Analysis of Simplex Crossover for Real-Coded Genetic Algorithms .......................... 365
   T. Higuchi, S. Tsutsui, M. Yamamura

Applying Self-Organised Criticality to Evolutionary Algorithms .......... 375
   T. Krink, P. Rickers, R. Thomsen

Genetic Algorithms, Clustering, and the Breaking of Symmetry ............ 385
   M. Pelikan, D. E. Goldberg
Pruefer Numbers and Genetic Algorithms: A Lesson on How the Low Locality of an Encoding Can Harm the Performance of GAs 395
F. Rothlauf, D. E. Goldberg

Median-Selection for Parallel Steady-State Evolution Strategies 405
J. Wakunda, A. Zell

The Origination of Diversity by Adaptive Clustering 415
N. Walton, G. D. Smith

Symbiotic Combination as an Alternative to Sexual Recombination in Genetic Algorithms 425
R. A. Watson, J. B. Pollack

Co-evolution
Island Model Cooperating with Speciation for Multimodal Optimization 437
M. Bessaou, A. Pétrowski, P. Siarry

Optimizing through Co-evolutionary Avalanches 447
S. Boettcher, A. G. Percus, M. Grigni

P. den Dulk, M. Brinkers

A Game-Theoretic Approach to the Simple Coevolutionary Algorithm 467
S. G. Ficici, J. B. Pollack

The Number of People with Whom a Man Interacts 477
M. Kubo, H. Satoh, Y. Inoue, K. Uno, A. Namatame

NK-Landscapes as Test Functions for Evaluation of Host-Parasite Algorithms 487
B. Olsson

Towards Balanced Coevolution 497
J. Paredis

Spatial Games with Adaptive Tit-For-Tats 507
E. S. Tzafestas

Competitive Segmentation: A Struggle for Image Space 517
C. J. Veenman, M. J. T. Reinders, E. Backer

Constraint Handling Techniques
An Adaptive Algorithm for Constrained Optimization Problems 529
S. Ben Hamida, M. Schoenauer
Test-Case Generator TCG-2 for Nonlinear Parameter Optimisation
M. Schmidt, Z. Michalewicz

Solving CSP Instances Beyond the Phase Transition Using Stochastic Search Algorithms
L. Schoofs, B. Naudts

Noisy and Non-stationary Environments
Steady-State Evolutionary Path Planning, Adaptive Replacement, and Hyper-Diversity
G. Dozier

Optimization of Noisy Fitness Functions by Means of Genetic Algorithms Using History of Search
Y. Sano, H. Kita

Evolvable Hardware and Hardware Implementation of EAs
An Efficient Random Number Generation Architecture for Hardware Parallel Genetic Algorithms
M. Bright, B. Turton

An Integrated On-Line Learning System for Evolving Programmable Logic Array Controllers
Y. Liu, M. Iwata, T. Higuchi, D. Keymeulen

Combinatorial Optimisation
Selection and Reinforcement Learning for Combinatorial Optimization
A. Berny

Ant Colony Optimization for the Total Weighted Tardiness Problem
M. den Besten, T. Stützle, M. Dorigo

Adaptive Fitness Functions for the Satisfiability Problem
J. Gottlieb, N. Voss

Large-Scale Permutation Optimization with the Ordering Messy Genetic Algorithm
D. Knjazew, D. E. Goldberg

A Hybrid GA for the Edge-Biconnectivity Augmentation Problem
I. Ljubić, G. R. Raidl, J. Kratica

A Temporal Representation for GA and TSP
I. Mitchell, P. Pocknell
A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem ................................................................. 661
   T. Stützle, A. Grün, S. Linke, M. Rüttger

A Genetic Algorithm for VLSI Floorplanning ................................. 671
   C. L. Valenzuela, P. Y. Wang

Applications

Scalability and Efficiency of Genetic Algorithms for Geometrical Applications ........................................................................ 683
   S. van Dijk, D. Thierens, M. de Berg

Genetic Optimization of the EPR Spectral Parameters:
   Algorithm Implementation and Preliminary Results ......................... 693
   B. Filipič, J. Štrancar

Fitting Fluorescence Spectra with Genetic Algorithms ....................... 702
   J. A. Hageman, R. Wehrens, R. de Gelder, W. L. Meerts, L. M. C. Buydens

Real-Coded Adaptive Range Genetic Algorithm Applied to Transonic Wing Optimization ......................................................... 712
   A. Oyama, S. Obayashi, T. Nakamura

Stream Cyphers with One- and Two-Dimensional Cellular Automata .... 722
   M. Tomassini, M. Perrenoud

Machine Learning and Classifier Systems

Investigating Generalization in the Anticipatory Classifier System ....... 735
   M. V. Butz, D. E. Goldberg, W. Stolzmann

A New Bootstrapping Method to Improve Classification Performance in Learning Classifier Systems ............................................ 745
   J. H. Holmes, D. R. Durbin, F. K. Winston

Towards Automatic Domain Knowledge Extraction for Evolutionary Heuristics ................................................................. 755
   M. Jelasity

New Algorithms and Metaphors

Expanding from Discrete to Continuous Estimation of Distribution Algorithms: The IDEA ....................................................... 767
   P. A. N. Bosman, D. Thierens

A New Genetic Algorithms Working on State Domain Order Statistics ... 777
   D. Delahaye, S. Puechmorel
A Factorized Distribution Algorithm Using Single Connected Bayesian Networks ................................................................. 787
   A. Ochoa, H. Muehlenbein, M. Soto

Optimization as Side-Effect of Evolving Allelopathic Diversity ............ 797
   L. Pagie, P. Hogeweg

Reaction-Diffusion Model of a Honeybee Colony’s Foraging Behaviour .... 807
   V. Tereshko

A Religion-Based Spatial Model for Evolutionary Algorithms .............. 817
   R. Thomsen, P. Rickers, T. Krink

Bayesian Evolutionary Optimization Using Helmholtz Machines .......... 827
   B.-T. Zhang, S.-Y. Shin

Multiobjective Optimisation

The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimisation ................................................................. 839
   D. W. Corne, J. D. Knowles, M. J. Oates

A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II .................................................. 849
   K. Deb, S. Agrawal, A. Pratap, T. Meyarivan

Mechanical Component Design for Multiple Objectives Using Elitist Non-dominated Sorting GA ................................................ 859
   K. Deb, A. Pratap, S. Moitra

On the Assessment of Multiobjective Approaches to the Adaptive Distributed Database Management Problem ............................. 869
   J. D. Knowles, D. W. Corne, M. J. Oates

A Hierarchical Genetic Algorithm Using Multiple Models for Optimization 879
   M. Sefrioui, J. Périaux

EA Software

Take It EASEA .................................................................................. 891
   P. Collet, E. Lutton, M. Schoenauer, J. Louchet

Evolutionary Computation Visualization: Application to G-PROP ........ 902
   G. Romero, M. G. Arenas, J. G. Castellano, P. A. Castillo, J. Caprio,
   J. J. Merelo, A. Prieto, V. Rivas

Author Index .................................................................................. 913
Invited Papers
Interacting Trajectories in Design Space and Niche Space: A Philosopher Speculates About Evolution

Aaron Sloman
School of Computer Science, The University of Birmingham
A.Sloman@cs.bham.ac.uk, http://www.cs.bham.ac.uk/~axs/

Abstract. There are evolutionary trajectories in two different but related spaces, design space and niche space. Co-evolution occurs in parallel trajectories in both spaces, with complex feedback loops linking them. As the design of one species evolves, that changes the niche for others and vice versa. In general there will never be a unique answer to the question: does this change lead to higher fitness? Rather there will be tradeoffs: the new variant is better in some respects and worse in others. Where large numbers of mutually interdependent species (designs) are co-evolving, understanding the dynamics can be very difficult. If intelligent organisms manipulate some of the mechanisms, e.g. by mate selection or by breeding other animals or their own kind, the situation gets even more complicated. It may be possible to show how some aspects of the evolution of human minds are explained by all these mechanisms.

1 Introduction: design space and niche space

Evolution can be thought of as involving trajectories in two different but related spaces, design space and niche space. Co-evolution involves multiple parallel trajectories, with complex feedback loops. This paper attempts to explain these ideas and explore some of their implications, building on previous partial expositions ([18, 20, 22, 24, 28–30]).

1.1 What is a niche?

A biological niche is something that can produce pressure for evolutionary change. A niche is sometimes thought of as a geographical location, but since two types of organisms in the same location (e.g. a bee and a wasp flying close together) can have different niches, a niche must be something more abstract. It can be compared with what an engineer would call a ‘requirements specification’.

However a niche is more subtle than a standard set of engineering requirements, since the nature of the niche, i.e. the collection of requirements for an organism to function well, is not some fixed externally determined specification, but depends in part on the organism’s existing features and capabilities.
E.g. its digestive mechanisms partly determine the food it needs, and that determines behavioural capabilities required to obtain such food in a given environment, and that determines the kinds of knowledge the organism requires, or needs to be able to acquire, and so on.

Such considerations support the view of an organism as a collection of co-evolving sub-organisms, an idea also found in Popper’s work [14]. Minsky [11] refers to a ‘society’ of mind. From our viewpoint ‘ecosystem of mind’ is more apt. In particular, evolution of motivational mechanisms can produce diversity of tastes and preferences as well as skills, abilities, knowledge, etc.

There are thus niches corresponding to components of organisms. The engineering analogue might be a dynamically changing requirements specification for work still to be done on a partially designed and implemented system, where requirements for the remaining designs will depend on design decisions already taken.

An even closer analogue would be the set of requirements for possible improvements in parts of a complete design, on the assumption that other parts, and the environment remain fixed. Different niches are determined by which components and which aspects of the environment are held fixed.

1.2 What is a design (genotype)?

Designs, like niches, are also obviously abstract, since a design may have different instantiations, especially a design for a self-modifying system. Two instances of such a design may be very different in their structure and capabilities. Different instances of the same abstract design can also be viewed as instances of different more specific designs.

Development of an individual organism involves ‘moving’ through a space of specific designs. Different trajectories may start in the same region of design space and then diverge, either because of cumulative effects of very small initial differences or because of different environmental influences.

The instances of such designs will have ill-defined boundaries, since, as Dawkins [4] and others have pointed out, the genotype affects not only the individual’s physical and behavioural capabilities but also typical products of its behaviour, such as nests, tools, paths, furniture, etc. For our present purposes, it is not necessary to be precise about the boundaries of instances (or organisms).

2 Fitness relationships

Regions in niche space (niche types) are abstract collections of requirements, and regions in design space are sets of designs for types of behaving systems.

The instances of any particular design (genotype) will be capable of functioning more or less well in any particular region of niche space. However this does not mean that there is a numeric, or totally ordered, fitness function representing how well a design fits a niche.
If a class of designs can be specified by two parameters (e.g. length and stiffness of a spring), then there is a 2-D space of designs. Adding a measure of fitness of the spring for a particular purpose, produces a 3-D fitness landscape. Typically, design spaces are far more complex than this, and cannot be specified by a fixed number of parameters, e.g. designs for Prolog compilers vary in structure and complexity. Moreover, many designs have no single fitness measure: Prolog compilers vary according to their portability, the speed of compilation, the speed of compiled code, the size of compiled code, the kinds of error handling they support, etc.

So there is a multi-dimensional fitness relationship whose complexity will vary according to the design and the niche (Figure 1). An approximation to this is to regard a design and a niche as determining not a fitness value but a fitness vector whose components may be not only numerical values but descriptions ([8, 16]). E.g. one element of the vector linking a design for a predator with a niche involving certain types of prey might be something like: 'able to catch prey only in bright sunlight'.

Fig. 1. Design space, niche space and mappings between them

Relations between designs and niches are complex and varied. 'Fitness' vectors describing the relations, indicated by the arrows linking designs and niches, are required rather than fitness values. Different arrows represent different types of fitness relations. Trajectories are not shown here.

In general there will not be an answer to the question: does this change (e.g. a mutation) lead to higher fitness? Rather there will be tradeoffs: the new
variant is better in some respects and worse in others, like products reviewed in consumer magazines. This is not the same as a "neutral" change which makes no relevant difference to the individual's abilities to fit the niche.

Sometimes there is a partial ordering of the fitness descriptions, and sometimes not even that, because there is no way to combine the different dimensions of comparison. Design A might be better than B in one respect, B better than C in another and C better than A in a third.

Tradeoffs between dimensions in the fitness vector may be exploited by the formation of cooperative behaviours and division of labour. Two individuals that excel in different ways (e.g. hunting and farming) may together be more competent than two with equal but intermediate levels of expertise. This is particularly true of individuals in a social group requiring many kinds of expertise. Useful division of labour can also occur across species.

3 Constraints on trajectories

Some regions of design space are not linked by possible trajectories for individual development. An acorn can transform itself into an oak tree, and by controlling its environment we can slightly modify what sort of oak tree (e.g. how big). But no matter how you try to train or coax it by modifying the environment, it will never grow into a giraffe. The acorn (a) lacks information needed to grow into a giraffe, (b) lacks the architecture to absorb and use such information, and (c) lacks the architecture required to modify itself into an architecture that can absorb the information.

Trajectories that are possible for an individual which adapts or changes itself will be called i-trajectories. Different sorts of i-trajectories could be distinguished according to the sorts of mechanisms of change.

Trajectories that are not possible for an individual machine or organism but are possible across generations will be called e-trajectories (evolutionary trajectories). Examples include development of humans and other animals from much simpler organisms and modifications of software structures by genetic algorithms. Conjectured e-trajectories leading to human minds are discussed in [5] and [28]. If identical individuals inhabit slightly different niches, reproductive success will be favoured by different traits. E.g. in a farming community physical strength may be more important than intelligence, whereas in a nearby industrialised region intelligence is more useful for acquiring resources to raise a family. Thus different e-trajectories can be explored in parallel within a species.

Whether two designs are linked by an e-trajectory or not will depend on the type of evolutionary mechanism available for manipulating genetic structures and the type of ontogenetic mechanism available for producing individuals (phenotypes) from genotypes. In biological organisms the two are connected: the ontogenetic mechanism can also evolve. Lamarckian inheritance (used effectively in some evolutionary computation mechanisms) allows i-trajectories to form parts of e-trajectories.
There are also some changes to individuals that are possible only through external intervention by another agent, e.g. performing repairs or extensions, or debugging software. These are *r-trajectories* (repair-trajectories).

Viewing a species as a type of individual, *e-trajectories* for *individuals* can be thought of as *i-trajectories* for a *species*, or a larger encompassing system, such as an ecosystem.

When organisms are able to form societies within which information is transmitted across generations, there are kinds of changes in knowledge, preferences, skills that depend on social and cultural influences. So there are trajectories of a fourth kind *s-trajectories* (social-trajectories).

4 Discontinuous spaces and trajectories

Since designs and niches can vary in degrees and kinds of abstraction, the space of designs and the space of niches each has a very complex topology, with different neighbourhood relations at different levels of abstraction. At a particular level of abstraction a design can be modified to produce different designs at that level. *Neighbouring* designs are those reached by minimal changes in the design.

A specification mentioning the number of legs of an organism could have as a minimal change the addition of a leg (which is not necessarily biologically feasible). This might improve some aspects of fitness for a niche and degrade others. A more specific design, which includes details of the limbs, might allow minimal changes which retain the number of legs, but change aspects of their shape or other features.

A minimal change in a very specific niche for an organism might include an increase in speed of movement of its prey. A minimal change in a more abstract specification of the niche might include introduction of a new kind of predator, or a new type of prey.

We see therefore that there are trajectories in design space and niche space with different levels of abstraction, where the trajectories may include small changes or large changes, continuous changes or discontinuous changes. As one species evolves that will change the niche (the requirements) for others and vice versa, as in arms races.

Neither space is continuous, though there may be regions of continuity where requirements or designs vary only quantitatively: e.g. increasing speed of locomotion. However, Darwinian evolution is inherently discontinuous, since between any two generations there are only finitely many steps. This discreteness also follows from the structure of DNA.

Natural trajectories in the space of very specific designs are usually ‘gradual’ rather than continuous, as they involve many *small* but discrete steps. There can also be quite large steps, e.g. copying some complex structure or capability and then differentiating. In [9] Maynard Smith and Szathmáry refer to this as ‘duplication and divergence.’

Human designers, e.g. software engineers, aircraft designers, composers, poets, also often come up with big jumps in relation to prior designs.
4.1 Step size and searching

Small steps allow exhaustive local sampling of possible directions of change, but these may lead to unsatisfactory local optima. So larger steps are desirable. However, as the maximum step size grows, the number of options increases quickly (e.g. as the volume of an N-dimensional hypersphere increases with radius). Informed jumping without exhaustive search requires a grasp of the structure of the problem (the niche) and the region of design space. E.g. understanding why a two-legged chair doesn’t stay upright helps one choose to add a third leg rather than explore variations in length or thickness of the two legs, or varying the size of the seat, etc.

Without such problem-specific knowledge an evolutionary search must use general purpose heuristics. Alas, for any such set of heuristics there will be design spaces for which those heuristics fail to find optimal, or even good, designs to fit a particular niche. In finite spaces exhaustive search is always a last resort, but where the space is large enough that is of no practical relevance.

Biological evolution, however, is generally not a search for solutions to some pre-defined problem. It is more like a large sea of viscous fluid seeping into all the nooks and crannies in some complex terrain.

Where a particular portion of the fluid goes, will depend both on the local structure of the terrain and also on the pressure from surrounding portions of fluid. The fluid’s movement will change the structure of the terrain, sometimes catastrophically.

This model is partially inaccurate insofar as it suggests that the terrain, the niche space, is mostly fixed. This is not the case in co-evolution where the “terrain” through which each organism’s design moves is constantly changing because of changes in other organisms. Can we find more accurate model?

4.2 Co-evolutionary trajectories

If large numbers of mutually interdependent species (designs) are co-evolving, getting a good understanding of the dynamics of the various trajectories can be very difficult. When organisms become sufficiently intelligent to understand and manipulate some of the mechanisms, e.g. by breeding other animals, by mate selection, or by selectively feeding and protecting their own infants according to their qualities, the situation gets even more complicated.

5 Causation in niche space and design space

Since niches and designs interact dynamically, we can regard them as parts of virtual machines in the biosphere consisting of a host of control mechanisms, feedback loops, and information structures. All of these are ultimately implemented in, and supervenient on physics and chemistry. But they and their causal interactions may be as real as poverty and crime and their interactions. They are not epiphenomena.
The biosphere is a very complex abstract dynamical system, composed of many smaller dynamical systems. Some of them are evanescent (e.g. tornados), some enduring but changing over diverse time scales (e.g. fruit flies, oak trees, ecosystems). Many subsystems impose constraints and requirements to be met or overcome by other subsystems: e.g. one component’s design is part of another component’s niche.

Through a host of pressures, forces and more abstract causal relations, including transfer of factual information and control information, systems at various levels are constantly adjusting themselves or being adjusted or modified. Some of the changes may be highly creative, including evolution of new forms of evolution, and new mechanisms for copying and later modifying modules to extend a design.

These ideas may seem wild, but they are natural extensions of ideas already accepted by many scientists and engineers, e.g. [7, 2].

Niche-pressure can influence movement of individuals or a species in design space, e.g. via adaptation along an i-trajectory, or movement of a gene pool or a subset of a gene pool, along an e-trajectory. There are many different sorts of causal relations to be analysed, including causal loops. Niches can interact with one another by producing pressure for changes in designs, which in turn can change niches. Such feedback can lead to continual change, to oscillations, or to catastrophes.

Parallel design changes can also occur within an individual, e.g. making an organism both physically stronger and better able to recognize complex structures. Problems arise when the changes are not independent: e.g. increasing agility may conflict with increasing strength. Which of two incompatible changes actually occurs may depend on subtle features of the total context.

Since designs have complex structures, a niche for an organism can change simply because of a change within its own design, without anything changing in the environment. This can generate positive feedback loops driving designs along e-trajectories without any environmental changes.

Where independent changes in different dimensions are possible, causation may be conditional. In one context one change will be favoured, and a different change in another context. Such changes in different directions can happen in parallel in members of the same species in slightly different niches, possibly leading to useful functional differentiation.

Design space and niche space are “layered”: regions within them are describable at different levels of abstraction and for each such region different “specialisations” exist. Some specialisations of designs are called implementations. The philosopher’s notion of “supervenience” and the engineer’s notion of “implementation” (or realisation) seem to be closely linked, if not identical. There are many confusions about these relations still to be clarified ([15]).

Both are inhomogeneous spaces: local topology varies with location in the space, since the minimal changes possible at various locations in the same space can be very different in type and number. Consider designs of different complexity: there are typically more ways and more complex ways, of altering a complex
design than a simple design. So they have neighbourhoods of different structures. By contrast, in most multi-dimensional spaces considered by scientists and engineers (e.g. phase spaces), each point has the same number of dimensions, i.e. the same number and the same types of changes are possible at all points (unless limited by equations of motion).

5.1 Evolution of mind

Both design space and niche space have very complex topologies, including many discontinuities, some small (e.g. adding a bit more memory to a design, adding a new step in a plan) some large (adding a new architectural layer, or a new formalism). Understanding natural intelligence may require understanding some major discontinuities in the evolutionary history of the architectures and mechanisms involved. This in turn may help us with the design of intelligent artefacts.

Important types of discontinuity in design space occur between systems that are able merely to perform certain tasks reactively, perhaps with slight modifications due to learning, and others which can use generalisations they have learnt about the environment to create new plans, i.e. between reactive and deliberative architectures. Deliberative capabilities, supporting 'what if' reasoning, require a collection of mechanisms not needed for purely reactive systems.

Some of these evolutionary discontinuities may have occurred in e-trajectories where an old mechanism was copied then modified. E.g. a mechanism which originally associates sensory patterns with appropriate responses could be copied and the new version used to associate possible future sensory patterns with predicted sensory patterns or with a set of available responses.

Discontinuities might also be involved in the evolution of the "reflective" abilities described below: not only being able to do X but having and being able to use information on how X was done, or why X was done, or why one method of doing X was used rather than another. (Compare [22, 26, 30].) What sorts of niche pressures in nature might favour such e-trajectories is an interesting biological question.

6 Never-ending discontinuous i-trajectories

A system which develops, learns or adapts changes its design. I-trajectories, like e-trajectories can be discontinuous (e.g. cell division) and link regions in inhomogeneous spaces. The most familiar examples are biological: e.g. a fertilised egg transforming itself into an embryo and then a neonate. In many animals, including humans, the information processing architecture seems to continue being transformed long after birth, and after the main physiological structures have been established: new forms of control of attention, learning, thinking, deliberating, develop after birth. Ontogeny may partly recapitulate phylogeny: but cultural influences may change this.

Humans follow a very complex trajectory in design space throughout their lives. A good educational system can be viewed as providing a trajectory through
niche space which will induce a trajectory in design space in self-modifying brains. A culture provides a set of developmental trajectories.

In general, following a trajectory in design space also involves a trajectory in niche space: the niches for an unborn foetus, for a newborn infant, a schoolchild, a parent, a professor, etc. are all different. Moreover, an individual can instantiate more than one design, satisfying more than one niche: e.g. protector and provider, or parent and professor. To cope with development of multi-functional designs we can include composite niches in niche space, just as there are composite designs in design space.

7 Trajectories for Virtual Machines in Software Systems

The distinction between i-trajectories and e-trajectories can be applied to software individuals inhabiting virtual machines. A word processor which follows an i-trajectory in which it adapts itself to different users may or may not be capable of turning itself into a language understanding system through such a series of adaptations, even if evolutionary mechanisms could so transform it. As with organisms there may be e-trajectories linking software designs that are not linked by i-trajectories.

Whether an e-trajectory exists from one software design to another in an artificial evolutionary system depends on (a) whether there is a principled way of mapping the features of the designs onto genetic structures which can be used to recreate design instances via an instantiation (ontogenetic) function, and (b) whether the structures can be manipulated by the permitted operators so as to traverse a trajectory in “gene space” which induces a trajectory in design space via the instantiation function. Whether some sort of evaluation function or niche pressure can cause the traversal to occur is a separate question [13]. E-trajectories can exist which our algorithms never find.

8 Evolution of Human-like Architectures

We have argued in [19, 27] and elsewhere (contra Dennett’s “intentional stance”) that many familiar mental concepts presuppose an information processing architecture. We conjecture that it involves several different sorts of coexisting, concurrently active, layers, including an evolutionarily old “reactive” layer involving dedicated highly parallel mechanisms each responding in a fixed way to its inputs. These may come from sensors or other internal components, and the outputs may go to motors or internal components, enabling loops. Some reactive systems have a fixed architecture except insofar as weights on links change through processes like reinforcement learning. Insects appear to have purely reactive architectures implementing a large collection of evolved behaviours. As suggested in [23, 28], sophisticated reactive architectures may need a global “alarm” mechanism to detect urgent and important requirements to override relatively slow “normal” processes. This can interrupt and redirect other subsystems (e.g. freezing, fleeing, attacking, attending).
A hybrid architecture, as shown in Figure 2, could combine a reactive layer with a “deliberative” layer which includes the ‘what if’ reasoning ability needed to create new temporary structures representing alternative possibilities for complex future actions, which it can then compare and evaluate, using further temporary structures describing similarities and differences. This plan-construction requires a long term memory associating actions in contexts with consequences. After creating and selecting a new structure the deliberative system may execute it as a plan, and then discard it. ‘Skill compilers’ can use practice to save new plans within the reactive layer for future use in fast, fluent mode. This happens when humans learn to drive a car or speak a new language.

Global alarm mechanisms may be needed for coping with dangers and opportunities requiring rapid reactions. In mammals this seems to use a variety of old and new mechanisms, including the limbic system. Emotional processes are often involved [3, 6].

A deliberative mechanism will (normally) be discrete, serial, and therefore relatively slow, whereas a reactive mechanism can be highly parallel and therefore very fast, and may include some continuous (analog) mechanisms, possibly using thresholds. Resource limits in deliberative mechanisms may generate a need for an attention filter of some kind, limiting the ability of reactive and alarm mechanisms to interrupt high level processing.

By analysing tradeoffs we may be able to understand how niche-pressures can lead to development of combined, concurrent, deliberative and reactive architectures in certain organisms. Other types explored alternative trade-offs, e.g. with large numbers of simple and expendable individuals.

Everything that can be done by a hybrid architecture could in principle be done by a suitably complex reactive architecture e.g. a huge, pre-compiled lookup table matching every possible history of sensory inputs with a particular combination of outputs. However, pre-requisites for such an implementation may be prohibitive: much longer evolution, with more varied evolutionary environments, to pre-program all the reactive behaviours, and far more storage to contain them, etc. For certain agents the universe may be neither old and varied enough for such development nor big enough to store all the combinations required to match a deliberative equivalent with generative power. Perhaps evolution “discovered” this and therefore favoured deliberative extensions for some organisms.

A deliberative mechanism changes the niches for perceptual and motor mechanisms, requiring them to develop new layers of abstraction, as indicated in Figure 2. Likewise, development of new, higher level, abstractions in perceptual and motor systems may change the niches for more central mechanisms, e.g. providing new opportunities for learning and simplified planning.

Meta-management. Reflection on and retrospective evaluation of actions can often lead to future improvements. This is also true of internal actions. Thus besides abilities to perceive the environment and how external actions change it, there is a use also for internal self-monitoring, self-evaluation, self-modification (self-control) applied to internal states and processes. This could explain the evolution of a third architectural layer, as indicated in Figure 2.
Sensory “qualia” arise in self-monitoring mechanisms with access to intermediate sensory information structures not normally attended to. Different kinds of sensory qualia depend on different perceptual abstraction layers. Such “self-knowledge” is distinct from normal perception providing knowledge about the environment. “Meta-management” capabilities produce other sorts of qualia related to thinking processes, deliberation, desires, etc.

Robots with these capabilities might begin to wonder how their mental processes are related to their physical implementation, just as human philosophers do. Some of them, not fully understanding the notion of virtual machine functionality and the varieties of forms of supervenience, might even produce spurious but convincing arguments that they have conscious processes which cannot be explained by or fully implemented in physical processes. They may wonder whether humans are actually zombies with all the behavioural capabilities of conscious robots, but lacking their consciousness. I believe this solves the so-called “hard” problem of consciousness, see [1]. (Earlier papers exploring these ideas are in the bibliography.)

Such agents (with a combination of reactive, deliberative and self-management sub-architectures) may combine to form social systems. Questions about trajectories in design space and niche space arise for social systems also. Human
social systems, follow s-trajectories and develop information and rules which are transmitted to individuals, including rules that control meta-management (e.g. through guilt).

9 Conclusions and Further Work

Milner [10] noted that in computer science theory often follows the lead of engineering intuitions. The process of mathematical formalisation can lag far behind. Likewise attempts to study and formalise the space of possible designs and the various trajectories in design space will lag behind intuitive understanding gained from empirical research in biology, psychology, and computational explorations.

Many have attempted to formalise features of evolution, individual learning, development etc. Kauffman [7] describes mathematical laws which constrain biological mechanisms and processes in surprising ways. The ideas discussed here deal with phenomena which still seem to be too ill defined for mathematical formulation and computational modelling. However that may change.

We need to find more precise ways of describing architectures, designs, niches and their causal interactions, to improve on the high level concepts used only intuitively at present. This will involve both abstracting from domain specific details, so as to replace empirical concepts with mathematical concepts, and also enriching our understanding of the details of the processes, so that we can characterise and model the dynamics.

If the intuitive notions of niche, genotype etc. in biology can be made sufficiently precise to enable us to understand precisely the relationships between niches and designs for organisms, this may provide a better understanding of the dynamics and trajectories in biological evolution, including the evolution of evolvability.

This could lead to advances in comparative psychology. Understanding the precise variety of types of functional architectures in design space and the virtual machine processes they support, will enable us to describe and compare in far greater depth the capabilities of various animals. We’ll also have a conceptual framework for saying precisely which subsets of human mental capabilities they have and which they lack. Likewise the discussion of mental capabilities of various sorts of machines could be put on a firmer scientific basis, with less scope for prejudice to determine which descriptions to use. E.g. instead of arguing about which animals, which machines, and which brain damaged humans have consciousness, we can determine precisely which sorts of consciousness they actually have.

We could also derive new ways of thinking about human variability and the causes and effects of mental illness, brain damage, senile dementia, etc. This could have profound practical implications.

Finally it should be noted that designs do not presuppose a designer and requirements (niches) do not presuppose a requirer.
Notes and Comments
This work is partly funded by the Leverhulme Trust. My ideas have benefited from discussions with many colleagues in the School of Computer Science at the University of Birmingham and Brian Logan in the School of Computer Science & IT University of Nottingham. Most of the papers from our group are available at the Cognition and Affect project web site:

http://www.cs.bham.ac.uk/research/cogaff/

Our software tools for exploring agent designs are included with the Free Poplog software repository:

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

References


Abstract. The paper surveys recent work on modeling the origins of communication systems in groups of autonomous distributed agents. It is shown that five principles gleaned from biology are crucial: reinforcement learning, self-organisation, selectionism, co-evolution through structural coupling, and level formation.

1 Introduction

It is by now well accepted that we can discover new computational and problem solving paradigms by studying natural systems, particularly complex dynamical systems. This strategy was still a dream 15 years ago [11] but has borne rich fruits. Novel computational problem solving paradigms now exist based on analogies with spin glasses, genetic evolution, immune system dynamics, collective insect behavior, DNA, biochemical reactions, and last but not least neural networks. In each case, a set of primitive elements and their behavior is defined, an interaction is set up with the environment, and the collective behavior is studied that emerges from interactions between the individual elements and the environment. A specific problem is solved by mapping it into the initial states of a complex system and defining the dynamics such that a solution corresponds to one of its attractors. For example, the traveling salesman problem can be mapped onto the initial state of a spin-glass system, and after the spin-glass dynamics has operated, a final solution can be read from the resulting state. The same problem can also be mapped into DNA strings [1] or the initial states of other dynamical systems.

About 5 years ago, a number of researchers started to adopt this same strategy with respect to language (see [13] for review of earlier work). The basic idea is that a community of language users (further called agents) can be viewed as a complex adaptive system which collectively solves the problem of developing a shared communication system. To do so, the community must reach an agreement on a repertoire of forms (a sound system in the case of spoken language), a repertoire of meanings (the conceptualisations of reality), and a repertoire of form-meaning pairs (the lexicon and grammar). Communication is not a general computational problem of course (although neither is the traveling salesman) but nevertheless a problem of great interest.

First of all there is a strong interest from a scientific point of view. Finding the key how communication systems of the complexity of human natural languages emerge may help to solve the problem how human language itself may
have originated and evolved. This longstanding fascinating question is receiv­
ing increasing attention lately ([5], [6]), but only clear scientific models that explain how language evolved (as opposed to enumerating conditions why lan­guage evolved) can be expected to steer us away from the many speculations that made the field suspect for a long time. By clear scientific models I mean that the cognitive structures and interaction behaviors of each agent is specified and that it is shown how they collectively lead to a language.

Second, there is an interest because of possible applications. On the one hand, autonomous artificial agents which need to coordinate their activity in open-ended environments could make use of these mechanisms to develop and continuously adapt their communication systems [14]. On the other hand, understanding how language develops and evolves is probably our only hope to ever get to technological artefacts that exhibit human-level language understanding and production. Human languages are constantly changing and differ signifi­cantly from one speaker to the next and from one context to the next. So, we need language technologies which exhibit the same adaptivity as humans.

The rest of the paper reviews some of the experiments conducted so far. They always have the same form: (1) They involve a population of (artificial) agents, possibly robots. (2) The agents engage in interactions situated in a specific environment. Such an interaction is called a game. (3) Each agent has a sensori-motor apparatus, a cognitive architecture, and a script determining how it interacts with others. (4) There is an environment (possibly the real world) which consists of situations that are ideally open-ended. The situation in modeling the evolution of communication systems is different from that of using spin glasses or other natural dynamical systems. Spin glasses are much simpler systems and have been thoroughly studied in the natural sciences, whereas how humans acquire, interpret and produce language remains to a great extent a mystery.

2 Imitation games for the emergence of sound system

The work of De Boer [3] is one of the best examples how a repertoire of forms may become agreed upon in a distributed group of agents. This work focuses exclusively on the emergence of vowels. Clear universal tendencies exist for vowel systems [10] and it was already shown that they are due to functional and sensori-motor constraints [8]. The question being addressed in the new experiments is how agents can come to share a system of vowels without having been given a pre-programmed set nor with central supervision.

In the robotic simulations, the sensori-motor apparatus of the agents consists of an acoustic analyser on the one hand, which extract the first formants from the signal, and an articulatory synthesiser on the other hand. The agents play an imitation game. One agent produces a random sound from its repertoire. The other agent (the imitator) recognises it in terms of its own repertoire and then reproduces the sound. Then the first agent attempts to recognise the sound of the imitator again and if it is similar to its own, the game is a success otherwise
a failure. This setup therefore adopts the motor theory of perception whereby recognition of a sound amounts to the retrieval of a motor program that can reproduce it.

To achieve this task, the agents in the De Boer experiment use two cognitive structures: The vowels are mapped as points into a space formed by the first, second and third formants (see figure 1) and a nearest-neighbor algorithm is used to identify an incoming sound with the sounds already stored as prototypes. These prototypes have an associated motor program that can be used to reproduce the sound. When an imitation game succeeds, the score of the prototype goes up, which means that the certainty that it is in the repertoire increases. There are two types of failure. Either the incoming sound is nowhere near any of the sounds already in the repertoire. In that case it is added to the prototype space and the agent tries to find its corresponding motor program by a hill-climbing process, producing and listening to itself. Alternatively, the incoming sound is near an existing sound but the reproduction is rejected by the producing agent. This means that the imitator does not make sufficiently fine-grained distinctions. Consequently the failure can be repaired by adding this new incoming sound as a new prototype to the repertoire and associating it with a motor program learned again by hill-climbing. In order to get new sounds into the repertoire, agents occasionally "invent" a new sound by a random choice of the articulatory parameters and store its acoustic image in the prototype space. Sounds which have consistently low scores are thrown out and two sounds that are very close together in the prototype space are merged.

Quite remarkably, the following phenomena are perceived when a consecutive series of games is played by a population of agents: (1) A repertoire of shared sounds emerges through self-organisation (see figure 1). (2) The repertoire keeps expanding as long as there is pressure to do so. (3) Most interestingly, the kinds of vowel systems that emerge have the same characteristics as those of natural vowel systems. The experiment therefore not only shows that the problem can be solved in a distributed fashion but also that it captures some essential properties of natural systems.

Three principles have been used: Reinforcement learning [18] based on feedback after each game. It is used to reinforce a vowel in the repertoire of an individual agent or dismiss it. Reinforcement learning in itself does not explain however how the group arrives at a shared solution. There is a second principle at work: self-organisation. Self-organisation (in the sense of Nicolis and Prigogine [9]) arises when there is a positive feedback loop in an open non-linear system. Concretely there is a positive feedback between use and success. Sounds that are (culturally) successful propagate. The more a sound is used the more success it has and it will be used even more. Self-organisation explains that the group reaches coherence, but not why these specific vowels occur and not others. For this we need a third principle, namely selectionism. The scores of vowels that can be successfully distinguished and reproduced given a specific sensori-motor apparatus have a tendency to increase and they hence survive in the population. Novel sounds or deviations of existing sounds (which automatically get pro-
Fig. 1. Example of the evolution of a vowel system. Vowels are represented in formant space (first and second formant). We see that progressively coherence as well as increased complexity emerges.

duced due to the unavoidable stochasticity) create variation, and sensori-motor constraints select those that can be re-produced and recognised. The closer we can model human natural sensori-motor behavior the more realistic the vowel systems become.

3 Discrimination games for evolving meaning repertoires

Another series of experiments has demonstrated how a meaning repertoire may emerge in a population. Once again, a population of agents is defined which play a consecutive series of games. The games are typically discrimination games. An agent perceives some part of reality, for example through a camera and low level segmentation and feature detection, and selects one of the objects (more precisely segments in the visual image) as the topic. The agent then tries to distinguish the topic from the other objects in the context, for example by finding a category, or a logical combination of categories, that is valid for the topic but not for the other objects in the context. Thus suppose the scene contains a red triangle to the left of the image, a green square to the right, and a red square above it. Suppose that the red triangle has been chosen as topic, then possible distinctive features are: red, triangle, object to the left, or a conjunctive combination of them.

In the experiments reported in [17] the agents start from a visual image captured by a camera. The cognitive structure being used consists of discrimination trees for every sensori-motor channel available to the agent (figure 2). The tree divides a sensory channel up into finer and finer subregions. New divisions are generated by randomly selecting a channel and making a further subdivision, somewhat like leaves growing on a tree in a random fashion. Data about each segment in the scene falls into a subregion on each of the channels and a distinctive subregion (and hence a category) is identified if the data of the topic falls into it but not the data of any other object in the scene. Each category has a score reflecting how much it has been used successfully. When categories have no
success they are eventually pruned. Experiments with robotic agents confronted with real world environments have shown that a stable repertoire of categories builds up. The categorial repertoires are not necessarily identical in each agent and they continue to expand or contract when the environment changes.

Fig. 2. The discrimination trees developed by two physically embodied agents (left) and (right). The top of the figure shows the trees after playing 100 games and bottom after 200 games.

Other similar experiments use other kinds of categorial mechanisms (for example prototypes or adaptive subspaces [4]. What is important therefore is not the specific learning mechanism, but rather the idea of organising the repertoire formation process in terms of a series of games. We see that some of the same mechanisms have been put to work as for emergent sound systems: Reinforcement learning, because the environment gives feedback on which categories are distinctive and this is used to maintain or eliminate them, and selectionism, because the spontaneously generated categorial distinctions undergo selection pressure from the environment and the discrimination task itself. There is no self-organisation because each agent constructs individually his own repertoire which will only be similar to the extent that the agents are in the same environment and use the same sensori-motor apparatus.

4 Naming games for form-meaning repertoires

Other work has focused on how a shared set of form-meaning relations could collectively be built up by a population of agents (see e.g. [5], [12]). Once again
there is a population of agents. They play naming games. The agents have two components in their cognitive architecture: A mechanism for categorisation, such as the discrimination trees discussed earlier, and a lexicon which consists of a two-way associative memory storing form-meaning pairs. The agents do not have a general overview nor can they inspect each others' lexicons.

When the speaker has a meaning to express, he looks up in his lexicon what the preferred word is. The hearer uses his own lexicon to retrieve the most expected meaning. The game succeeds when the meaning retrieved by the hearer is compatible with that of the speaker. There is lateral inhibition: the scores of the associations that were used go up and those of competing associations go down. When the game fails, scores of participating associations go down. In some simulations, speaker and hearer have access to each other's meanings, which violates the "no-telepathy" assumption. In other experiments, this is not the case and speaker and hearer only indirectly can learn whether the same meaning was used. This introduces additional difficulties such as word-sense ambiguity (one word with multiple meanings) in addition to synonymy (one meaning multiple words).

Figure 3 (from [15]) gives a typical example of experimental results in which an increasing population of distributed agents collectively create a shared lexicon by playing naming games (and discrimination games as discussed in the previous subsection). The agents in this case are robots perceiving geometric scenes in the form of colored geometric figures pasted on the white board in front of them. We clearly see a winner-take-all situation in which one word dominates after a struggle against alternatives. We also see that word-sense ambiguity gets damped. This is more difficult because often more than one meaning is compatible with the same situation and agents have to wait until a situation arises that disambiguates the word-meaning relation.

Analysing this experiment we see the same three principles as in earlier experiments. There is clearly again a form of reinforcement learning which is now based on success or failure in the language game as a whole. The game succeeds if the hearer identifies the object that the speaker had in mind. There is self-organisation due to the positive feedback loop between use and success: When a word-meaning pair is successful, the score goes up. Agents prefer the word that has the highest score for producing and interpreting, hence words that have more success will be used and this leads again towards greater success. Selectionism also plays a role here because the sensori-motor apparatus of the agent and the situations they encounter acts as selectionist forces. For example, a word-meaning pair for which the meaning cannot reliably be recognised by the hearer will have less chance of survival.

A fourth principle, also widely used in biology, plays an additional role now, namely structural coupling (Maturana and Varela) leading to co-evolution between meaning repertoires and lexicons. Specifically a meaning formation repertoire (such as the one discussed in the previous subsection) is coupled to the lexicon formation repertoire in two ways: New categories may be generated at any time to be successful in discrimination. This obviously influences what word-
meaning pairs may arise. But conversely, success in the language game influences the score of the categories used, so the categorisation process becomes tuned to be better adapted to the language in the environment of the agents. This way the ontologies of the different agents become similar even though there is no telepathy and it is not innately given.

5 Experiments in the origins of grammar

Several researchers, most notably Batali [2], Kirby [7] and Steels [16],[17] have been conducting experiments to explain how languages with the grammatical complexity of human natural languages may emerge. This requires a scale-up along all dimensions (form, meaning, and form-meaning association) and it is therefore not surprising that many open questions remain. I briefly discuss experiments conducted by Batali [2] as a representative example.

The experiment once again starts by setting up a population of agents. They have a cognitive architecture which consists of a repertoire of meaning structures and a grammar able to relate (structured) meanings with expressions that have a syntactic structure. The computational and learning mechanisms used by Batali are based on recurrent neural networks but other types of learning such as memory-based learning or grammar induction could equally well be used. The agents play a language game in which the speaker conceives of a meaning, uses his grammar to translate that to a form, and then transmits that to the hearer.
The hearer parses the form and interprets it into a possible meaning. In the Batali experiment the (unrealistic) assumption is made that speaker and hearer share meaning independent of language, but other experiments (such as [17]) do not make this assumption and agents only get indirect feedback whether the meaning they guessed was the right one. If the game fails (the meanings are not equal), the networks of the hearer adapts to be more successful in future games. Batali has shown that syntactic structures indeed emerge when a consecutive series of games is played. The syntactic structures are surely not of the same complexity as syntactic structures found in human languages and the grammar does not exhibit the sometimes very regular systematicity found in natural languages, but nevertheless compositionality is clearly visible: new sentences could be constructed by the combination of parts built up from earlier sentences.

The Batali experiment (and other experiments in emergent grammar such as the ones reported in [16]) use the same principles as discussed in earlier experiments: reinforcement learning by the individual to tune in to the conventions present in the population, self-organisation based on a positive feedback loop between use and success to get coherence, selectionism constrained by the environments, the sensori-motor apparatus and the cognitive architecture of the agents, and co-evolution between syntax and semantics through structural coupling. But the study of grammar also introduces a new phenomenon, namely level formation, resulting in hierarchical structures and compositionality. Levels form because partial structures can be reused and thus form stable islands within larger structures. Level formation is also found in many biological systems. For example, when a symbiotic relation develops between organisms they may evolve into a dependent relationship leading to a new higher level organism.

6 Conclusions

Although we are only at the beginning of the evolutionary approach to linguistics briefly sketched in this paper, it is already quite clear that some general principles are emerging to understand how a group of distributed agents might autonomously generate communication systems of the complexity of human natural language. These principles are: reinforcement learning, self-organisation, selection, co-evolution through structural coupling, and level formation. It is not surprising that all these principles have been inspired by biology. The view that emerges from this research is that language can best be seen as a living system that is continuously evolving and adapting in a cultural process based on the distributed activity of its users. Consequently the computational investigations into genetic evolution, ant path formation, neural networks, and other biological systems are an important source of insight. This view is in stark opposition to the Chomskyan approach to linguistics, which suggests that language is a largely innate static abstraction uniformly present in the population. The paradigm shift implicit in the work reported here is as profound and important as the shift in biology from typological thinking to the population thinking that started the Darwinian revolution.
Acknowledgement

This work was supported by the Sony Computer Science Laboratory and by a GOA grant to the VUB AI Lab. I think the members of Sony CSL, particularly Angus McIntyre, Frederic Kaplan, and Pierre-Yves Oudeyer as well as members of the VUB AI Lab, particularly Edwin de Jong, Bart de Boer, Joris Van Looveren, and Paul Vogt for many discussions and participation in the experiments.

References


Analysis and Theory of EAs
Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio

Enrique Alba and José Mª Troya

Dpto. Lenguajes y Ciencias de la Computación
E.T.S.I. Informática, Campus de Teatinos (3.2.12)
29071, Málaga, Spain
eat@lcc.uma.es

Abstract. Spatially structured evolutionary algorithms (EAs) have shown to be endowed with useful features for global optimization. Distributed EAs (dEA) and cellular EAs (cEA) are two of the most widely known types of structured algorithms. In this paper we deal with cellular EAs. Two important parameters guiding the search in a cEA are the population topology and the neighborhood defined on it. Here we first review some theoretical results which show that a cEA with a 2D grid can be easily tuned to shift from exploration to exploitation. We initially make a study on the relationship between the topology and the neighborhood by defining a ratio measure between them. Then, we encompass a set of tests aimed at discovering the performance that different ratio values have on different classes of problems. We find out that, with the same neighborhood, rectangular grids have some advantages in multimodal and epistatic problems, while square ones are more efficient for solving deceptive problems and for simple function optimization. Finally, we propose and study a cEA in which the ratio is dynamically changed.

1 Introduction

Evolutionary algorithms (EAs) are optimization techniques for searching complex spaces for an optimum [4]. Normally, EAs use a single population (panmixia) although there exists a long tradition in using structured populations, especially for parallel implementations. Among the most widely known types of structured EAs, distributed and cellular algorithms are popular optimization tools [1].

Decentralizing a single population can be achieved by partitioning it in several sub-populations where island EAs are run with sparse string exchanges (distributed EAs), or in the form of neighborhoods (cellular EAs). Let us see these three EA types in Fig. 1. In distributed EAs, additional parameters controlling when migration occurs and how migrants are selected/incorporated in the target islands are needed. In cellular EAs the existence of overlapped small neighborhoods helps in exploring the search space [12]. These two kinds of EAs provide a better sampling of the search space and improve the numerical and runtime behavior of the algorithm in many cases [2].
The main difference of a cEA with respect a panmictic EA is its decentralized selection, since the reproductive loop is performed inside each of the numerous string pools. In a cEA, one given string has its own pool defined by neighboring strings, and at the same time, one string belongs to many pools. This 2D structure with overlapped neighborhoods is used to provide a smooth diffusion of good solutions along the grid. We directly focus on a grid of strings due to its generality [10].

The importance of cellular EAs is growing for several reasons. First of all, they are endowed of an internal spatial structure that allows fitness and genotype diversity for a larger number of iterations [12]. Also, some works established the advantages of using cEAs for complex optimization tasks (high efficacy and reduced number of steps) in relation with other EA models [5] [9]. Finally, the similitude of cEAs and cellular automata [13] [15], their potential applications, and their suitability for being implemented in SIMD machines make them worth of study. We analyze these algorithms as a kind of 2D spatially structured EAs run sequentially. Using a SIMD machine would have allowed for a much quicker execution, but the search is basically the same.

Pervious works have suggested the higher efficacy obtained when using cEAs in which the grid is not perfectly square i.e., thin grids [3]. See also the results of [9] for large TSP problem instances or [5] [8] for function optimization. They suggest (but not study) that the shape of the grid influences the quality of the search.

As we will show, the size, shape, and neighborhood defined for the grid have a major impact on the behavior of the cEA. Of course, other factors such as the operators and application rates influence the search, but the same holds for any other EA model. However, decentralized selection is determined by the topology and by the neighborhood; and decentralized selection is by far the main feature of a cEA [11].

The contribution of this work is to get a deeper insight into the importance of the relationship between topology and neighborhood. We use the ratio of the cEA [10] to quantify this relationship, and then study both a static and a dynamic ratio definitions. This study is presented in connection with the kind of problem being addressed, and not generically, in the research spirit inspired from the No Free Lunch theorem [16].

The paper is organized as follows. In the next section we characterize the shape and neighborhood of the cEA. Section 3 describes the problem domains we will use in the study. In Section 4 we present the comparative results of using cEAs with different ratios in a problem-dependent manner, and then extend this study to the case of changing the ratio during the search. Finally, Section 5 provides some conclusions.
2 Characterizing the Cellular EA

In this section we include a pseudo-code description of the canonical cEA and explain the algorithm. The EA we are using as a case study is a cellular GA (cGA) (see Fig. 2). The population is structured in a toroidal 2D grid and the neighborhood defined on it -line 5- always contains 5 strings: the considered one (position(x,y)) plus the north, east, west, and south strings -NEWS-. Fitness proportional selection is used in the neighborhood (Local_Select) -lines 5, 6, 7- along with a two point crossover operator that yields only one child (the one having the larger portion of the best parent: DPX1 -lines 8, 9-), and a traditional bit-flip mutation -line 10-.

In our monoprocessor implementation of this cGA, the successive populations replace each other completely -line 16-. Meanwhile, the new strings generated by local selection, crossover, and mutation are placed in a temporal population -lines 12, 13-. This replacement step -lines 12, 13- could be implemented by using the old population (e.g. replacing if the new string is better than the old one) or not (always adding the new string to the next population). The first issue (replacement by binary tournament) is the preferred one [3]. Computing basic statistics -line 17- is rarely found in the pseudo-codes of other authors. However, it is necessary for the work and monitoring of the algorithm. The same holds for the step that computes the list of neighbors of a given string with coordinates (x,y) -line 5-. This operation is needed whenever it is planed to change the type of neighborhood (although we do not change it here).

After presenting the algorithm, we now characterize the grid of our cGA. We extend the work in [10] to define a “radius”. The basic difference here is that the grid is not forced to be square. We consider the grid to have a radius equal to the dispersion of n* points in a circle centered in (x,y) (Equation 1). This definition also provides a radius value for the neighborhood. As shown in [10], the grid-neighborhood relationship can be quantified by the relative ratio between their radii (Equation 2). Algorithms with similar ratio show a similar selection pressure.

PSEUDO CODE OF A SIMPLE CELLULAR GA

```
1. proc Steps_Up(cga):
2.   for s←1 to MAX_STEPS do
3.     for x←1 to WIDTH do
4.       for y←1 to HEIGHT do
5.         n_list ←Compute_Neigh(cga,position(x,y));
6.         parent1←Local_Select(n_list);
7.         parent2←Local_Select(n_list);
8.         DPX1(cga.Pc,n_list[parent1],n_list[parent2],
9.           aux_ind.chrom);
10.        Mutate(cga.Pm,aux_ind.chrom);
11.        aux_ind.fit←cga.Fit(Decode(aux_ind.chrom));
12.        Insert_New_Ind(position(x,y),aux_ind,
13.           [if_better|always],cga,aux_pop);
14.     end_for;
15.   end_for;
16.   cga.pop←aux_pop;
17.   Update_Statistics(cga);
18. end_for;
19. end_proc Steps_Up;
```
\[ \text{rad} = \sqrt{\frac{\sum (x_i - x)^2 + \sum (y_i - y)^2}{n^*}} \]

\[ x = \sum_{i=1}^{n^*} x_i \quad y = \sum_{i=1}^{n^*} y_i \]  

(1)

\[ \text{ratio}_{cGA} = \frac{\text{rad}_{\text{neighborhood}}}{\text{rad}_{\text{topology}}} \]  

(2)

Although it is called a “radius”, \( \text{rad} \) measures the dispersion of \( n^* \) patterns. Other possible measures for symmetrical neighborhoods like the \( \text{radius} \) of a circle surrounding a rectangle containing the neighborhood or an \textit{asymmetry coefficient} would allocate the same numeric value to different neighborhoods (which is undesirable).

When solving a given problem with a constant number of individuals \((n=n^*)\) the grid radius will increase as the grid gets thinner (Fig. 2b). Since the \textit{neighborhood} is kept constant in size and shape throughout this paper (we always use NEWS, Fig. 2a), then the thinner the grid the smaller the resulting ratio.

\[ \text{rad}_1, \text{rad}_2, \text{rad}_{\text{linear}5} = \sqrt{\frac{2^2 + 2}{3 + 2}} = 0.8944 \quad \text{rad}_2 > \text{rad}_1 \quad \& \quad \text{ratio}_2 < \text{ratio}_1 \]

(a) (b)

Fig. 2. Neighborhood NEWS -linear 5- and its radius (a), and a square and thin grids of \((5\times5=25\) and \(3\times8=25\)) equal number of individuals with two different radii (b).

Reducing the ratio means reducing the global selection intensity on the population, thus promoting exploration. This is expected to allow for a higher diversity in the algorithm that improves the results in difficult problems (e.g., multimodal and epistatic problems). On the other hand, the search performed inside each neighborhood is guiding the exploitation of the algorithm. We explore in this paper how this ratio effects on the search efficiency over a variety of domains. Changing the ratio during the search is a unique feature of cEAs that can be used to shift from exploration to exploitation at a minimum complexity without introducing just another new algorithm.

3 Test Suite

In this section we explain the optimization tasks addressed in the paper. The problems used for this study are MMDP, FMS and P-PEAKS, representing three large classes of difficulty with interest in evolutionary computation namely: deception, multimodality, and epistasis. All of them are sought for a maximum (maximization).
The massively multimodal deceptive problem (MMDP) has been specifically designed to be difficult for an EA [7]. MMDP is made up of $k$ subproblems of 6 bits each one. The optimum has a value of $k$ and it is attained when every subproblem is composed of zero or six ones (unitation). Every subproblem contributes to the fitness according to its unitation (Fig. 3). The number of local optima is quite large (22^k), while only 2^k are global solutions. Thus, the degree of multimodality is defined by $k$. In this paper we are using a considerably large instance of $k=40$ subproblems.

$$f_{\text{MMDP}}(\vec{x}) = \sum_{i=1}^{k} \text{unitation}(x_i)$$

Fig. 3. Tabulated (a) and graphic (b) aspects of the basic 6-bit bipolar deception.

The Frequency Modulation Sounds (or FMS) parameter identification problem [14] has been proposed as hard real task consisting in adjusting a general model $y(t)$ to a basic sound function $y_0(t)$. The goal is to minimize the sum of square errors given by

$$f_{\text{FMS}}(\vec{x}) = \sum_{i=0}^{100} (y(t) - y_0(t))^2$$

The problem is to evolve 6 parameters $\vec{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ in order $y(t)$ to fit the target $y_0(t)$. The evolved and target models have the following expressions:

$$y(t) = a_1 \sin[w_1 t \theta + a_2 \sin(w_2 t \theta + a_3 \sin(w_3 t \theta))]$$
$$y_0(t) = 1.0\sin(5.0t\theta - 1.5\sin(4.8t\theta + 2.0\sin(4.9t\theta)))$$

In the above equations $\theta=2\pi/100$, and each real parameter is encoded with 32 bits in the range -6.4 to +6.35. The resulting problem is a highly complex multimodal function having strong epistasis with minimum value in $f_{\text{FMS}}=0.0$. We stop the algorithms when the error falls below 10^-2.

The last optimization task solved in this paper is a problem generator proposed in [6]. A problem generator is an easily parameterizable task which has a tunable degree of epistasis, thus allowing to derive instances with growing difficulty at will. Also, using a problem generator removes the opportunity to hand-tune algorithms to a particular problem, therefore allowing a larger fairness when comparing algorithms. With a problem generator we evaluate our algorithms on a high number of random problem instances, thus increasing the predictive power of the results for the problem class as a whole. Here we utilize the multimodal generator we call P-PEAKS [6].
The idea is to generate a set of $P$ random $N$-bit strings that represent the location of the $P$ peaks in the search space. To evaluate an arbitrary bit string, first locate the nearest peak in Hamming space. Then the fitness of the bit string is the number of bits the string has in common with that nearest peak, divided by $N$ (Equation 7). Problems with a small/large number of peaks are weakly/strongly epistatic. Our instance uses $P=100$ peaks of $N=100$ bits each, which represents a medium-high epistasis level.

$$f_{P-PEAKS}(x) = \frac{1}{N} \max_{i=1}^{P} \left\{ N - \text{HammingD}\left(x, \text{Peak}_i\right) \right\}$$ (7)

4. Analysis of the Performance with Different cEA Ratios

We now proceed to present and analyze the results of our cGA when solving these three different problems with different ratios. In all the problems we have used the same parameterization (summarized in Table 1).

<table>
<thead>
<tr>
<th>Table 1. Parameterization of the cGA for solving the problems MMDP, FMS, and P-PEAKS.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Popsize</strong></td>
</tr>
<tr>
<td><strong>String Length (L)</strong></td>
</tr>
<tr>
<td><strong>Parent's Selection</strong></td>
</tr>
<tr>
<td><strong>Crossover</strong></td>
</tr>
<tr>
<td><strong>Bit Mutation</strong></td>
</tr>
<tr>
<td><strong>Replacement</strong></td>
</tr>
</tbody>
</table>

The selected techniques and parameter values are very usual in EA's in order to get results widely useful for the research community. We use 400 individuals (of bit length $L$), proportional selection of two parents within the neighborhood of 5 strings, double point crossover with probability 1.0, and bit-flip mutation with probability $1/L$. The exception is for FMS, because it is very difficult to achieve an error below $10^{-2}$ with a mutation rate of $1/L$; thus, we have used the larger value $10/L$ for it.

For every point in the grid, we replace it only if the new string is better than the existing one. The stopping condition for all the algorithms and problems is to find a solution. We analyze the cost of solving a problem by measuring the number of function evaluations. For every problem, the same cGA is applied in which we only change the ratios between neighborhood and topology. Three clearly different ratios are used corresponding to the grid shapes 20×20, 10×40, and 4×100 individuals. The computed ratio values with NEWS is 0.110, 0.075, and 0.031 respectively (Table 2).

<table>
<thead>
<tr>
<th>Table 2. Three different grid shapes, their radius, and their ratio with NEWS.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shape</strong></td>
</tr>
<tr>
<td>20×20</td>
</tr>
<tr>
<td>10×40</td>
</tr>
<tr>
<td>4×100</td>
</tr>
</tbody>
</table>
Besides studying the impact of using a given single ratio during the search, we have also analyzed two cGAs having dynamic ratios, that is, a change in the ratio from its initial value to a different one while the cGA is still running. One cGA varies its ratio to shift from exploitation (0.110) to exploration (0.031), and conversely, the other cGA varies its ratio to shift from exploration (0.031) to exploitation (0.110).

The change in the ratio (shape) of the grid is performed in the "middle" of a "typical" execution \( t_m \). We choose the middle of a typical run \( t_m \) as located in half the mean number of evaluations needed by the square shape (ratio 0.110) to solve the problem in hands (Fig. 4). Thus, \( t_m \) is problem-dependent. In short, we have five different algorithms for solving each problem, each one performing the same cEA. Three cEAs have different static ratios, and two more algorithms use dynamic ratios.

All five algorithms then utilize a different kind of ratio. In Fig. 4 we plot the shape used in every case: a constant shape (first three cases above) and two kinds of shape change. Notice in Fig. 4 that the shape does not relate to the number of generations that it is used, although someone could erroneously think that the narrow shape seems to span across more generations than a square one.

The results are shown in Table 3. We summarize the average results of 30 independent runs for solving the three problems. If we first analyze the case of using a static ratio (the first three columns) we can see that reducing the ratio is more efficient in FMS and P-PEAKS. For irregular (FMS) and, especially, for highly epistatic problems (P-PEAKS) the reduced selection pressure of a smaller ratio improves exploration. However, for the MMDP problem the higher pressure of the square grid (0.110) is more efficient. We note in bold the more efficient results in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>Static Ratio 0.110 (20×20)</th>
<th>Static Ratio 0.075 (10×40)</th>
<th>Static Ratio 0.031 (4×100)</th>
<th>Dynamic Ratio: 0.110 to 0.031 (20×20 to 4×100)</th>
<th>Dynamic Ratio: 0.031 to 0.110 (4×100 to 20×20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMDP</td>
<td>170637.9</td>
<td>237003.4</td>
<td>331719.6</td>
<td>1002900.0</td>
<td>1002900.0</td>
</tr>
<tr>
<td>FMS</td>
<td>612112.1</td>
<td>594093.9</td>
<td>593519.1</td>
<td>692205.2</td>
<td>575353.8</td>
</tr>
<tr>
<td>P-PEAKS</td>
<td>50458.1</td>
<td>50364.6</td>
<td>48653.6</td>
<td>57769.7</td>
<td>48012.1</td>
</tr>
</tbody>
</table>

Figure 4. Static and dynamic ratio cEAs yielding 5 different algorithms. From top to bottom: a square grid, a rectangular grid, and a narrow grid, kept constant throughout the search. The two last cases show two cEAs that change from high (low) to low (high) ratio at generation \( t_m \).

Table 3. Mean Number of Evaluations.
These results with a static ratio confirm our previous work [3] in which a high ratio is more efficient in the absence of epistasis, and also for a medium degree of multimodality. A small ratio is desirable when epistasis or multimodality are high (P-PEAKS and FMS), because it speeds up the search and quickly gets out of local optima, since the grid is mapped to different search regions sought at the same time.

As to performing a change in the ratio during the search, we can see that changing from a large to a small ratio is undesirable (0.110 to 0.031) in all the studied problems. If we assume exploitation to be related to a high ratio (square grids) and exploration to be related to a small ratio (narrow grids) we can explain this result. On the contrary, we might thought that the complementary change in the ratio (from 0.031 to 0.110) will lead to a better or equal performance than using a static ratio in the domains where using a small ratio is already worth.

In effect, changing the ratio in this last way improves the efficiency when solving P-PEAKS and also it gives a new better efficiency for FMS (see Table 3). We cannot expect dynamic ratios to be desirable for all problems (NFL theorem) [16], because in fact the dynamic ratio e.g., in MMDP performs poorly: we had to stop the algorithm after a million evaluations without finding a solution. For non-epistatic problems a static square grid (high ratio) seems to be the better (initial) choice.

Since we want to be sure that these results are statistically significant, Tables 4, 5 and 6 contain the results of performing t-tests on the average results of Table 3. We consider a 0.05 level of significance, although slightly worse values like 0.06 and 0.08 are considered to indicate a significant trend in the analyzed result.

If we take a problem-dependent interpretation of the results we can see that a dynamic ratio is statistically worse than a static ratio for MMDP (see Table 4). It is clear for this deceptive problem that a high pressure (ratio) works out a better efficiency.

When solving a real problem with a large number of irregularities and local optima (FMS), a reduced ratio seems better than a large one (Table 3), although none of the static ratios can be statistically said to have advantages over the two others (see Table 5). A dynamic ratio cGA is worse when the pressure is high at the beginning and low at the end (0.110 to 0.031). This is common sense in evolutionary algorithms: exploitation plus exploration are not the normal phases of an EA. Conversely, the change from low to high ratio (0.031 to 0.110) has improved the results in FMS. This improvement in efficiency is clear with respect the other dynamic cGA, although we cannot ensure a statistical significance over the static ratios (see in Table 5 the associated p-values). In general, the FMS problem only allows us to show the superiority of enlarging (and not of diminishing) the ratio during search.

Table 4. MMDP: t-test indicating the significance of the results (p-values).

<table>
<thead>
<tr>
<th>p-values</th>
<th>0.110</th>
<th>0.075</th>
<th>0.031</th>
<th>0.110 / 0.031</th>
<th>0.031 / 0.110</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.110</td>
<td>-</td>
<td>0.0177</td>
<td>5.126e-06</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
</tr>
<tr>
<td>0.075</td>
<td>0.0177</td>
<td>-</td>
<td>0.0181</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
</tr>
<tr>
<td>0.031</td>
<td>5.126e-06</td>
<td>0.0181</td>
<td>-</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
</tr>
<tr>
<td>0.110 / 0.031</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
<td>-</td>
<td>1.0</td>
</tr>
<tr>
<td>0.031 / 0.110</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
<td>2.2e-16</td>
<td>1.0</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 5. FMS: t-test indicating the significance of the results (p-values).

<table>
<thead>
<tr>
<th>p-values</th>
<th>0.110</th>
<th>0.075</th>
<th>0.031</th>
<th>0.110 / 0.031</th>
<th>0.031 / 0.110</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.110</td>
<td>-</td>
<td>0.6818</td>
<td>0.7128</td>
<td>0.1089</td>
<td>0.4115</td>
</tr>
<tr>
<td>0.075</td>
<td>0.6818</td>
<td>-</td>
<td>0.9904</td>
<td>0.0397</td>
<td>0.6536</td>
</tr>
<tr>
<td>0.031</td>
<td>0.7128</td>
<td>0.9904</td>
<td>-</td>
<td>0.0665</td>
<td>0.7087</td>
</tr>
<tr>
<td>0.110 / 0.031</td>
<td>0.1089</td>
<td>0.0397</td>
<td>0.0665</td>
<td>-</td>
<td>0.0164</td>
</tr>
<tr>
<td>0.031 / 0.110</td>
<td>0.4115</td>
<td>0.6536</td>
<td>0.7087</td>
<td>0.0164</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6. P-PEAKS: t-test indicating the significance of the results (p-values).

<table>
<thead>
<tr>
<th>p-values</th>
<th>0.110</th>
<th>0.075</th>
<th>0.031</th>
<th>0.110 / 0.031</th>
<th>0.031 / 0.110</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.110</td>
<td>-</td>
<td>0.9409</td>
<td></td>
<td>0.0829 2.852e-08</td>
<td>0.0212</td>
</tr>
<tr>
<td>0.075</td>
<td>0.9409</td>
<td>-</td>
<td>0.1531</td>
<td>3.514e-07 0.0531</td>
<td></td>
</tr>
<tr>
<td>0.031</td>
<td>0.0829</td>
<td>0.1531</td>
<td>-</td>
<td>8.03e-12 0.4966</td>
<td></td>
</tr>
<tr>
<td>0.110 / 0.031</td>
<td>2.852e-08</td>
<td>3.514e-07</td>
<td>8.03e-12</td>
<td>-</td>
<td>1.061e-12</td>
</tr>
<tr>
<td>0.031 / 0.110</td>
<td>0.0212</td>
<td>0.0531</td>
<td>0.4966</td>
<td>1.061e-12</td>
<td>-</td>
</tr>
</tbody>
</table>

The results with the P-PEAKS problem confirm the mentioned trends because the domain is highly epistatic. A dynamic ratio is more efficient than a static ratio only when enlarging the pressure during the search. Also, from the three tested static ratio cGAs, the algorithm with a narrow grid (0.031) is the more efficient one, especially when compared with the square grid (0.110). See in Table 6 the calculated p-values.

5. Conclusions and Further Work

In this paper we have analyzed the performance of cellular EAs with a grid of strings folded to a torus. The used model is embedded in many other similar algorithms, and hence we hope these results to be broadly helpful. Our initial motivation has been the idea that the ratio influences the search in structured populations, assuming a constant neighborhood [3]. We wanted to get deeper in researching on how this occurs.

The main conclusion of this work is that a cEA with a thin grid -low ratio- is more efficient for multimodal and/or epistatic problems while it compares unfavorably with higher-ratio cEAs in non-epistatic and simple problems. A thin grid provides a twofold advantage: (1) it reduces the selection pressure (more exploration) and (2) it allows local exploitation in every neighborhood by using common variation operators or other problem-specific techniques the user might want to include (e.g. memetic EAs).

Whenever a small ratio shows efficient, changing the ratio from low to high during a run is also efficient, since we explicitly run the idea that an EA undergoes two optimization phases, namely exploration and then exploitation. Enlarging the ratio in the middle of the execution has yielded satisfactory results only in epistatic problems, and marginal benefits in multimodal problems with moderate epistasis.

As a future work it could be interesting to extend these results to new kinds of problems. Besides that, the natural next step is to make the algorithm to self-adjust the ratio depending on some genotypic or phenotypic measure such as diversity, average improvement along several generations, or other indicators of the progress of the cEA.
References


Acknowledgment

This work is partially supported by the CICYT project TIC1999-0754-C03-03.
Efficiency and Mutation Strength Adaptation of the \((\mu/\mu_I, \lambda)\)-ES in a Noisy Environment

Dirk V. Arnold and Hans-Georg Beyer
Department of Computing Science XI
University of Dortmund
44221 Dortmund, Germany
{arnold,beyer}@ls11.cs.uni-dortmund.de

Abstract. Noise is present in many optimization problems. Evolutionary algorithms are frequently reported to be robust with regard to the effects of noise. In many cases, there is a tradeoff between the accuracy with which the fitness of a candidate solution is determined and the number of candidate solutions that are evaluated in every time step. This paper addresses this tradeoff on the basis of recently established results from the analysis of the local performance of a recombinant multi-parent evolution strategy on a noisy sphere. It is shown that, provided that mutation strengths are appropriately adapted, the strategy is indeed able to cope with noise, and that results previously obtained for single-parent evolution strategies do not carry over to multi-parent strategies. Then, the problem of mutation strength adaptation in noisy environments is addressed. Mutative self-adaptation and the cumulative mutation strength adaptation algorithm are compared empirically in a simple fitness environment. The results suggest that both algorithms are prone to failure in the presence of noise.

1 Introduction

Noise is a common phenomenon in many real-world optimization problems. It can stem from a variety of sources, including measurement limitations, the use of randomized algorithms, incomplete sampling of large spaces, and human-computer interaction. Evolutionary algorithms (EAs) are frequently reported to be comparatively robust with regard to the effects of noise. In fact, noisy environments are considered a prime application domain for EAs.

For many problems of practical interest, it is possible to evaluate candidate solutions approximately using statistical sampling techniques. In such cases, there is a tradeoff between the accuracy with which a candidate solution is evaluated and the number of candidate solutions evaluated per time step. Assuming that fitness measurements are statistically independent, the variance of the measurements can be decreased by a factor of \(\kappa\) by averaging over \(\kappa\) measurements at any one parameter space location. Ignoring the overhead resulting
from averaging, the computational costs increase \( \kappa \)-fold\(^1\). Again ignoring computational overhead, it is therefore reasonable to ask whether greater benefits can be achieved by a \( \kappa \)-fold increase in the population size.

In the realm of genetic algorithms (GAs), Fitzpatrick and Grefenstette [4] have found an affirmative answer to this question. Their empirical analysis of the performance of a GA on a simple model function as well as empirical results they arrived at on a more complex image processing problem has shown that more efficient search results may be obtained with less accurate evaluations. It seems thus reasonable to say that for the problems under investigation, a GA is able to cope with noise.

For evolution strategies (ESs), previously published results seem to hint in the opposite direction. An analysis of the \((1, \lambda)\)-ES on a noisy sphere by Beyer [2] has shown the benefit of reducing the noise strength by means of averaging to be higher than that of increasing the number of offspring \( \lambda \). Follow-up research by Hammel and Bäck [6] not only found empirical support for Beyer's results, but they also reported that better results could be obtained by averaging over multiple samples rather than by increasing the population size for a recombinant multi-parent ES with self-adaptive mutation strength. On the basis of these results, it is hard to justify considering an ES a strategy which works well in noisy environments. After all, removing the noise seems to be more efficient than letting the ES deal with it.

However, results by Gruenz and Beyer [5] that show that self-adaptation in recombinant ES can be sensitive to an externally chosen learning parameter even in the absence of noise raise the possibility that the results obtained by Hammel and Bäck may have to be attributed to deficiencies in the self-adaptation mechanism rather than to a general superiority of averaging over multiple samples over increasing the population size.

The present paper first investigates the influence of the number of samples to be averaged over and of the population size on the local efficiency of a recombinant multi-parent ES for a simple fitness model. It relies on recent results from the analysis of the local performance of the \((\mu/\mu_1, \lambda)\)-ES in a noisy environment presented in [1]. It is shown that, in agreement with the results for GAs, for the environment investigated, locally an increased population size is preferable to a reduction of the noise strength by means of averaging provided that the mutation strength is optimally adapted. This is encouraging as it can serve as an explanation for the frequently observed robustness of ESs with respect to noise. Then, the problem of mutation strength adaptation in noisy environments is addressed. Mutative self-adaptation and the cumulative mutation strength adaptation algorithm are compared empirically. The results suggest that both algorithms may be prone to failure, and that more work is needed to obtain a better understanding of the influence of noise on self-adaptive processes.

\(^1\) Using computational schemes for averaging efficiently such as that suggested by Stagge [11] may make it possible to save some fitness function evaluations with an only minor loss in performance. The viability of such an approach may however depend on the amount noise present and is difficult to investigate analytically.
2 Algorithm, Fitness Environment, and Local Performance

Let \( f : \mathbb{R}^N \rightarrow \mathbb{R} \) be a function to be minimized. In what follows, \( f \) is referred to as the fitness function. The \((\mu/\mu_I, \lambda)\)-ES with isotropic normal mutations is an optimization strategy that, in every generation, produces \( \lambda \) offspring from a population of \( \mu \) parents and subsequently replaces the parental population by the \( \mu \) best of the offspring. Generation of an offspring individual consists in adding a mutation vector with independent, normally distributed components with mean 0 and variance \( \sigma^2 \) to the center of mass of the parental population. The standard deviation \( \sigma \) of the components of the mutation vector is commonly referred to as the mutation strength. Note that all parents participate in the generation of every single offspring individual, and that recombination is intermediate in that it involves computation of the center of mass as the arithmetic mean of the parental population.

The performance of the algorithm is frequently measured locally for single generations. A popular local performance measure is the expected fitness gain \( q = E[f((x)_{\mu}^{(g)}) - f((x)_{\mu}^{(g+1)})] \), the mean difference between the fitness of the center of mass \((x)_{\mu}^{(g)}\) of the parental population and the fitness of the center of mass \((x)_{\mu}^{(g+1)}\) of the selected offspring, which of course is the parental population of the following time step. Clearly, the expected fitness gain of the algorithm depends on parameters such as the size of the parental population \( \mu \), the number of offspring \( \lambda \), and the mutation strength \( \sigma \). In what follows, the strategy is considered to perform optimally if the parameters are chosen such that the expected fitness gain is maximized.

It is common practice in many theoretical studies of ES to investigate the performance of the strategies in environments that can locally be approximated by very simple fitness models. The goal of such analyses is to arrive at an understanding of the working principles that make certain strategies more successful than others in some environments. Such analyses can deliver information on how quantities measuring performance scale with parameters such as the dimension of the search space, the population size, or the mutation strength. Frequently, such information leads to prescriptions for adjusting parameters that turn out to be useful in more complex fitness environments as well.

The most commonly investigated simple fitness model is the quadratic sphere

\[
 f(x) = \sum_{i=1}^{N} (x_i - \hat{x}_i)^2, 
\]

where \( x = (x_1, \ldots, x_N)^T \) denotes a candidate solution and \( \hat{x} = (\hat{x}_1, \ldots, \hat{x}_N)^T \) is the location of the optimum in parameter space. In what follows, it is assumed that there is noise involved in the process of evaluating the fitness function. Evaluation at parameter space location \( x \) is assumed to yield a value which is normally distributed with mean \( f(x) \) and variance \( \sigma^2 \). Quite naturally, \( \sigma \) is referred to as the noise strength. It is furthermore assumed that consecutive evaluations of the fitness function yield statistically independent results.
Introducing normalizations

\[ \sigma^* = \frac{N}{r}, \quad \sigma^*_c = \sigma_c \frac{N}{2 r^2}, \quad \text{and} \quad q^* = \frac{N}{2 r^2}, \]

where \( r \) denotes the distance to the location of the optimum of the center of mass of the parental population, a recent paper \[1\] derives the fitness gain law

\[ \mathcal{E}[q^*] = \frac{c_{\mu/\mu, \lambda} \sigma^*}{\sqrt{1 + (\sigma^*_c/\sigma^*)^2}} - \frac{\sigma^*}{2 \mu} \] (1)

which is valid for the local performance of the \((\mu/\mu_I, \lambda)\)-ES on the quadratic sphere in the limit \( N \to \infty \) and under the condition that

\[ \sigma^* \ll \sqrt{N}. \] (2)

The \((\mu/\mu, \lambda)\)-progress coefficient is given by

\[ c_{\mu/\mu, \lambda} = \frac{\lambda - \mu}{2 \pi} \left( \frac{\lambda}{\mu} \right) \int_{-\infty}^{\infty} e^{-x^2} [\Phi(x)]^{\lambda-\mu-1} \left[ 1 - \Phi(x) \right]^{\mu-1} dx, \]

where \( \Phi \) denotes the cumulative distribution function of the standard normal distribution. It can according to \[3\] for large \( \lambda \) be approximated as

\[ c_{\mu/\mu, \lambda} \approx \frac{1}{\sqrt{2 \pi \alpha}} \exp \left( -\frac{1}{2} \left( \Phi^{-1} (1 - \alpha) \right)^2 \right), \] (3)

where \( \alpha = \mu/\lambda \) denotes the truncation ratio and \( \Phi^{-1} \) denotes the inverse function of \( \Phi \). Note that for large \( \lambda \), \( c_{\mu/\mu, \lambda} \) solely depends on the truncation ratio.

### 3 Efficiency of the \((\mu/\mu_I, \lambda)\)-ES

Defining the efficiency \( \eta_{\mu/\mu_I, \lambda} \) of the \((\mu/\mu_I, \lambda)\)-ES as the expected fitness gain per evaluation of the fitness function, it follows from (1) that

\[ \eta_{\mu/\mu_I, \lambda} = \frac{\mathcal{E}[q^*]}{\lambda} = \frac{c_{\mu/\mu, \lambda} (\sigma^*/\lambda)}{\sqrt{1 + ((\sigma^*_c/\lambda)/(\sigma^*/\lambda))^2}} - \frac{(\sigma^*/\lambda)^2}{2 \alpha}. \] (4)

As according to (3) for large \( \lambda \) the \((\mu/\mu, \lambda)\)-progress coefficient depends only on the truncation ratio \( \alpha \), (4) can for sufficiently large \( \lambda \) be used to compute the optimal truncation ratio \( \hat{\alpha} \) as a function of \( \sigma^*_c/\lambda \). Fig. 1 shows the result obtained by numerical root finding of the derivatives of \( \eta_{\mu/\mu_I, \lambda} \) with respect to \( \alpha \) and to \( \sigma^*/\lambda \). For \( \sigma^*_c = 0.0 \) of course the well known result \( \hat{\alpha} \approx 0.270 \) holds (compare \[3\]). However, it can also be seen that the optimal truncation ratio increases with increasing noise strength up to a value of 0.5 at the point...
where no positive progress can be made in the mean and therefore the optimal mutation strength becomes 0.0. That is, the ES can partially compensate for a lack of reliable information by using the noisy information provided by a larger number of parents than it would optimally use in the absence of noise.

The question of whether it is more beneficial to improve the accuracy of the fitness estimate for a candidate solution by averaging over \( K \) evaluations of the fitness function or to increase the population size \( \kappa \)-fold can now be answered. Fig. 2 shows the optimal efficiency that can be achieved by different strategies all of which use the same number of fitness function evaluations per time step and that therefore have the same computational costs per generation. It can be seen that the strategy which relies on a large population size rather than on a reduction of the noise strength by averaging consistently outperforms strategies with \( \kappa > 1 \). While for low noise strengths this agrees with the corresponding results for \((1, \lambda)\)-ESs, the results for high noise strengths markedly differ. According to [2], for \((1, \lambda)\)-ESs, at high noise strengths it is more beneficial to average over multiple samples rather than to increase the number of offspring \( \lambda \). For \((\mu/\mu_1, \lambda)\)-ESs with \( \mu > 1 \) however, evolutionary progress is achieved in the mean if \( \eta_{\mu/\mu_1, \lambda} > 0 \) and therefore, using (4), if

\[
\sigma^2 + \sigma^*_\epsilon^2 < (2\mu c_{\mu/\mu, \lambda})^2.
\]

Therefore, positive progress can only be achieved in the mean if

\[
\sigma^*_\epsilon < 2\mu c_{\mu/\mu, \lambda}
\]

holds. Thus, a \((\mu/\mu_1, \lambda)\)-ES which operates on an infinite-dimensional quadratic sphere at normalized noise strength \( \sigma^*_\epsilon \) that averages values obtained in \( \kappa \) fitness function evaluations is capable of operating with a positive efficiency if

\[
2\sqrt{Kc_{\mu/\mu, \lambda}} > \sigma^*_\epsilon.
\]

A \((\kappa_\mu/\kappa_\mu_1, \kappa\lambda)\)-ES operating in the same environment that uses fitness values without averaging is capable of operating with a positive efficiency if

\[
2\kappa c_{\kappa_\mu/\kappa_\mu, \kappa\lambda} > \sigma^*_\epsilon.
\]

As for sufficiently large \( \lambda \) the progress coefficients \( c_{\mu/\mu, \lambda} \) and \( c_{\kappa_\mu/\kappa_\mu, \kappa\lambda} \) approximately agree, and as for \( \kappa > 1 \) the relation
Fig. 2. Optimal efficiency $\hat{\eta}$ of different ($\mu/\mu_I, \lambda$)-ES that average over $\kappa$ independent fitness measurements for optimally tuned mutation strength and truncation ratio as functions of the transformed normalized noise strength $\sigma^*_v/\kappa \lambda$. The curves display, from top to bottom, the results for strategies with $\kappa$ times averaging and $\lambda/\kappa$ offspring individuals per generation for $\kappa = 1, 2, 4, \text{ and } 8$.

$\kappa > \sqrt{\kappa}$ holds, it is preferable to increase $\lambda$ rather than to average over several noisy fitness values.

There are at least two reasons however that practically make it unwise to choose $\lambda$ too large. The first reason is that the optimal normalized mutation strength $\hat{\sigma}^*$ increases with increasing $\lambda$, and that the limit (2) on the validity of (4) may be reached. The second reason is that the local view taken here is a limited one. The adaptation of the mutation strength requires information from previous time steps, making it preferable to make several steps with a smaller $\lambda$ rather than a single step with a large one. The problem of mutation strength adaptation is subject of the next section.

4 Self-Adaptation in Noisy Environments

In most fitness environments, continual adaptation of the mutation strength $\sigma$ is a necessary prerequisite for optimal or near optimal performance of an ES. As has been shown in the previous section, the observed superiority of averaging over multiple samples over increasing the population size in [6] does not hold locally if the mutation strength is optimally adapted. Thus, experimental results hinting in the opposite direction may have to be attributed to deficiencies in the mutation strength adaptation component of ES in noisy environments. In what follows, two mutation strength adaptation schemes — mutative self-adaptation (MSA), introduced by Rechenberg and Schwefel [9,10], and cumulative muta-
tion strength adaptation (CSA), due to Hansen and Ostermeier [8,7] — are experimentally evaluated for the noisy sphere. Note that although only isotropic mutations are discussed here, variants of both algorithms that adapt arbitrary normal mutation distributions do exist.

The \((\mu/\mu_i, \lambda)-ES\) with mutative self-adaptation of the mutation strength considered here assigns offspring individual \(i, i = 1, \ldots, \lambda\), a mutation strength

\[
\sigma_i = \sigma \exp (\tau N_i),
\]

where \(\sigma\) is either the arithmetic or the geometric mean of the mutation strengths of the parent individuals, \(\tau\) is set to \(1/\sqrt{N}\), and \(N_i\) is a standard normally distributed random variable. In what follows, the strategies using arithmetic and geometric recombination of the mutation strengths are denoted by MSA_+ and MSA_x, respectively. The individual mutation strengths are used in the process of mutating the object parameters of the respective offspring individuals. The underlying reasoning is that an individual with a well adapted mutation strength is likely to generate successful offspring, and that therefore its mutation strength enters the averaging of mutation strengths in the next generation.

The coupling between the mutation strength and the fitness of an individual is a rather loose one as the mutation strength merely determines the variance of the mutations while the fitness of an offspring individual depends on the particular realization of the mutation vector. The cumulative mutation strength adaptation algorithm has been suggested as a “derandomized” mutation strength adaptation scheme. It relies on the observation that if the current mutation strength is below its optimal value, consecutive mutation steps tend to be parallel, and that if the current mutation strength is too high, consecutive mutation steps tend to be antiparallel. Instead of using information from individuals with different mutation strengths at a single time step, it relies on a single mutation strength for all individuals during any time step and uses information from a sequence of time steps. The “evolution path” \(s\) is defined by \(s^{(0)} = 0\) and the recursive equation

\[
s^{(g+1)} = (1-c)s^{(g)} + \sqrt{c(2-c)} \sqrt{\mu}(z)^{(g+1)},
\]

where \((z)^{(g+1)}\) denotes the vector from the center of mass of the parental population at generation \(g\) to that at generation \(g+1\) divided by the mutation strength \(\sigma^{(g)}\) at generation \(g\). The mutation strength is updated according to

\[
\sigma^{(g+1)} = \sigma^{(g)} \exp \left( \frac{\|s^{(g+1)}\| - \chi_N}{D \chi_N} \right),
\]

where \(D\) denotes a damping constant and \(\chi_N\) is the mean distance per generation traveled in case of perpendicularity of consecutive mutation steps. For large \(N\), \(\chi_N\) equals approximately \(\sqrt{N}\). The constants \(c\) and \(D\) are set to \(1/\sqrt{N}\) and \(\sqrt{N}\), respectively, according to recommendations in [7], page 12.

Figure 3 shows results obtained in computer experiments with \((3/3_1, 10)-ESs\) employing either MSA or CSA. The fitness environment is a quadratic sphere with parameter space dimension \(N = 40\) and normalized noise strength \(\sigma^*_e \in \)
Fig. 3. Normalized mutation strength $\sigma^*$ (left) and fitness $f$ (right) over generation $g$ of $(3/3t, 10)$-ESs with MSA$_+$ (solid lines), MSA$x$ (dotted lines), or CSA (dashed lines) on a quadratic sphere with parameter space dimension $N = 40$. The graphs correspond to, from top to bottom, normalized noise strengths $\sigma^*_c = 0.0$, 2.0, 4.0, and 8.0. The results are averaged over 400 independent runs.
{0.0, 2.0, 4.0, 8.0} which is assumed to be constant throughout the parameter space. In all cases, the normalized mutation strengths are initialized to 10.0. Note that this setting is above the optimal values. The initial rapid decrease is however not due to a decrease of the mutation strength but rather to an increase in distance to the location of the optimum.

It can be observed that in the absence of noise all mutation strength adaptation mechanisms achieve approximately constant normalized mutation strength. The normalized mutation strength resulting from CSA is above the optimal value of about 3.0, those resulting from the two variants of MSA are below it. The normalized mutation strength resulting from MSAx is below that resulting from MSA+. All strategies achieve linear convergence order.

For \( \sigma^*_c = 2.0 \), CSA and MSA+ still achieve approximately constant normalized mutation strength in the vicinity of the optimal value. In fact, the resulting mean fitness gain per generation is even closer to its optimal value than in the absence of noise. MSAx on the other hand generates mutation strengths which decrease over time to values several orders of magnitude below the optimal value. The performance of the ES degrades correspondingly.

For \( \sigma^*_c = 4.0 \), CSA begins to exhibit the form of behavior that MSAx has generated for \( \sigma^*_c = 2.0 \) already, but at a slower pace. Alone MSA+ is capable of generating a constant normalized mutation strength in the vicinity of the optimal value which is still at about 3.0.

For \( \sigma^*_c = 8.0 \), things are different in that a reduction of the fitness is not possible on the average with only ten offspring individuals per generation. Thus, the optimal mutation strength is zero. CSA and MSAx generate decreasing normalized mutation strengths that lead to a comparatively slow deterioration of the fitness while MSA+ still generates constant normalized mutation strength of the same order of magnitude as without noise. Comparatively rapid divergence is a consequence.

5 Summary and Future Work

In this paper, the influence of noise on the performance of \((\mu/\mu_t\lambda)\)-ESs in a simple environment has been investigated. It has been shown that with increasing noise strength the loss of reliable information can partially be compensated for by letting a higher number of parents participate in the generation of offspring. Moreover, it has been shown that the strategy is able to cope with noise in that an increase in the population size is potentially more beneficial than the artificial reduction of the noise strength by means of averaging. This is in qualitative agreement with results from the realm of GAs and markedly differs from results obtained for \((1, \lambda)\)-ESs that are rather wasteful in that they discard a large amount of information present in unsuccessful offspring.

However, successful adaptation of the mutation strength is a prerequisite for good performance, and it has been demonstrated that both mutative self-adaptation and cumulative mutation strength adaptation can fail in the presence of noise in different ways. An analytical investigation of the performance of
mutation strength adaptation algorithms for \((\mu/\mu_I, \lambda)\)-ESs that can reveal the reasons for the potential failure and yield insight into the effects of different settings of the exogenous parameters such as \(\tau\) and \(c\) remains as a challenge for the future.

Acknowledgements

The first author is supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Be 1578/6-1. The second author is supported by the DFG as a Heisenberg fellow under grant Be 1578/4-1.

References

An Analysis of the Configuration Space of the Maximal Constraint Satisfaction Problem

Meriem Belaidouni¹ and Jin-Kao Hao²

¹ LGI2P, EMA-EERIE, Parc Scientifique Georges Besse, F-30000 Nîmes
² LERIA, Université d'Angers, 2 bd Lavoisier F-49045 Angers Cedex 01
{meriema@site-eerie.ema.fr, Jin-Kao.Hao@univ-angers.fr}

Abstract. This paper presents an analysis of the configuration space of a well-known combinatorial search problem called MAX-CSP. The analysis is based on a measure called "density of states (d.o.s)". We show experimentally that the configurations of a random MAX-CSP instance follow a normal distribution. This distribution allows us to get some insights about the behavior of a random walk search.

1 Introduction

The notion of landscape has been widely used to study the behavior of heuristic search algorithms such as genetic algorithms and local search. Landscape measures like autocorrelation indicate when a landscape is easy or difficult to search. Similarly, studies of local optima contribute to explaining why a landscape is difficult.

Recently, a new measure has been proposed aiming to study the configuration space of search problems [1,10]. This measure, called the density of states (d.o.s.), gives complementary and valuable information to predict or explain the dynamics of heuristic search methods. The d.o.s. simply counts the number of configurations per cost value in the configuration space. The counting may be carried out in different ways - analytically on some problems [7,8], by enumeration on small instances [4] or by approximation on large instances [1,3].

This paper undertakes the study of the configuration space of the Maximal Constraint Satisfaction Problem (MAX-CSP), a well-known combinatorial search problem widely studied by people working in artificial intelligence. The MAX-CSP is a general model allowing the formulation of numerous problems such as the MAX-SAT and the k-Coloring problem as well as many real-world applications such as scheduling and planning.

Using the d.o.s., we show that the density of states of random MAX-CSP instances follows a normal distribution characterized by a mean and a variance. This result allows us to obtain some intriguing insights into Random Walk (RW) dynamics. Indeed, previous experiments with RW have shown that the search invariably stagnates within zones having cost values which differ little, independent of the start point of the search. The normal distribution of the configuration space allows us to explain this RW phenomenon.
This article is organized as follows: After a brief recall of the Maximal Constraint Satisfaction Problem in Section 2, Section 3 presents some experiments with the Random Walk method. Section 4 deals with the principles of the measure of density of states. Section 5 establishes the relation between Random Walk and density of states with the help of intensive experiments. Section 6 concludes and suggests some perspectives.

2 Maximal Constraint Satisfaction Problem (MAX-CSP)

2.1 Definition

The MAX-CSP can be defined by means of the notion of constraint networks. A constraint network is a triplet $< V, D, C >$ where:

- $V = \{V_1,...,V_n\}$ is a finite set of variables;
- $D = \{D_1,...,D_n\}$ is a finite set of value domains associated with the variables;
- $C = \{C_1,...,C_n\}$ is a finite set of constraints, each constraint being a subset of the Cartesian product of the domains of several variables, specifying the forbidden value tuples.

Given a constraint network $< V, D, C >$, the problem of maximal constraint satisfaction consists in finding a value in $D_i$ for each variable $V_i$ such that the number of satisfied constraints is maximal [11]. In practice, instead of maximizing the number of satisfied constraints, one minimizes the number of unsatisfied constraints - which is rigorously equivalent. Subsequently, we adopt this minimization version of the problem. The objective function is noted $f$.

2.2 Instance generation

The test instances that are used in this work correspond to random, binary constraint networks (each constraint concerns only two variables) generated according to a standard model. A network class is defined by $< n, d, p_1, p_2 >$ which has $n$ variables, $d$ values per variable, $p_1.n.(n - 1)/2$ constraints taken randomly from $n.(n - 1)/2$ possible ones ($p_1$ is called the density), and $p_2.d^2$ forbidden pairs of values taken randomly from $d^2$ possible ones for each constraint ($p_2$ is called the tightness). For each given class $< n, d, p_1, p_2 >$, different instances can be generated using different random seeds $s$. A constraint network can be under-constrained or over-constrained [6]. These different regions are characterized by a factor called constrainedness: $\kappa = \frac{n-1}{2}p_1 \log_d(\frac{1}{1-p_2})$. $\kappa = 1$ separates under- ($\kappa < 1$) from over- ($\kappa > 1$) constrained networks. Networks with $\kappa \approx 1$ correspond to critically-constrained ones.

3 Experiments with the Random Walk

In this section, we carry out some experiments with Random Walk. We are especially interested in simple questions such as: Does RW follow any direction during its search? Where does a RW search go when it begins with very different starting points? Where does the search stop?
3.1 Random Walk

The Random Walk (RW) is a simple yet important search process for studying search spaces. RW is particularly relevant for an important class of optimization methods based on local search. RW is defined by an iterative process. It begins with an initial point $s$ in the configuration space $S$ and then moves repeatedly from the current $s$ to a randomly chosen neighboring one $s' \in S$ according to a neighborhood relation $N$. The experiments described below are based on this simple RW.

3.2 Experiments

The experiments aim to demonstrate the relevance of density of states to the dynamics of local search heuristics. Density of states carries new information which is not covered by measures based on correlation measure or local optima density. The experiments are carried out as follows. We take a random MAX-CSP instance $I=(S, f)$. We run a Random Walk search on the instance $I$ search space from three different kind of configurations:

- RW-I: from a random configuration,
- RW-II: from a random optimal or near-optimal configuration,
- RW-III: from a random very bad configuration.

![Fig. 1. Cost evolution for RW-I (RW from a random starting configuration)](image)

The instance used here has a known optimal cost of $f^* = 0$. Fig. 1 and Fig. 2 summarize the evolution of cost values obtained with RW-I, RW-II and RW-III. RW-I starts its evolution most probably around $f \approx 190$. All generated cost values oscillate around the cost area $[150,210]$. RW-II leaves the optimal cost area around $f = 0$ and goes toward the cost area $[150,210]$. RW-III leaves the bad cost values $f \approx 438$ and moves toward the cost area $[150,210]$.

These experiments are intriguing because they demonstrate RW follows a particular direction independently of its starting configuration. Indeed all these search processes stagnate around the same cost area. It seems that RW is strongly attracted by the configurations of this cost area. It is therefore interesting to consider the configurations according to their cost values.
3.3 Cost Density of Random Walk

Now we re-run the above RW search and count the number of times a cost is encountered. This leads to a distribution that can be called cost density (Figure 3). Figure 3 displays clearly a normal distribution. Consequently it appears that RW-II and RW-III leave their initial cost areas (low and high) to coincide with this distribution. The remark holds also true for RW-I. Two factors may be responsible for this distribution: 1) the neighborhood relation and 2) the number of configurations per cost (density of states). In what follows, we present a general approximation method for finding density of states and show tight relations between density of states and the dynamics of Random Walk.

4 Density of Cost States

For a given configuration space \((S, f)\) associated with a combinatorial problem, the density of states (d.o.s) gives the number of configurations per cost value.
Asselmeyer et al. [1, 10] have proposed a statistical method to approximate d.o.s that emits no hypothesis on the configuration space studied. The main purpose of the d.o.s. is to establish a classification of problem difficulty based on the mean and the variance of the density of states.

4.1 Method for Approximating Density of States

The approximation technique for d.o.s draws an analogy between energy particles $F$ in thermodynamics and configurations $s$ of cost $f(s)$ of the space $S$. The method bases itself on the following law:

$$P_{eq}(F) \sim N(F)e^{-\Delta F/T}$$  \hspace{1cm} (1)

- which links the number $N(F)$ of particles of energy $F$ to the frequency of their appearance $P_{eq}$ in a sampling process using the Metropolis method at temperature $T$ (Metropolis method was introduced in 1953 by N. Metropolis et al. [9]. This process moves from configuration $s$ to configuration $s'$ with probability $p= e^{-\Delta f/T}$ if $\Delta f = f(s') - f(s) \geq 0$ and probability $p=1$ otherwise. The approximation proceeds via the following stages:

1. Run the Metropolis sampling process at temperature $T$. On the sample of size $N$ and for each energy state $F$, count the number $N(F)$ of configurations having value $F$, approximate the density $P_{eq}(F)$ by the frequency distribution $N(F)/N$.

2. Adjust the density $P_{eq}(F)$ to the scale by applying a multiplier coefficient of $e^{F/T}$ as follows:

$$w(F) = P_{eq}(F)e^{F/T}$$  \hspace{1cm} (2)

3. Normalize to obtain an approximation of the state density

$$W(F) = N \frac{w(F)}{\sum_{F=0}^{c} w(F)}$$  \hspace{1cm} (3)

where $N$ is the total number of configurations, and $c$ the number of values that $F$ can take.

4. Execute the above simulation at different temperatures in order to scan the whole range of possible states.

5 Experiments

We carry out our experimental tests on random, binary MAX-CSP landscapes. The first goal is to confirm the behavior (see section 3) of Random Walk for different initializations. The second goal is to explain this behavior using d.o.s.
5.1 Random Walk Behavior and Cost Density

Table 1 presents 7 classes of instances used in our experiments. Each class includes ten different instances. The constrainedness is given by $\kappa$, $f^*$ is the best known or optimal cost. On each class, we run Random Walk with three different starting configurations: (I) a random configuration, (II) an optimal configuration, (III) a very bad configuration (see section 3.2). $f_{\text{begin}}$ (resp. $f_{\text{end}}$) indicates the initial (resp. final) cost value. The neighborhood relation involved is $N_I$; two configurations are neighbors if they differ by only one variable value.

Now if we consider the case of $<100.10.15%.25%>$, all initializations converge to the cost area $[130, 240]$. The same phenomenon occurs for the other classes of instances with different final cost intervals $f_{\text{end}}$. Fig. 4 shows the RW cost density of instance classes (a) to (g). It appears that the final cost intervals for each class of instances correspond to the most probable costs. This explains why RW is attracted by these cost values. It remains to explain why this density occurs for costs generated by Random Walk and why the random initial cost values are never very far from each other. The reason for these two phenomena lies, in large part, in the density of states (d.o.s.).

<table>
<thead>
<tr>
<th>Classes</th>
<th>$\kappa$</th>
<th>$f^*$</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) 100.10.15%.25%</td>
<td>0.93</td>
<td>0</td>
<td>$\approx 204$</td>
<td>$[130, 240]$</td>
<td>$\approx 2$</td>
</tr>
<tr>
<td>(b) 100.15.20%.30%</td>
<td>1.22</td>
<td>18</td>
<td>$\approx 293$</td>
<td>$[240, 360]$</td>
<td>$\approx 25$</td>
</tr>
<tr>
<td>(c) 100.10.20%.25%</td>
<td>1.24</td>
<td>19</td>
<td>$\approx 259$</td>
<td>$[180, 320]$</td>
<td>$\approx 21$</td>
</tr>
<tr>
<td>(d) 100.15.10%.45%</td>
<td>1.64</td>
<td>11</td>
<td>$\approx 234$</td>
<td>$[170, 270]$</td>
<td>$\approx 16$</td>
</tr>
<tr>
<td>(e) 200.20.25%.04%</td>
<td>0.34</td>
<td>0</td>
<td>$\approx 194$</td>
<td>$[140, 260]$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>(f) 200.20.05%.20%</td>
<td>0.37</td>
<td>0</td>
<td>$\approx 218$</td>
<td>$[150, 260]$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>(g) 200.20.18%.14%</td>
<td>0.9</td>
<td>0</td>
<td>$\approx 518$</td>
<td>$[420, 580]$</td>
<td>$\approx 2$</td>
</tr>
</tbody>
</table>

Table 1. Classes of different size for confirming the behavior of Random Walk.

Fig. 4. Cost density of Random Walk for classes (a) to (g).
5.2 Approximation of Density of States

In what follows, we propose to approximate the d.o.s of the class of instances \(< 100.10.15%.25% >\) using the method described in section 4.1. The density of states will help us to understand the cost density of Random Walk and the random initial cost values.

**Tuning of parameters** Applying the Metropolis algorithm to approximate a frequency distribution requires fixing the following parameters: the size of the sample, the neighborhood and the temperature. For a given instance, the best values of these parameters are usually determined empirically.

1. **size of sample**: This parameter is very important for the validity of an approximation. In the absence of analytical formulae giving a minimum value, trials are conducted on successively larger sizes with a view to stabilizing the approximation. It can happen though that this doesn’t bring a solution - notably when the minimum size is too large. For our experiments, we start typically with a size of 1,000 and end at 500,000. Beyond 500,000, the curves of frequency distribution are no longer sensitive to increase in sample size.

2. **neighborhood**: Two configurations are neighboring if they differ by the value of a single variable.

3. **temperature**: Compared with the sample size, temperature plays an even more critical role. It is this parameter that allows the search to reach zones of low cost\(^1\). In our experiments, we conduct trials at successively lower temperatures with a view to the greatest coverage of low cost zones. We typically use temperatures between \(T = 35\) and \(T = 0.5\). For values below 0.5 or above 35, the frequency distributions furnish no further information to the d.o.s. For \(T > 35\) the distributions differ little from that of \(T = 35\) while below \(T = 0.5\), multiplication by the scale factor obliterates the results.

**Results** We show now detailed results obtained on the class \(< 100.10.15%.25% >\) (This class has a known optimum \((f^* = 0)\) [5]). Fig. 5 presents the configuration distribution for this class: on the x-axis, the set of costs - going from 70 to 240 with a step of 20; on the y-axis, the estimated number of configurations for each cost. In Fig. 5, one notices that the configuration distribution approaches a normal law. Of the \(10^{100}\) configurations in the space, the majority, or \(\approx 3.5 \times 10^{98}\), seems to concentrate around the average of 185. The number of configurations diminishes when one departs from this average. The significant information lies in the area between 140 and 220. The extensions of the curve from 140 to 80 and from 220 to 240 correspond to regions where approximation has yielded figures which are almost null in relation to the other values.

At this point, one can explain why the random initial cost value for RW-I was \(f \approx 190\). In fact, the selected random cost responds to the cost distribution in the configuration space. More precisely, it belongs to the most probable cost values of the corresponding instance, here \([130, 240]\). This conclusion is confirmed by the

\(^1\) Recall that we are minimizing the number of violated constraints.
Fig. 5. Density of states of the instance < 100.10.15%.25% >: on the x-axis, costs of the objective function and on the y-axis, estimated number of configurations of instances (a) to (g). Indeed, all the initializations of Table 1 agree with the d.o.s. of these instances (204 ∈ [130,240]; 293 ∈ [240,360]; 259 ∈[180,320]; 234 ∈ [170,270]; 194 ∈ [140,260]; 218 ∈ [150,260]; 518 ∈ [420,580]).

5.3 Does d.o.s Explain the Cost Density of Random Walk?

After explaining the effect of d.o.s on the random initial cost, we try to establish its relation with the cost density of Random Walk.

Similarity - Shape Observing the d.o.s (Fig. 5) and the cost density of RW (Fig 3.), one can notice a common point: the normal shape. To show that this remark is not limited to the class of instances < 100.10.15%.25% >, we undertake one more experiment on the 7 classes of instances of Table 1. Notice that Fig. 4 shows already normal shapes for Random Walk cost density. To confirm the normality of the d.o.s. for these instances, we re-run the above described sampling procedure and then carry out a normality test using Matlab’s ‘normplot’. This test gives perfect linear curves for these instances, confirming thus the normality nature of the d.o.s. Let us mention that we carried out experiments on other MAX-CSP instances and used a "Random Selection" sampling technique which simply takes a set of configurations in $S$ in a random and independent manner. All these experiments have led to the same conclusion.

At this point, we wish to mention the study presented in [12]. It shows, using an analytical approach, the following result: taking the independence of constraint satisfaction, the random variable $\text{nuc} \sim B(nc,p_2)$; where $nc = p_1 \frac{n(n-1)}{2}$ is the number of constraints, $n$ the number of variables, $p_1$ the density and $p_2$ the hardness. We notice that this analytical result agrees perfectly with our experimental results since it is possible to approximate a binomial by a normal under certain conditions (conditions satisfied in this case).

Differences - Mean and Variance Now we turn to consider the differences between density of states and RW cost density using the average and the vari-
ance. In table 2 one can find the mean and the standard deviation of the binomial $B(nc,p_2)$ and the Random Selection—(data based on 10000 random configurations). Both are considered to be approximations of the exact mean and standard deviation of the classes (a) to (g).

We compare these approximated mean and deviation with the RW’s mean and standard deviation (data based on 1000000 configurations generated by Random Walk). The first neighborhood relation involved is $N_1$ (two configurations are neighbors if they differ by one variable value), the second is $N_2$ (two configurations are neighbors if they differ at a single conflicting variable). The results show that the mean and the standard deviation of Random Walk depend on the mean and standard deviation of the density of states (represented by the Binomial and Random Selection mean and standard deviation). They are also sensitive to the neighborhood relation. Thus, the samplings involving $N_1$ and $N_2$ are biased and present different results comparing with the Random Selection and the Binomial law. This phenomenon has an explanation: the use of a neighborhood relation favors the neighboring configurations and so introduces a correlation between the generated configurations while a pure random sampling (here Random Selection) using no neighborhood is a process without memory. One can also notice that the bias introduced by $N_2$ is larger than the one introduced by $N_1$. Indeed we obtained smaller means values with $N_2$. Moreover, the bias of $N_2$ is not regular: it is much more significant in instances (e) (whose mean is about 144 instead of 198 for $N_1$) and (f) (where the mean is around 156 instead of 198 for $N_1$) than in others.

We conclude from this experiment that the mean and the standard deviation of the cost density of RW result from the mean and the standard deviation of the density of states plus a bias introduced by the neighborhood relation. This bias depends on the instance. Before presenting our future work, it is worth pointing out that the mean and standard deviation of Random Walk can be considered as measures for neighborhood efficiency. These measures have the particularity (or quality) of being instance dependent.

### Table 2. Approximation of mean and standard deviation for different neighborhood relations.

<table>
<thead>
<tr>
<th>Classes</th>
<th>$B(nc,p_2)$</th>
<th>Random Selection</th>
<th>Random Walk ($N_1$)</th>
<th>Random Walk ($N_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>p% Conf.</td>
<td>p% Conf.</td>
<td>p% Conf.</td>
<td>p% Conf.</td>
</tr>
<tr>
<td>10</td>
<td>10% 15% 25%</td>
<td>185.62 185.65</td>
<td>184.47 184.58</td>
<td>117.70 117.80</td>
</tr>
<tr>
<td>100</td>
<td>15% 20% 30%</td>
<td>214.70 214.94</td>
<td>143.50 143.58</td>
<td>120.80 121.00</td>
</tr>
<tr>
<td>1000</td>
<td>20% 25% 30%</td>
<td>247.50 247.60</td>
<td>136.70 136.80</td>
<td>137.70 137.80</td>
</tr>
<tr>
<td>10000</td>
<td>20% 25% 30%</td>
<td>277.80 277.90</td>
<td>149.70 149.80</td>
<td>145.80 146.00</td>
</tr>
<tr>
<td>100000</td>
<td>20% 25% 30%</td>
<td>290.00 290.10</td>
<td>154.20 154.30</td>
<td>150.30 150.40</td>
</tr>
<tr>
<td>(e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

show that the mean and the standard deviation of Random Walk depend on the mean and standard deviation of the density of states (represented by the Binomial and Random Selection mean and standard deviation). They are also sensitive to the neighborhood relation. Thus, the samplings involving $N_1$ and $N_2$ are biased and present different results comparing with the Random Selection and the Binomial law. This phenomenon has an explanation: the use of a neighborhood relation favors the neighboring configurations and so introduces a correlation between the generated configurations while a pure random sampling (here Random Selection) using no neighborhood is a process without memory. One can also notice that the bias introduced by $N_2$ is larger than the one introduced by $N_1$. Indeed we obtained smaller means values with $N_2$. Moreover, the bias of $N_2$ is not regular: it is much more significant in instances (e) (whose mean is about 144 instead of 198 for $N_1$) and (f) (where the mean is around 156 instead of 198 for $N_1$) than in others.

We conclude from this experiment that the mean and the standard deviation of the cost density of RW result from the mean and the standard deviation of the density of states plus a bias introduced by the neighborhood relation. This bias depends on the instance. Before presenting our future work, it is worth pointing out that the mean and standard deviation of Random Walk can be considered as measures for neighborhood efficiency. These measures have the particularity (or quality) of being instance dependent.

### 6 Conclusions and Perspectives

In this paper, we have analyzed the MAX-CSP configuration space. We have shown that the density of states (d.o.s) of random instances approaches a normal law. This distribution sheds light on some interesting questions related to the dynamics of random walks: (1) we understand now why RW is attracted by some
cost areas. In fact, these cost areas correspond to space areas that contain large concentrations of configurations. However the mean cost of random walks and d.o.s. are not (generally) equal. This is due to the bias that the neighborhood relation introduces. (2) We learn that a random initial configuration will have a cost around the mean of the d.o.s. Density of states appears thus to be the first ingredient for understanding the behavior of search methods, the second and third ingredients being the neighborhood relation and the stochastic local search strategy chosen.

In our future work we want to approximate the cost densities of advanced local search methods such as Tabu Search and Simulated Annealing and analyze their relation with d.o.s. Also, we start to examine other problems like Graph Coloring and SAT. Finally, let's note that configuration analysis, like cost analysis, can contribute to understanding stochastic local search [2].

Acknowledgements The authors highly appreciate the valuable and detailed comments of the anonymous reviewers.

References

On the Desired Behaviors of Self-Adaptive Evolutionary Algorithms

Hans-Georg Beyer¹ and Kalyanmoy Deb²

¹ Department of Computer Science XI
University of Dortmund, D-44221, Germany
beyer@zappa.cs.uni-dortmund.de

² Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
deb@iitk.ac.in

Abstract. In this paper, we postulate some desired behaviors of any evolutionary algorithm (EA) to demonstrate self-adaptive properties. Thereafter, by calculating population mean and variance growth equations, we find bounds on parameter values in a number of EA operators which will qualify them to demonstrate the self-adaptive behavior. Further, we show that if the population growth rates of different EAs are similar, similar performance is expected. This allows us to connect different self-adaptive EAs on an identical platform. This may lead us to find a more unified understanding of the working of different EAs.

1 Introduction

With the need of optimizing dynamically changing objective functions, finding the global optimum, and finding the optimum with arbitrary precision, there had been an increasing amount of interest with the so-called self-adaptive evolutionary algorithms (SA-EAs). Although most studies exist using evolution strategies (ESs), recently self-adaptive behavior of real-parameter genetic algorithms (GAs) and evolutionary programming (EP) have also been demonstrated. In this paper, we postulate two main behaviors that any self-adaptive ES should possess: (i) the population mean should remain unchanged and (ii) the population variance must increase by the application of a recombination or a mutation operator.

In the remainder of the paper, we present results on the population mean and variance of a number of EAs including the isotropic self-adaptive ES and a number of real-parameter GAs. By equating the growth of population variance of different EAs, we demonstrate that they all exhibit self-adaptive behavior and show similar performance on a sphere model. The calculations and simulations presented in the paper mark a first step towards a deeper understanding of the connections between different EAs, a matter which may lead us to a unified theory of evolutionary algorithms.

2 Self-Adaptive Evolutionary Algorithms

There are primarily three different ways self-adaptation is used in an ES: (i) a hierarchically organized population-based meta-ES [1], (ii) adaptation of covariance matrix
determining the probability distribution for mutation [2], and (iii) explicit use of self-adaptive control parameters [3, 4]. The meta-ES method of self-adaptation uses two levels of ESs – the top level optimizes the strategy parameters (such as mutation strengths), a solution of which is used to optimize the true objective function in the lower level ES. The second method (CMA) records the population history for some number of iterations to calculate covariance and variance information among object variables. In this paper, we use the third type of self-adaptive ES, where the strategy parameters are explicitly coded and updated using the log-normal update rule in each generation. In its simplest form, a single mutation strength $\sigma$ is coded along with $N$ object variables with the following update rules:

$$\sigma^{(t+1)} = \sigma^{(t)} \exp(\tau_0 N(0, 1)), \quad (1)$$

$$x_i^{(t+1)} = x_i^{(t)} + \sigma^{(t+1)} N_i(0, 1), \quad (2)$$

where $N(0, 1)$ and $N_i(0, 1)$ are realizations of a one-dimensional normally distributed random variable with mean zero and standard deviation one. The parameter $\tau_0$ is the learning parameter which must be set as $\tau_0 \propto N^{-1/2}$, where $N$ is the dimension of the variable vector [4]. There exist non-isotropic and correlated self-adaptive ESs [5] which use basically the same ideas. They are not considered in this paper.

Recently, a number of real-parameter GAs with a recombination operator (and no explicit mutation operator) have been found to demonstrate the self-adaptive behavior. Most of these recombination operators use two parent solutions $(x_{1,k}$ and $x_{2,k})$ variable-by-variable and produces two children solutions $(y_{1,k}$ and $y_{2,k})$ according to some probability distributions [6]:

$$y_{1,k} := \frac{1}{2} \left[ (1 - \beta_k) x_{1,k} + (1 + \beta_k) x_{2,k} \right],$$

$$y_{2,k} := \frac{1}{2} \left[ (1 + \beta_k) x_{1,k} + (1 - \beta_k) x_{2,k} \right]. \quad (3)$$

The parameter $\beta_k$ is a sample from a random number generator with a density function $p(\beta)$ that depends on the recombination operator used. We present this probability density function for three recombination operators in the following.

For the simulated binary crossover (SBX-\(\eta\)) operator [7], a bi-modal probability distribution with its mode at the parent locations is used:

$$p(\beta) := \begin{cases} \frac{1}{2} (\eta + 1) \beta^n, & \text{if } 0 \leq \beta \leq 1 \\ \frac{1}{2} (\eta + 1) \frac{1}{\beta^{\eta + 2}}, & \text{if } \beta > 1. \end{cases} \quad (4)$$

The parameter $\eta$ determines the extent of search. A small value means wider search and a large value means restricted search. In most instances, $\eta = 2$ is used.

For the BLX-\(\alpha\) operator, two children solutions are chosen uniformly from a region proportional to the spread of the parent solutions using the following distribution:

$$p(\beta) := \frac{1}{1 + 2\alpha} \quad \text{for } 0 \leq \beta \leq (1 + 2\alpha). \quad (5)$$

The fuzzy recombination operator [8] is similar to SBX-1, but uses a triangular distribution with mode at $\beta = 1$:

$$p(\beta) := \begin{cases} 1 - \frac{1 - \beta}{2d}, & \text{if } (1 - 2d) \leq \beta < 1 \\ 1 - \frac{\beta - 1}{2d}, & \text{if } 1 \leq \beta \leq (1 + 2d). \end{cases} \quad (6)$$
In all these recombination operators, one aspect is common: The difference between the parent and a likely child solution is proportional to the difference between the parent solutions. This aspect is responsible for these GAs to demonstrate self-adaptive behavior. Since the spread of children solutions is always in proportion to the spread of the parents, overall convergence or divergence of the population depends on the balance of the explorative tendencies of the variation operators and the selection operator’s effect to exploit the underlying fitness function. No explicit strategy parameter is needed to determine the spread of children population. This is automatically provided by the parent population. In [6], an intuitive connection between the isotropic self-adaptive ES and real-parameter GAs with SBX has been drawn. Here, we show a more deeper connection with population statistics.

3 Two Postulates for Desired Self-Adaptation

In the following, we call a recombination or a mutation or a combination of both as the variation operator, since it perturbs the parent population obtained after the selection operator. Although it is not absolutely necessary, most EA variation operators do not use any fitness information, instead partial information between parents are shared and rearranged, respectively. If both operators use fitness information, there may be too much emphasis on the exploitation aspect and resulting GAs may prematurely converge. Put it another way, selection operators use the information from the fitness space, whereas the variation operators explores the search region indicated by the parent population. Since an ideal variation operator may only process search space information of the selected population, the variation operator should not introduce any bias on that population level. Therefore, we postulate the following:

Postulate 1 Under a variation operator, the expected population mean should remain unchanged.

Since the variation operator, in general, does not use fitness information, there is no reason for it to shift the parent population mean in any direction in the search space. This is true with crossover operators used in binary-coded GAs where binary string coding is used for real-parameter variables. Standard mutation operators in ES/EP do not use fitness information, instead use a zero-mean probability distribution to create a mutated child. Thus, the expected population mean is not changed under a standard mutation operator.

Under the selection operator, a population of \( \mu \) solutions is expected to lose some not-so-good solutions, only to make duplicates of some good solutions. This process of deleting some solutions and making duplicates of some other solutions will, in general, reduce the variance of the population. That is, the variance of population in the search space after the selection operation is, in most situations, smaller than that before the reproduction operation. Since the selection operator tends to reduce variation (in general), it is important that the variation operators possess certain properties in order to adequately counteract this trend. Thus, we have the second postulate:

Postulate 2 The expected population variance should increase with generation number.
In the following, we explain why we think this may be desired on a number of fitness landscapes. For a fairly flat fitness scenario at the current population (Figure 1(a)), it is desired to go out of the current region as quickly as possible in the hope for better regions. Thus, it seems desirable to increase the population variance exponentially faster.

In the case of linear fitness landscape (Figure 1(b)), again it makes sense to increase the population variance exponentially faster to reach near the optima. For unimodal landscapes, if the entire population resides on one side of the optimum, it is equivalent to the linear fitness landscape and the population variance must be increased. On the other hand, if the population brackets the optimum, it is desirable to reduce the population variance. We argue here that even in this case population variance under the variance operator may increase, provided the reduction of population variance by the selection operator has a larger effect, so that overall population variance reduces with generation. However, in the case of multi-modal fitness landscapes (Figure 1(c)), if the population is trapped in a local optimum a variance operator with an increasing population variance property is useful, particularly because it has a non-zero probability of getting out of the current optimal basin. Thus we observe that, in general, it is desired to have a variance operator which increases the population variance.

4 Analysis of Population Mean and Variance in Different EAs

Since zero-mean Gaussian distribution is used for self-adaptive ESs, the population mean is preserved between parent and children population. For real-coded GAs, the population mean of the object parameters is given as \( \langle y \rangle := \frac{1}{\mu} \sum_{k=1}^{\mu/2} (y_{1,k} + y_{2,k}) \). By using (3), we obtain

\[
\langle y \rangle := \frac{1}{\mu} \sum_{k=1}^{\mu/2} (x_{1,k} + x_{2,k}) = \frac{1}{\mu} \sum_{m=1}^{\mu} x_{m} = \langle x \rangle. \tag{7}
\]

Here it was taken into account that the \( x_{1,k}, x_{2,k} \) are independent samples from the parental population and the \( x_{m} \) is just a renumbering of the individuals. Thus, GAs with a recombination operator described in (3) preserves the population mean of the parental population as required by Postulate 1.

4.1 Population variance in self-adaptive ESs

In order to generate a new offspring individual with index \( l \) for generation \( g + 1 \), first the strategy parameter set is recombined and after that the result is mutated by multi-
plication with a random number $\Xi$. The obtained new mutation strength $\sigma_l^{(g+1)}$ serves as the standard deviation for the mutation of offspring $l$ and is incorporated in the offspring genome. The object parameter vector $y_l$ is obtained by first recombining the $\mu$ parental object parameter vectors and then applying a Gaussian mutation with strength $\sigma_l^{(g+1)}$. Therefore, we have the following ES update rule expressing the ES algorithm in a condensed form

$$\forall l = 1, \ldots, \lambda : \begin{cases} \sigma_l^{(g+1)} := \text{Reco}_\sigma(y_{1:1;\lambda}, \ldots, y_{\mu:1;\lambda}) \cdot \Xi_l \\ y_l^{(g+1)} := \text{Reco}_y(y_{1:1;\lambda}, \ldots, y_{\mu:1;\lambda}) + \sigma_l^{(g+1)} \mathcal{N}_l(0, 1). \end{cases}$$

(8)

For flat fitness functions (i.e., constant fitness), the population variance of a single object parameter entry $y_i := (y_i)$ after selection is given by (writing $y = y_i$)

$$\sigma^2[y]^{(g+1)} = \mathbb{E} \left[ \text{Var} \left\{ \text{Reco}_y(y_{1;\lambda}, \ldots, y_{\mu:1;\lambda}) \right\} \right] + \mathbb{E} \left[ \sigma^2[y]^{(g)} \right].$$

(9)

The first part is the effect of recombination of object parameters and the second part is that of the strategy parameter. We calculate these two parts separately.

There are two types of recombination operators commonly used for object parameters – intermediate (“I”) and dominant (“D”), although usually the latter is recommended [5]. For each object parameter, in the former case, an average of all parameter values is chosen as a child and the latter case, one parameter value from $\mu$ parents is chosen with uniform probability. We have the following results for a $(\mu/\mu, \lambda)$-ES [9]:

$$\mathbb{E} \left[ \text{Var} \left\{ \text{Reco}_y(y_{1;1;\lambda}, \ldots, y_{\mu:1;\lambda}) \right\} \right] = \begin{cases} 0, & \text{if } (\mu/\mu_I, \lambda)-\text{ES} \\ (1 - \frac{1}{\mu}) \sigma^2[y]^{(g)}, & \text{if } (\mu/\mu_D, \lambda)-\text{ES}. \end{cases}$$

(10)

It is clear that the expected population variance can only reduce or remains unchanged due the recombination of object parameters.

Usually, an intermediate recombination strategy is used for the strategy parameter, alternatively geometric recombination has been suggested [10]. Under the geometric recombination, a geometric mean of all strategy parameters is chosen. As to the mutation operator, the log-normal update rule is used, i.e., $\Xi := e^{\tau \mathcal{N}(0,1)}$.

In order to investigate the effect of the variation operators in isolated manner, the behavior of the ES in a flat fitness landscape will be considered. It is important to realize that due to the selection neutrality in flat fitness landscapes, the first line in (8) is decoupled from the second line. That is, the $\sigma$ evolution is fully determined by the first line in (8). Therefore, the expected population variance (9) can be determined by calculating the evolution of the second moment of the first line in (8). Due to length restrictions, the simple but lengthy calculations must be omitted here. The overall population variance growth equations under dominant recombination (“D”) of object parameters and both intermediate (“I”) and geometric (“G”) recombination of strategy parameter are found to be as follows [11]:

$$(\mu/\mu_D, \lambda)-\text{ES}:$$

$$\sigma^2[y]^{(g)} = \sigma^2[y]^{(0)} \left(1 - \frac{1}{\mu}\right)^g + \frac{\sigma^2[y]^{(0)} e^{2\tau \varphi}}{(1 + \frac{1}{\mu} e^{\tau \varphi - 1})} \left( \frac{e^{2\tau \varphi} \left[1 + \frac{1}{\mu} (e^{\tau \varphi - 1})\right]}{(e^{\tau \varphi} \left[1 + \frac{1}{\mu} (e^{\tau \varphi - 1})\right]) - (1 - \frac{1}{\mu})} \right).$$

(11)
\( (\mu/\mu_{\text{DG}}, \lambda) \)-ES:

\[
\sigma^2[y]^{(0)} = \sigma^2[y]^{(0)} \left( 1 - \frac{1}{\mu} \right)^9 + \frac{\sigma_r^{2(0)} e^{2\tau^2 \mu/\mu} - \left( 1 - \frac{1}{\mu} \right)^9}{e^{2\tau^2 / \mu} - \left( 1 - \frac{1}{\mu} \right)}.
\] (12)

Here, \( \sigma_r^{2(0)} \) is the initial value of the square of the mutation strength. As one can easily verify, both Eq. (11) and (12) are in accordance with Postulate 2.

### 4.2 Population variance of real-parameter GAs

Using the following definition of population variance

\[
\sigma^2[y] := \mathbb{E}[\text{Var} \{y\}] := \mathbb{E} \left[ \frac{1}{\mu} \sum_{m=1}^{\mu} (y_m - \langle y \rangle)^2 \right],
\] (13)

and after some lengthy calculations [11], we obtain the following relationship:

\[
\sigma^2[y] = \frac{1}{2} \left( 1 - \frac{2}{\mu} + \overline{\beta^2} \right) \sigma^2[x],
\] (14)

where \( \overline{\beta^2} = \int \beta^2 p(\beta) d\beta \). Thus, in order to agree with Postulate 2, the following conditions must hold

\[
\overline{\beta^2} > 1 \quad \text{and} \quad \mu > \frac{2}{\overline{\beta^2} - 1}.
\] (15)

The calculation of \( \overline{\beta^2} \) for the three recombination operators using (4), (5), and (6) is a straightforward task. For example, fuzzy recombination (6) yields \( \overline{\beta^2} = 1 + 2d^2/3 \). The condition (15) for an increase in population variance becomes

SBX-\( \eta \): \( \mu > \frac{1}{2}(\eta + 3)(\eta - 1) \),

BLX-\( \alpha \): \( \mu > \frac{3}{2\alpha + 2\alpha^2 - 1} \),

FR-\( d \): \( \mu > \frac{3}{d^2} \).

Equation (14) allows us to compare different recombination operators. Equating the corresponding variance growth terms under the flat fitness case, given one of the control parameters, say \( \eta \), we obtain the corresponding parameter values for the other recombination operators (see Table 1).

### 4.3 Self-adaptive ESs and real-parameter GAs

In order to compare ES and GA performance on a reasonable basis, it is conjectured that both EAs should exhibit similar variance evolution. As a first approximation, one can equate the population variance growth among self-adaptive ESs and real-parameter GAs
Table 1. Parameter values for identical population variance growth in three crossover operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>n</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBX</td>
<td>n</td>
<td>0.662</td>
<td>0.500</td>
<td>0.419</td>
<td>0.381</td>
</tr>
<tr>
<td>BLX α</td>
<td>0.662</td>
<td>0.500</td>
<td>0.419</td>
<td>0.381</td>
<td></td>
</tr>
<tr>
<td>FR d</td>
<td>1.095</td>
<td>0.707</td>
<td>0.433</td>
<td>0.226</td>
<td></td>
</tr>
</tbody>
</table>

in flat fitness landscapes. For example, we obtain the following relationship for SA-ES with log-normal σ-mutations and \( (μ/μ_N) \)-recombination, and a real-coded GA with fuzzy recombination (FR) by equating the population variance growth rates (in order to get the respective ES formula derived in [11], substitute \( (1 - \frac{1}{μ}) \) ≡ 0 in Eq. (11))

\[
\frac{σ^2[y]^{(g)}}{σ^2[y]^{(g-1)}} = e^{\tau^2} + \frac{1}{μ_{ES}} e^{\tau^2} \left( e^{\tau^2} - 1 \right) = 1 + \frac{d^2}{3} - \frac{1}{μ_{GA}}. \tag{19}
\]

For large population sizes, the following relationship is obtained

\[
τ ≈ \sqrt{\ln \left( 1 + \frac{d^2}{3} \right)}. \tag{20}
\]

These relations will be used in Section 5.2.

5 Simulation Results

First, we verify the population growth calculations mentioned in the previous section by simulating different EAs on the flat fitness function. Later, we investigate their effects on a sphere model.

5.1 The flat fitness model

Figure 2 shows the theoretical predictions and obtained results with (6/6,60)-SA-ES. Two recombinations for the strategy parameter – intermediate and geometric – are used. Simulation results have been obtained by averaging over 500 independent runs. In all experiments \( σ^2[y]^{(0)} = 0 \) and \( σ_r^{(0)} = 0.001 \) have been chosen. A learning parameter \( τ = 0.3 \) has been used. The upper curve is for intermediate recombination. There are two observations. First, the results closely match the theoretical predictions ((11) and (12)). Second, the rate of growth under geometric recombination is much smaller than under intermediate recombination. Considering (11) and (12) one can estimate that the geometric recombination has at most \( μ \) times slower rate than intermediate recombination.

Next, we verify the theoretical predictions of real-parameter GAs. In all simulations here, the population has been randomly initialized with variance 1. First, we use GAs with the SBX operator with a population size \( μ = 10 \) on a flat fitness function. Two values of \( n = 2 \) and 5 are used. Equation 16 suggests that population variance should
increase in the first case and decrease in the second case with generation. We observe the same behavior in Figure 3. The deviation of the simulation from the theoretical predictions are due to cumulation of statistical errors.

Next, we apply GAs with BLX-0.5 operator on the flat fitness function. Figure 4 shows two cases: $\mu = 60$ and $\mu = 4$. As predicted by Eq. (17), the population variance increases and decreases, respectively.

Finally, the GA with FR-0.5 operator is applied. Two population sizes $\mu = 60$ and $\mu = 8$ show expected behavior in Figure 5, as predicted by Eq. (18).

### 5.2 The sphere model

It is important to note that the parameter values tabulated in Table I are for an equivalent expected population variance growth under different EAs and are valid only for flat
fitness function. In this subsection, we use the same parameter values but apply the EAs to a sphere model.

We use a 20-dimensional sphere model with optimum at $x_i = 0$. For GAs, we use binary tournament selection. A population size of 100 is chosen. We have initialized the population at $x_i \in [-5, 5]$. For the three GAs, we choose the parameter setting, presented in the first column of Table 1.

For the self-adaptive runs, we have used a (50/50, 100)-ES, in order to maintain a selection pressure similar to that in the binary tournament selection. For $d = 1.095$ and $\mu_{GA} = 100$, the learning parameter for SA-ES is $\tau = 0.57$ (Equation 19). Figure 6 shows the distance of the best solution in the population from the optimum for all three crossover operators and self-adaptive ES (averaged over 10 independent runs). It is clear from the plot that all EAs exhibit similar linear convergence behavior (residual distance to optimum at generation $g$).

6 Conclusions

In this paper, we have postulated properties for population mean and variance for self-adaptive evolutionary algorithms (SA-EAs) in real-valued (unrestricted) search spaces. With the task of exploration and exploitation in mind, we argued that the population mean should not be changed by the variance operator. We have also argued that it may be better strategy, in general, to have a tendency toward increasing the population variance by a variation operator (that is, by recombination and mutation).

We have calculated the population variance growth of self-adaptive ESs and of three recombination operators – blend crossover (BLX), simulated binary crossover (SBX), and fuzzy recombination (FR) – commonly-used in real-coded GAs. Theoretical predictions of increasing and decreasing nature of population variance are validated with
experimental results on a flat fitness landscape. We have shown that GAs with the above three crossover operators and self-adaptive ESs exhibit similar self-adaptive features on the sphere model, as long as all exhibit similar population variance growth in flat fitness landscapes. This reveals a deep connection between the working of self-adaptive ESs with real-parameter GAs and takes us a step ahead towards a unified theory for the working principles of different EAs.

Acknowledgements

The first author acknowledges support as Heisenberg Fellow of the Deutsche Forschungsgemeinschaft under grant Be1578/4-1. The second author acknowledges the support from Alexander von Humboldt Foundation, Germany and Department of Science and Technology, Government of India, during the course of this study.

References

Abstract. Three theoretical perspectives upon conservation of performance in function optimization are outlined. In terms of statistical information, performance is conserved when the distribution of functions is such that all domain subsets of a given size have identically distributed random values. In terms of algorithmic information, performance is conserved because almost all functions are algorithmically random. Also, an optimizer's algorithmic complexity bounds the information gained or lost in its processing of the test function. The practical consequences of these theoretical results are explored, with emphasis upon the results from algorithmic information theory, which are new.

1 Introduction

The concept of conservation of performance began with Schaffer [1] in the domain of machine learning, and was subsequently extended to function optimization by Wolpert and Macready [2]. The general idea of conservation, as originally constituted, is that if all problems in a domain are equally likely, then all algorithms for solving problems in that domain have identically-distributed performance scores. English [3] subsequently derived conservation of optimizer performance in terms of conservation of statistical information, and showed that conservation obtains if and only if the distribution of test functions is such that all domain subsets of a given size have identically distributed random values.

Recently English [4] gave a unified treatment of conservation of performance in learning and generalization, shifting from statistical to algorithmic information. The result is a body of conservation results that make no reference to the distribution of test cases. Indeed, the new results apply to particular functions.

The purpose of the present work is to summarize the new theory of [4], and to give a preliminary exploration of the practical consequences in evolutionary optimization. The following sections give conventions and notation, as well as a brief introduction to algorithmic information theory. Then conservation results are stated, and practical consequences are developed. A conclusion follows.
2 Conventions and Notation

All algorithms operate upon finite objects. Let $T$ be the set obtained by sampling the real interval $[-b, b)$ uniformly at $2^L$ points, including $-b$ and $0 < b$. The set of all test functions is $F = \{ f : S \to T \}$, with $|S| = M \geq 1$, and $|F| = (2^L)^M = 2^{LM}$. Thus the functions in $F$ can be placed in one-to-one correspondence with the bit strings of length $N = LM$. Functions in $F$ also can be represented as strings in $T^M$.

Here optimizers are modeled as deterministic, though perhaps pseudorandom with a constant seed, and may be sequential or parallel. They must be nonredundant in the sense of evaluating no point twice, exhaustive in the sense of evaluating every point, and algorithmic in the sense of terminating when every point has been evaluated.

For the sake of analysis, test functions are presented to optimizers as input strings. Evaluating a point is equivalent to reading the sequence of bits giving its value. An optimizer writes a permutation of the output string which indicates the order in which values were first read from the input string. The permutation must be honest in the sense of not using the values to determine the order in which they are written. In the case of parallel optimizers, inputs that are read simultaneously are written consecutively in the output string, and are ordered arbitrarily with respect to one another (e.g., according to input ordering). The following illustrates sequential optimization.

```
Algorithm Optimize
    $\pi \leftarrow y \leftarrow \lambda$
    While $|\pi| < |S|$ Loop
        Select $p \in S$ Not In $\pi$
        Read $x_p$ into $b$
        $\pi \leftarrow \pi p$
        $y \leftarrow y b$
    EndLoop
    Write $y$
EndAlgorithm Optimize
```

The measure of optimizer performance may be any function of the output, $y$. This is very abstract, but has the advantage of subsuming virtually all practical objective functions.

3 Algorithmic Information Theory

In algorithmic information theory [5] the algorithmic complexity of binary strings is defined in terms of halting programs for a universal computer (e.g., a Turing machine). Programs are required to be self-delimiting binary strings. The universal computer is typically not specified, because any universal computer may
simulate any other with code of constant length. That is, if the length of a program is \( n \) bits, the length of a simulator executing the program is \( n + O(1) \) bits.

Let \( x \) and \( y \) be strings in \( \{0,1\}^* \). The \textit{algorithmic complexity} of \( x \), denoted \( h(x) \), is the length of the shortest program that outputs \( x \) and halts. The \textit{relative complexity} of \( x \) given \( y \), denoted \( h(x \mid y) \), is the length of the shortest program that, given a program “for free” \([5]\) that generates \( y \), outputs \( x \) and halts (see Fig. 1). The \textit{joint complexity} of \( x \) and \( y \) is

\[
h(x, y) = h(x) + h(y \mid x) + O(1). \tag{1}
\]

The \( O(1) \) term absorbs both the increase in length associated with adding a main program to invoke programs for \( x \) and \( y \) as subroutines and the decrease in length associated with reducing the number of self-delimiting objects from two to one. The \textit{mutual complexity} of \( x \) and \( y \) is

\[
h(x : y) = h(x) + h(y) - h(x, y) \tag{2}
\]

\[
= h(x) - h(x \mid y) + O(1) \tag{3}
\]

\[
= h(y) - h(y \mid x) + O(1). \tag{4}
\]

The mutual complexity of \( x \) and \( y \) may be relative to \( z \in \{0,1\}^* \), e.g.,

\[
h(x : y \mid z) = h(x \mid z) - h(x \mid y, z) + O(1). \tag{5}
\]

These identities have close analogs in conventional information theory. In fact, there is an asymptotic equivalence of Shannon entropy and algorithmic complexity (see Thm. 7.3.1 in \([6]\]):

\[
E \left[ h(X^n \mid n) \right] /n \to H(X) \quad \text{as } n \to \infty , \tag{6}
\]

where \( E \) is the expectation operator and \( X^n \) is a sequence of \( n \) i.i.d. random variables distributed as \( X \) on \( \{0,1\} \). Now if functions are drawn uniformly from \( F \), the \( N \) bits in descriptions are i.i.d. uniform on \( \{0,1\} \) with one bit of entropy apiece. Setting \( n = N \) and \( X \sim \text{Uniform}\{0,1\} \) in (6), and assuming large \( N \), the
expected complexity of the description is $N$. Thus the average complexity of descriptions is $N$ bits.

It is perhaps surprising that the average complexity, relative to $N$, of function descriptions is the length of descriptions. The reason is that some descriptions can be generated only by self-delimiting programs that are greater than $N$ in length, even when $N$ is given. In general, for any binary string $x$ of length $n > 0$, the algorithmic complexity is

$$h(x) \leq h(x \mid n) + \log^* n + O(1),$$

where $\log^* n = \log n + \log \log n + \ldots$, with the sum including all positive iterates [6]. An informal explanation of (7) is that $\log^* n$ is the number of bits in a recursive encoding of $n$, and the number of bits in a program to decode $n$ and output the $n$ bits of $x$ is constant.

4 Summary of Theoretical Results

Although the primary concern of the present work is algorithmic information in optimization, several statistical results remain useful not only as an introduction to conservation of performance, but in their own right.

4.1 Statistical Results

Optimizers Generate Bijective Permutations. Recall that optimizers must meet the requirements of Sect. 2. If the values in the input string are independently and identically distributed, the distribution of the output string is identical to that of the input string:

**Theorem 1.** Let $X = X_1, \ldots, X_M$ be a sequence of random variables i.i.d. on $T$. If $\pi : T^M \to T^M$ is the input-output relation of an optimizer, $\pi(X) \sim X$.

If input strings are uniform on $T^M$, then the $M$ values are i.i.d. on $T$, and it follows from the theorem that output strings are uniform on $T^M$. Thus if every string in $T^M$ is input once to an optimizer, every string in $T^M$ occurs once in the output. A similar result holds if functions are represented as binary strings.

**Theorem 2.** The input-output relation $\pi$ of an optimizer is a bijection on $T^M$.

**Corollary 1.** If $\pi'$ is a bijection from $T^M$ to $\{0,1\}^N$, then $\pi' \circ \pi$ is a bijection on $\{0,1\}^N$.

Conservation of Optimizer Performance. It is apparent in Thm. 1 that if $\pi(X) = X$ for all $\pi$, all optimizers share a single distribution of output strings. Thus for any measure of performance that is a function of the sequence of observed values, all optimizers have identically distributed performance. In this sense, performance is conserved when function values are i.i.d. A weaker condition is necessary and sufficient for conservation, however: the distribution of
functions must be such that for \( n = 1, 2, \ldots, M - 1 \) all subsets of \( n \) domain points have identical joint distributions of values [3]. This is essentially a symmetry condition on the joint distribution of random values. Intuitively, conservation obtains when the set of observed values, irrespective of the set of evaluated points, conveys identical information about the value of each unevaluated point.

4.2 Almost All Functions Are Highly Random

Assume that the domain of functions is large, and recall from Sect. 3 that the average complexity of function descriptions \( x \in \{0,1\}^N \) is \( h(x \mid N) = N \). A very short program can, given \( N \) and description \( x \), write \( x \) and halt. This implies that the upper bound on \( h(x \mid N) \) is close to the mean, which suggests in turn that \( h(x \mid N) \approx N \) for almost all descriptions \( x \). Indeed,

\[
|\{x \in \{0,1\}^N \mid h(x \mid N) < N - k\}| < 2^{-k},
\]

(8)

\( 0 < k \leq N \) [6]. Intuitively, the variable \( k \) represents the compression of \( x \) by some program. Eqn. (8) states that the number of descriptions algorithmically compressible by more than \( k \) bits decreases exponentially in \( k \). Because the input-output relation of an optimizer is a bijection on descriptions, this inequality applies to both inputs and outputs.

If \( h(x \mid N) \geq N \) for \( x \in \{0,1\}^N \), \( x \) is algorithmically random. This condition entails all computable tests for randomness [6]. The degree of randomness decreases as the ratio \( h(x \mid N)/N \) decreases from unity. Because \( h(x \mid N) \approx N \) for almost all descriptions \( x \), almost all descriptions are highly random. To appreciate the strength of “almost all,” consider the set of all functions mapping a set of 32-bit numbers to itself. Each function is described by \( 2^{32} \times 2^{32} = 2^{65} \) bits. Compression by more than \( 1/128 \approx 0.8\% \) corresponds to \( k = N/128 = 2^{30} \) in (8), giving \( 2^{-k} = 2^{-1073741824} \). Note also in (8) that more than half of all descriptions satisfy the crisp definition of algorithmic randomness.

4.3 Conservation of Algorithmic Information

For optimizer \( p \), the complexity of any input \( x \) relative to \( p \) is the complexity of the corresponding output \( y \) relative to \( p \). Furthermore, an optimizer cannot add or remove more algorithmic information than is present in itself.

**Theorem 3.** Given \( p \) and \( N \), the difference in complexity of \( x \) and \( y \) is bounded. That is,

\[
h(y \mid p, N) = h(x \mid p, N) + O(1).
\]

(9)

**Theorem 4.** The magnitude of information gain in the output of \( p \) is bounded by the complexity of \( p \). That is,

\[
|h(x \mid N) - h(y \mid N)| + O(1) = |h(x : p \mid N) - h(y : p \mid N)|
\]

\[
\leq \max\{h(x : p \mid N), h(y : p \mid N)\}
\]

(11)

\[
\leq h(p \mid N).
\]

(12)
These relations are depicted in Fig. 2. In practice, optimizers generally halt before evaluating all points in the domain. A lower bound on the complexity of a non-null prefix $y'$ of $y$ is derived from (12) by padding $y'$ with zeros to obtain a string of length $N$. This implicitly changes $n = N - |y'|$ unexamined bits in $x$ to 0, yielding $x'$. Thus

$$h(x' \mid N) - h(y'0^n \mid N) \leq h(p \mid N) + O(1)$$  \hspace{1cm} (13)

$$h(x' \mid N) - h(y' \mid N) - h(0^n \mid y', N) \leq h(p \mid N) + O(1)$$  \hspace{1cm} (14)

$$h(x' \mid N) - h(y' \mid N) \leq h(p \mid N) + \log N + O(1),$$  \hspace{1cm} (15)

where $\log N + O(1)$ is an upper bound on the complexity of $0^n$ relative to $N$. For practical optimizers, $h(p \mid N)$ dominates the right-hand side of (15).

5 Practical Implications

5.1 General Observations

Statistical results on conservation of optimizer performance have generally been ignored by practitioners of evolutionary computation, many of whom assume (perhaps correctly, as the next subsection suggests) that their functions are drawn from distributions that do not have the symmetry property necessary for statistical conservation. The new results based upon algorithmic complexity give a much stronger, distribution-free form of conservation. For any performance measure that is a function of the sequence of observed values, almost all nonredundant, exhaustive, algorithmic optimizers have approximately equal performance on almost all functions.

It is also significant that the complexity of an optimizer bounds the magnitude of complexity gain (and loss) in processing. Put simply, short optimization
codes are ineffective when performance requires high information gain. This is not typically the case, however, because almost all function descriptions are highly random, with little useful information to gain.

The ubiquity of highly random functions is easily mistaken for a negative result. In fact, good values are likely to be obtained by evaluating relatively small subsets of the domain, selected in any manner whatsoever [3]. Low compressibility of the description implies that all values in the range of the function have approximately the same number of preimages. There is little or no order in the arrangement of these values in the description. Thus a random sample of points is as good as any other, and should be expected to produce as many good values as bad values. Only an orderly description can "hide" optima from an optimizer.

The relationship between algorithmic complexity and optimization is subtle. It is usually the case that much of the complexity of the input and the output are unrelated to the objective. Sometimes $h(y' \mid N) > h(x' \mid N)$ when performance is optimal by some measure. For instance, if $y_1$ is the least value in $x$, the remaining values in $x'$ may be permuted to obtain a $y'$ of higher complexity. In the present state of the theory, there is no way to "tease out" objective-relevant complexity from irrelevant complexity.

5.2 Algorithmic Complexity of Example Functions

There is not yet a simple answer to the question of which problem classes permit the design of superior optimizers. This subsection illustrates some of the considerations through examples.

Needle in a Haystack Functions. Even when the functions in a class are highly compressible, there is no guarantee that any optimizer is superior to a point enumerator. Consider the set of all "needle in a haystack" functions from $S$ to $\{0,1\}$. Each function assigns 1 to exactly one domain point and 0 to all the rest. Despite the very high compressibility of the functions, optimizer performance is conserved when functions are drawn uniformly from the set [3]. This case of conservation has been explained in terms of the statistical information, but not algorithmic information.

Note that the average number of points that must be explored to maximize these functions is $N/2$, with any honest optimizer requiring each of 1, 2, $\ldots$, $N$ trials for the functions. The mean is the worst possible for any class of functions, and the entropy of the distribution of required trials is also the greatest possible. On the other hand, the minimum is found on the first trial for all but one of the functions, and then it is found at the second point examined.

One might suppose that complexity reduction makes one form of optimization easier at the expense of the other. This is refuted by adding a "pin" to the haystack. That is, let the range of the functions be $\{-1,0,1\}$. For each function, there is exactly one $w \in S$ such that $f(w) = -1$ and exactly one $x \in S$ such that $f(x) = 1$. Now minimization is as hard as maximization, and the complexity of functions in the class has increased only slightly.
Quadratic Functions. Every quadratic function is computed by a program of lower complexity than that of, say, a Newtonian optimizer. One might guess that this is at the root of the optimizer’s efficiency on quadratics, but note that the information gain is not as large as it might seem. After going immediately to the optimum of a quadratic, the Newtonian optimizer is not equipped to go steadily to worse and worse points. With the modification of the optimizer to include an enumerator, the points can be visited in reverse order of goodness after the optimum is located. This illustrates that not all information about a function is of equal value. An optimizer may provide little information overall, but if it quickly provides the optimum of a function, it is extremely effective. The two versions of the Newtonian optimizer illustrate that while complexity greater than that of all test functions is not sufficient for high gain of information in every case, it is necessary.

Sampled, Quantized Polynomials. Let \( K = |T| \). For \( n = 0, 1, \ldots, K - 1 \), let \( P_n \) be the set of all \( f \in F \) such that for some polynomial \( q \) of order \( n \) or less \( f(x) = [q(x)] \) for all \( x \in T \), where \([\cdot]\) denotes rounding to the nearest element of \( T \). Any function on \( T \) can be fit exactly by a polynomial of order \( K - 1 \) or less, so \( P_{K-1} = F \). Thus optimizer performance is conserved when test functions are drawn uniformly from \( P_{K-1} \).

Optimizer performance is not conserved for all \( n < K - 1 \), however. Reducing \( n \) makes the polynomials corresponding to \( P_n \) smoother. On the face of it, this is equivalent to making nearby points’ random values more highly correlated than distant points’ values, but the fact that polynomial values of arbitrarily large magnitude are rounded to \(-b\) and \( \max T \) complicates formal reasoning. Although it is surmised that optimizer performance is not conserved for any \( n < K - 1 \), the present work addresses just the simplest case of \( n = 1 \).

Theorem 5. Let random variable \( F \) be distributed on \( P_1 \), with \( \Pr\{F = f\} > 0 \) for all \( f \in P_1 \). If functions are randomly drawn according to \( F \), then optimizer performance is not conserved.

Proof. Let \( c = \max T \), and assume that \( c > 1 \). The functions in \( P_1 \) correspond to linear polynomials. Thus for realization \( f \) of \( F \),

\[
\Pr\{f(-c) = -c, f(0) = 0, f(c) = c\} > 0 ,
\]

but

\[
\Pr\{f(-c) = -c, f(1) = 0, f(c) = c\} = 0
\]

because the specified points are not colinear. Thus the necessary condition that all equal-sized subsets of random values have identical joint distributions is violated, and performance is not conserved.

Taylor Polynomials. In each \( P_n \), some of the elements correspond to Taylor polynomials. Low-order Taylor polynomials give accurate approximations to many of the common mathematical functions, and they are utilized heavily in
computer math libraries. Furthermore, a large number of the test functions studied in evolutionary computation can be approximated accurately with low-order Taylor polynomials. The import is that the results on sampled, quantized polynomials apply to a wide range of functions that are not immediately thought of as polynomials.

5.3 Performance Evaluation

The Taylor polynomials underlying many of the test functions studied in evolutionary optimization are much less complex than most, if not all, evolutionary and genetic algorithms. Thus substantive complexity reduction in processing is feasible. It bears mention that complex functions are not as likely to survive in mathematical culture as less complex functions. Thus it is misleading to treat functions from mathematics as generally representative.

It is also potentially misleading to present the “fitness landscape” of a function as an indication of its complexity. Consider that fractal images can be generated by programs of very low algorithmic complexity.

It has been shown that the non-constant functions of lowest algorithmic complexity are the easiest and the hardest to optimize. It is also known that the functions of highest complexity are relatively easy to optimize. For there truly to be an optimization problem, there must be some structure in the function, and this means that the function must be algorithmically compressible to a substantial degree. Thus it is appropriate to evaluate algorithms with functions of lower complexity. However, this does not imply that testing almost exclusively with low-order Taylor polynomials and combinations thereof is appropriate.

6 Conclusion

Algorithmic randomness is pervasive in descriptions of functions from finite domains to finite codomains. Optimizer design is trivial for descriptions of high complexity. In practice, the complexity of descriptions is generally much lower than that of the optimizer. Thus widespread impressions of the nature of optimization are based upon an unacknowledged restriction of the problem domain to highly atypical functions. It is essential that practitioners clearly identify the classes of functions they address, keeping in mind that with the finite precision of a digital computer, many functions from mathematics are indistinguishable from low-order polynomials.

It has been established that performance is conserved for a uniform distribution of sampled and quantized polynomials, provided the maximum order is sufficiently high. It has also been established that performance is not conserved if the maximum order is sufficiently low. Future work should address the question of how the performance distributions of optimizers change as the maximum order is varied.

Another issue for future work is how to address the gain of information relevant to the objective. Although treatment of optimization in terms of algorithmic
complexity has provided substantive insights, that good performance does not equate neatly with complexity reduction is a shortcoming.

References


Large Deviations, Evolutionary Computation and Comparisons of Algorithms

Olivier François

Laboratoire de Modélisation et Calcul, BP 53, 38041 Grenoble cedex 9, France

WWW home page: http://www-lmc.imag.fr/lmc-sms/Olivier.Francois

Abstract. This paper summarizes large deviations results for Markov chains with exponential transitions, and discusses their applications in comparing stochastic optimization algorithms. In the second half, two specific algorithms will be focussed on: Mutation/Selection vs parallel simulated annealing. Conditions which tell when an algorithm should be preferred to the other will be given. These conditions combine the parameters of both algorithms and a number of relevant geometric quantities.

1 Introduction

Large deviations is a mathematical theory that deals with estimating the probabilities of rare events thanks to a functional called entropy. This theory is relevant to Markov models with rare transitions, and provides useful insights on the behavior of the chain. Many stochastic optimization algorithms such as simulated annealing, genetic algorithms, and even some variations on evolution strategies can be fitted into this framework, up to minor modifications of their internal dynamics. Simulated annealing (SA) is one of the most popular technique in combinatorial optimization [1]. This algorithm has received much attention in the framework of Markov chains with rare transitions [16], [8], [2]. The relationship between genetic algorithms (GAs) and SA has been underscored several times [4], [11], [9], [3], [5], [13]. For these algorithms, the dynamics get usually caught in metastable states (e.g. local minima), and exit from these states according to typical time scales and trajectories [10]. This paper summarizes known results concerning exit times and success probabilities for Markov chains with rare transitions, and emphasizes their relationship to evolutionary computation.

Consider a finite set $E$, and the minimization problem associated to a non-negative one-to-one function $f$ on $E$. (Let $a^*$ denote the absolute minimum of $f$, and assume that $f(a^*) = 0$.) The set of all population vectors is denoted by $X$, and is used as the state space for the models. The paper is organized as follows. Section 2 gives the mathematical definitions associated with Markov chains with exponential transitions, and some examples of minimization algorithms are presented within this setting. Section 3 recalls important large deviations estimates for the hitting time of the absolute minimum and for success probabilities. The rest of the paper is concerned with comparisons between algorithms. In particular, this paper states conditions by which an evolutionary strategy should be
preferred to parallel simulated annealing. The value of the obtained conditions are illustrated on a standard test problem.

2 Markov chains with exponential transitions

2.1 Large deviations functionals

In studying a Markov chain model, one often needs to evaluate the chance that the chain reaches a given point \( y \), in less than \( r \) steps, starting from an arbitrary initial point \( x \in X \). To manage the computation, one needs to sum the chances that the chain follows each trajectory from \( x \) to \( y \). Let \( \gamma : x_0 = x \to x_1 \to \ldots \to x_r = y \) be a such trajectory. Then, the probability writes as follows

\[
\text{Prob} = \sum_{\gamma : |\gamma| \leq r} \prod_{i \in \gamma} p(x_i, x_{i+1}),
\]

where \( p(x_i, x_{i+1}) \) denotes the probability of an elementary transition. This global computation may be hardly feasible as the state space becomes critically large. In this situation, some simplifications may help. Assume, for instance, that the chain has exponential transitions

\[
C(x, y) < \infty, \quad T > 0,
\]

with \( C(x, y) \) a nonnegative cost functional, and \( c_1, c_2 \) some positive constants. A transition between \( x \) and \( y \) is said to be admissible iff \( C(x, y) < \infty \). The parameter \( T \) is called the temperature, and must be thought as being small. In order to estimate the value of the sum, the Laplace method can be applied. For small \( T \), one actually has

\[
\text{Prob} \approx \exp(- \min_{\gamma : |\gamma| \leq r} C(\gamma)/T)
\]

with

\[
C(\gamma) = \sum_{x_i \to x_{i+1} \in \gamma} C(x_i, x_{i+1}),
\]

where the symbol \( \approx \) must be understood as a logarithmic equivalent. Loosely speaking, considering low temperatures amounts to replace the classical \((+, \times)\) algebra by a \((\min, +)\) algebra, in which the computation could be simpler. In the large deviations formalism, the cost functional \( C(\gamma) \) is termed the entropy, as the chain follows those trajectories with minimal entropy [7]. The next paragraphs show why Markov chains with exponential transitions are relevant to minimization and evolutionary computation algorithms.

2.2 Simulated annealing

To emphasize the relationship to evolutionary computation, consider \( n \) independent versions of SA running in parallel. Interacting versions can be studied as
well [11]. However, Trouvé [14] proved that the convergence exponent for parallel annealing based on periodically interacting searches is always worse than for independent searches.

Here is a description of the algorithm. Let \( x = (x^1, \ldots, x^n) \) and \( y = (y^1, \ldots, y^n) \) be two population vectors of size \( n \), and define the transition \( x \rightarrow y \) as \( n \) simultaneous independent uses of the well-known Metropolis rule. Start from \( x = x_0 \in X \) and repeat the following steps. For each individual \( x^i \), create one offspring \( y^i \) by mutation, and compute the difference \( \Delta f(x^i, y^i) = f(y^i) - f(x^i) \). If \( \Delta f(x^i, y^i) \leq 0 \), replace the parent, i.e., make \( x^i := y^i \). Otherwise, replace the parent with probability \( p = e^{-\frac{1}{T} \Delta f(x^i, y^i)} \). Doing so, a Markov chain model with exponential transitions can be defined. This model can be associated with the cost

\[
C(x, y) = \sum_{i=1}^{n} C_i(x^i, y^i),
\]

where \( C_i(x^i, y^i) = \max\{f(y^i) - f(x^i), 0\} \) when the transition \( x \rightarrow y \) is admissible.

### 2.3 Genetic algorithms

To use large deviations arguments, the classical GA must be slightly modified. The modified GA can be described as follows [3]. Start from \( x = x_0 \in X \) and repeat the following steps. For each individual \( x^i \), \( i = 1, \ldots, n \), create one offspring \( y^i \) by mutation with probability \( p_{\text{mut}} = e^{-\frac{1}{T} \Delta f(x^i, y^i)} \). Otherwise, take \( y^i \) equal to \( x^i \). Now, create a sample \( (y^i)_{i=1,\ldots,n} \) according to the following probability distribution

\[
P(Y = \zeta^i) \propto \exp(-\theta f(\zeta^i)/T), \quad \theta > 0.
\]

Then, replace \( x^i \) by \( y^i \), for all \( i = 1, \ldots, n \).

This description actually corresponds to the genetic algorithm without crossover and with Boltzmann roulette selection. The vector \( \zeta^i \) represents the intermediate population vector obtained after the mutation step has been applied. The parameter \( \theta \) represents the selection intensity, and allows balancing the respective weights of mutation and selection. The Boltzmann distribution makes the population converge on the best offspring \( \zeta^* \) (as \( T \) goes to zero), and therefore warrants that some minimization process actually takes place. Of course, the usual roulette selection could be used instead of the Boltzmann rule. However, the process would, in the exponential transition setting, behave as \( n \) independent random walks, and run against our interests. As far as Boltzmann selection is concerned, a Markov chain model can be associated to the cost

\[
C(x, y) = \min_{\zeta} \left\{ \sum_{i=1}^{n} (1 - \delta(x^i, \zeta^i)) + \theta \sum_{i=1}^{n} f(y^i) - f(\zeta^*) \right\},
\]

when the transition \( x \rightarrow y \) is admissible (\( \delta(a, b) \) denotes the Kronecker symbol). Here, admissibility depends on the way the mutation operator has been
implemented. The contribution \( \sum_{i=1}^{n}(1 - \delta(x^i, \zeta^i)) \) merely counts the number of offspring by mutation. The contribution \( \theta \sum_{i=1}^{n} f(y^i) - f(\zeta^i) \) represents the cost for not selecting the best offspring. Formally, there are no additional difficulties in including a crossover operator, if we assume that there is positive probability that the operator leaves two individuals unchanged when they are chosen for mating (which means that \( p_{\text{cross}} < 1 \)).

2.4 Others

Considerably much simpler large deviations functionals can be defined than those associated with genetic algorithms. For example, set

\[
C(x, y) = \sum_{i=1}^{n}(1 - \delta(x^*, y^i)),
\]

where

\[
x^* = \arg\min \{f(x^i); i = 1, \ldots, n\}.
\]

This cost functional counts those individuals in \( y \) which differ from \( x^* \). A Markovian algorithm can be described as follows. Start from \( x = x_0 \in X \) and repeat the following steps. Find \( x^* \), the best individual in the current population. Then, for each individual \( x^i, i = 1, \ldots, n \), create one offspring \( y^i \) by mutation with probability \( p_{\text{mut}} = e^{-1/T} \). Otherwise, define \( y^i \) as being equal to \( x^* \). This algorithm corresponds to a non-ordered version of the Mutation/Selection (MS) strategy presented in [5]. This strategy is based upon a small random number of offspring by mutation at each generation, whereas this number is usually constant in other strategies.

Relying on similar algorithmic principles, more elaborate cost functionals could also be considered. Let, for instance, \( x \) and \( y \) be two population vectors and \( I(x, y) \) be the subset of labels \( \ell \) for which \( y^i \not\in \{x^1, \ldots, x^n\} \). Perform mutation as before, and base selection upon the Boltzmann roulette as in GAs. Then, the cost of an admissible transition can be described as

\[
C(x, y) = \#I(x, y) + \theta \sum_{i \in I(x, y)} (f(y^i) - f(x^*)).
\]

Now, the term \( \#I(x, y) \) counts the number of offspring by mutation, and the other term represents the antiselection cost. To implement these algorithms, mutation and selection are independently and simultaneously applied to the same population vector. In contrast, selection in GAs is usually applied to the offspring of mutation.

3 Hitting time and error probabilities

One way of dealing with the convergence issue in evolutionary computation is by using estimates on the hitting time of optimal solutions (as soon as they are
available) [12]. The success probability (i.e., the probability for reaching the absolute minimum) is also often asserted as a useful measure of performance. While these quantities are generally difficult to evaluate for evolutionary algorithms, many results are known for Markov chains with exponential transitions. This section gives an account of the main results known about hitting times and error probabilities for these chains. Assume that the Markov chain defined in equation (2) is ergodic, i.e., converges to a well-defined distribution $\pi_T$. First of all, an energy can be associated with a Markov chain with exponential transition in a natural way $V(x) = -\lim_{T \to 0} T \log \pi_T(x)$, ($x \in X$). For single SA processes, the energy equals the objective function. For $n$ independent SA running in parallel, it equals

$$V(x) = \sum_{i=1}^{n} f(x^i). \quad (11)$$

In general, the energy is much more complex. To give a definition, consider all minimal trajectories leading to $x$, and the subgraph of the chain containing these minimal trajectories. This subgraph is a tree $T_x$ rooted at $x$ (otherwise minimality could be contradicted). Following [7], the energy can be defined as

$$V(x) = \sum_{(y \to z) \in T_x} C(y, z), \quad (12)$$

up to an additive constant (which corresponds to the minimal value $V_*$ computed over all $x$). In other words, the energy corresponds to the cost of a minimal spanning tree rooted at $x$.

### 3.1 Exit (and hitting) times

In describing the dynamics of a Markov chain with exponential transitions, subsets called cycles play a central role [7], [8], [3]. A subset $C \subset X$ is a cycle if either it consists of a single population vector, or for all $x, y$ in $C$, the expected number of “cyclic” visits to $x$ followed by $y$ before exiting from $C$ is exponential

$$E[N_{xy}(C)] \approx e^{K_{xy}(C)/T}, \quad K_{xy}(C) > 0. \quad (13)$$

As a consequence, a cycle should be systematically explored (on an exponential time scale) before the chain exits and proceeds with another cycle. This concept is closely related to the phenomenon of metastability known in statistical mechanics. Here is a trajectorial definition of cycles, which is more amenable to a mathematical analysis [15], [3]. For $x, y \in X$, $x \neq y$, and each trajectory $\gamma_{xy}$, define the elevation as

$$H(\gamma_{xy}) = \max_{0 \leq k < r} \left\{ V(x_k) + C(x_k, x_{k+1}) \right\}, \quad (14)$$

where the maximum is taken over all vertices in $\gamma_{xy}$. Let $H(x, y)$ be the lowest possible value of $H(\gamma_{xy})$ over all self-avoiding trajectories $\gamma_{xy}$ from $x$ to $y$. The
quantity $H(x, y)$ is called the *communication altitude*, and is symmetrical in $x$ and $y$ [15]. Now, let $\lambda \geq 0$ and $V_\lambda = \{ x \in X : V(x) \leq \lambda \}$. Say that $x$ and $y$ communicate at height $\lambda$ in $V_\lambda$ if $H(x, y) \leq \lambda$. A subset $C \subset X$ is a cycle if all populations are able to communicate at height $\lambda$ for some $\lambda > 0$.

As evolutionary algorithms can reach uniform populations at null cost, the induced chain (on the set of uniform populations) gives a right picture of the dynamics. While the hierarchy of cycles may be extremely complex in general, a remarkable fact is that, for large population sizes of GAs and MS, a cycle which does not contain the minimal population $a^*$ reduces to a single uniform population. For GAs, the critical population size $n_*$ has been estimated by Cerf [3]. For MS, the critical value is lower than [5]

$$n_* = \max_{a \neq a^*} d(a, a^*),$$

where $d(a, b)$ is the minimal number of mutation steps needed to reach $a$ from $b$ in $E$.

Now, denote by $\tau_C$ the exit time of the subset $C$ (the hitting time of $C$). The expected value can be computed as

$$E[\tau_C] \approx e^{H_e(C)/T},$$

where, according to [15], $H_e(C)$ (*the exit height of C*) is equal to

$$H_e(C) = \max_{x \notin C} \min_{y \in C} \{ H(x, y) - V(x) \}.$$  \hfill (17)

Let $\tau_*$ denote the hitting time of the absolute minimum. The expected value of $\tau_*$ can therefore be approximated as

$$E[\tau_*) \approx e^{H_1/T},$$

where

$$H_1 = \max \{ H_e(C) : C \text{ cycle not intersecting } V_* \},$$

and $V_* = \arg \min V$. For GAs and MS, $H_1$ can be given as

$$H_1 = \max_{x \neq (a^*)} \{ H(x, (a^*)) - V(x) \},$$

whenever $n > n_*$, and $(a^*)$ denotes the uniform population $(a^*, \ldots, a^*)$.

3.2 Error probability

Consider an arbitrary decreasing sequence of temperatures $(T(t))$, and let $(X_t)$ denote the non-homogeneous Markov chain obtained when $T$ is updated at each generation. As proved in [15], the error probability satisfies

$$\frac{R_1}{t_{a^*}} \leq \inf_{0 \leq \tau(T(t)) \leq \ldots \leq T(1)} \max_{x \in X} \Pr(X_t \neq a^* \mid X_0 = x) \leq \frac{R_2}{t_{a^*}},$$

\hfill (21)
for some constants \( R_1, R_2 > 0 \), with \( \alpha_* \) (the convergence exponent) given by

\[
\alpha_* = \min\left\{ \frac{V(C) - V_*}{H_\alpha(C)} ; C \text{ is a cycle not intersecting } V_* \right\},
\]

where \( V(C) \) is the minimum of \( V \) over the subset \( C \). The convergence exponent describes the minimum error of the algorithm after a fixed number of generations, computed over all (maybe nonconstant) decreasing temperature schedules. This constant is sometimes viewed as an index of difficulty associated with the energy landscape [2].

### 3.3 A small example

Consider the minimization problem defined on \( E = \{0, 1, 2, 3, 4, 5\} \) where the objective function is given by \( f(0) = 2, f(1) = 3, f(2) = 8, f(3) = 1, f(4) = 0, f(5) = 5 \). The basic mutation operation consists of choosing \( k + 1 \) or \( k - 1 \) with equal chances (if the value is in \( E \)) when a parent individual is in state \( k \).

The values of \( H_1 \) and \( \alpha_* \) are hereafter discussed for the three algorithms presented in section 2.

**Simulated annealing.** For the Metropolis algorithm (with the population size equal to one), the hierarchy of cycles can be described as

\[
C = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{0, 1\}, \{3, 4\}, \{3, 4, 5\}, \{0, 1, 2, 3, 4, 5\}\}.
\]

The value of \( H_1 \) is given by Hajek's critical value, which corresponds to the exit height of the deepest non global minimum: \( H_1 = 6 \), and we have \( \alpha_* = 1/3 \). Hence, the simulated annealing process has at best a convergence speed of order \( t^{1/3} \).

Considering the best value obtained from \( n \) independent runs of the Metropolis chain yields a convergence exponent equal to \( \alpha_* = n/3 \), which is exactly \( n \) times the value of a single process (this results from independence). In addition, the value of the critical height for the parallel version is still \( H_1 = 6 \).

**Genetic algorithm.** Consider now the genetic algorithm without crossover, and use large population sizes. To compute \( H_1 \), one needs to evaluate the elevation of the trajectory \( (0) \rightarrow \ldots \rightarrow (3) \), where the whole population is started from 0. This only requires sending a single explorer to vertex 3, as selection is able to make the whole population converge on this vertex. The cost is \( H_1 = 3 + 7\theta \), as three mutations are needed, and the first two have antiselection costs equal to 1 and 6.

The value of \( \alpha_* \) is more complex to compute. To obtain an upper bound, one can evaluate the cost of the optimal trajectory from (4) to (0), then subtract \( H_1 \) and divide by \( H_1 \). The most probable trajectories are as follows.

- **Path 1:** Mutate the whole population to (2), and then send a single explorer to 0. The cost is \( C(\gamma_1) = 2n + 2 \).
- Path 2: Send a single explorer to 0, and use selection to call the rest of the population. The cost is \( C(\gamma_2) = 4 + 12\theta + 2\theta n \).
- Path 3: Consider the trajectory: \((4) \to (2, 4, \ldots, 4) \to (2, 5, \ldots, 5) \to (1) \to (0)\). The cost is \( C(\gamma_3) = n + 4\theta + 3 \).

For large \( n \) and \( \theta \geq 1/2 \), the minimal cost is obtained for path \( \gamma_3 \), whereas path \( \gamma_2 \) is the best for \( \theta < 1/2 \). For large \( n \) and \( \theta \geq 1/2 \) (the best case), the convergence exponent is lower than \( (n - 3\theta)/(3 + 7\theta) \), which is worse than \( n \) parallel SA. As a general result, Cerf [3] proved that the optimal exponent increases linearly with population sizes. The above example nevertheless shows the difficulty in obtaining precise values for the convergence exponent of GAs.

**Mutation/Selection.** As far as MS is concerned, the situation is clearer than for GAs. For population sizes greater than \( n_\ast \) (given in equation 15), [5], [6] show that \( H_1 = d_\ast \) where

\[
\alpha_\ast = \max_{a \neq a^\ast} \min_{b : f(b) < f(a)} d(a, b),
\]

and the convergence exponent satisfies

\[
\frac{n - n_\ast}{d_\ast} \leq \alpha_\ast \leq \frac{n(n_\ast + 1 - d_\ast) - 1}{d_\ast}.
\]

Here, we have \( H_1 = 3 \) and \( n_\ast = 4 \). Thus \( (n - 4)/3 \leq \alpha_\ast \leq (2n - 1)/3 \). The minimal path from \((4)\) to \((0)\) is as follows: Send one explorer to 3 and the rest of the population to 5 (i.e., obtain \((3, 5, \ldots, 5)\)). Then, send the first explorer to 0, and use selection. The cost is \( C(\gamma) = n + 3 \). Hence, one has \( \alpha_\ast = n/3 \), and the algorithm’s performance is the same as \( n \) independent Metropolis chains. However, MS hits \( a^\ast \) faster than the parallel SA (3 < 6). GAs have the worst convergence exponents and hitting times.

### 4 Simulated annealing vs Mutation/Selection

#### 4.1 Comparisons

Algorithm \( A \) should outperform algorithm \( B \) when the success probability of \( A \) is higher than that of \( B \), i.e., \( \alpha_\ast(A) > \alpha_\ast(B) \). In this section, the focus is on SA and MS which provide a simpler analysis than GAs. Fix the population size \( n > n_\ast \), and consider \( n \) independent runs of the simulated annealing algorithm (called \( \text{SA}(n) \)). For \( \text{SA}(n) \), one actually obtains \( \alpha_\ast(\text{SA}(n)) < n f_\ast/h_\ast \), where \( h_\ast \) is Hajek’s critical height, \( f_\ast \) the value of \( f \) at the bottom of the deepest non-global minimum, and \( d_\ast = \min\{f(a) - f(a^\ast); a \neq a^\ast \in E\} \). According to equation (24), MS should outperform \( \text{SA}(n) \) when \( h_\ast > f_\ast d_\ast \), and \( n \) is chosen so that

\[
n(1 - \frac{f_\ast d_\ast}{h_\ast}) > n_\ast.
\]
4.2 Rastrigin’s function

In this paragraph, the relevance of condition (25) will be illustrated through the study of a classical test problem. We have used the Rastrigin’s function in \( D = 5 \) dimensions

\[
f(a) = \sum_{j=1}^{D} a_j^2 - 10(\cos(2\pi a_j) - 1), \quad a \in E = [-5,+5]^D. \tag{26}
\]

The minima are located on the integer lattice, and the absolute minimum is at the origin.

To check condition (25), mutations have been performed by adding a uniform variable taken in the interval \([-0.5,+0.5]\) to a randomly chosen coordinate of an individual. The deepest non global minimum is located at the point with all coordinates equal to 1, and is evaluated to \( f_* = D \). To compute the critical height of \( \text{SA}(n) \), consider an exit trajectory that starts from \((1, \ldots, 1)\) and goes to \((0,1,\ldots,1)\). This trajectory must cross a saddle point at \((0.5,1,\ldots,1)\). The cost is thus equal to \( h_* = 19.25 \). As concerns \( \text{MS} \), any nonoptimal solution can be improved through two successive mutation steps. Hence, one obtains \( d_* = 2 \). Reaching the most distant point from the origin requires at least \( 2D \) mutation steps. Hence, one has \( n_* = 2D \). Putting together these values in equation (25) leads to the following claim. \( \text{MS} \) would outperform \( \text{SA}(n) \) as soon as \( n > 20 \).

To check this, numerical experiments have been performed with \( T = 0.3 \) and \( n = 21 \) individuals. Within 80,000 fitness evaluations, about 80% of \( \text{MS} \) runs reached the fitness value \( f = 0.01 \). As 80,000 fitness evaluations represents about 90,000 generations, the results were compared to those obtained after 90,000 generations of \( \text{SA}(21) \). The empirical success probability for \( \text{SA}(21) \) over 20 repetitions was null.

The parameter settings used in this comparison are nevertheless far from optimal settings for the both algorithms. When the parameters are set to \( n = 10 \), \( p_{\text{mut}} = 0.01 \), and the mutation range \( r = 1.2 \), \( \text{MS} \) reaches a value lower than 0.01 after 2,000 fitness evaluations. Using optimal settings should be the standard framework in making comparisons. However, these optimal settings are usually unknown (as they depend on the problem under study), and their investigation is beyond the scope of this paper.

5 Conclusion

The large deviation formalism is a powerful approach in analysing Markov models of evolutionary computation. This theory emphasizes the critical influence of some geometric quantities on convergence. It must be pointed out that the theory is asymptotical: mutation probabilities must be small. Nevertheless, this assumption is often checked in practice. In addition, large deviations theory is a unifying formalism that can serve as a basis to perform theoretical comparisons between algorithms. Several couplings of mutation and selection can be investigated which correspond to different entropy functionals. From this perspective,
designing efficient algorithms amounts to create cost functionals according to the available geometric information on the problem to be solved. Therefore, implementing an algorithm amounts to simulate the Markov chain associated with these costs. To conclude, this theory provides a useful tool in deciding which algorithm should be chosen to solve a given problem.

References

On the Choice of the Mutation Probability for the (1+1) EA*

Thomas Jansen and Ingo Wegener

FB 4, LS 2, Univ. Dortmund, 44221 Dortmund, Germany
{jansen, wegener}@ls2.cs.uni-dortmund.de

Abstract. When evolutionary algorithms are used for function optimization, they perform a heuristic search that is influenced by many parameters. Here, the choice of the mutation probability is investigated. It is shown for a non-trivial example function that the most recommended choice for the mutation probability 1/n is by far not optimal, i.e., it leads to a superpolynomial running time while another choice of the mutation probability leads to a search algorithm with expected polynomial running time. Furthermore, a simple evolutionary algorithm with an extremely simple dynamic mutation probability scheme is suggested to overcome the difficulty of finding a proper setting for the mutation probability.

1 Introduction

Evolutionary algorithms (EAs) are randomized search heuristics that are often applied to the task of function optimization. They are influenced by many parameters. Though EAs are in general assumed to be robust and more or less insensitive to the setting of the parameters, it is a well-known fact that the choice of the parameter settings has great impact on the success and efficiency of the search. We consider optimization by means of EAs and look for parameter settings that allow an efficient exact optimization of a given objective function that we consider to be unknown to the algorithm.

We concentrate on maximization of discrete functions and assume that the objective function is some function $f: \{0,1\}^n \rightarrow \mathbb{R}$. We consider the (1 + 1) evolutionary algorithm ((1 + 1) EA). It is a very simple EA using only mutation and selection. In the next section, we give a formal definition of the (1 + 1) EA and discuss known examples that demonstrate that the most recommended choice for the mutation probability is by far not optimal. These examples are not of practical relevance, since also the best mutation probability leads to an algorithm with an expected exponential running time. In Section 3, we present a function with the following properties. The (1+1) EA with a mutation probability which is substantially larger or smaller than $(\log n)/n$ needs superpolynomial time with overwhelming probability while mutation probabilities growing as $(\log n)/n$ lead to a (1+1) EA which finds the optimum in an expected polynomial number of

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).
steps and even in a polynomial number of steps with overwhelming probability. In Section 4, we introduce a variant of the (1+1) EA that employs an extremely simple dynamic variation scheme for the mutation probability. We prove that this dynamic EA optimizes the example function efficiently. We finish with some concluding remarks.

2 The (1+1) EA

As already mentioned we concentrate on the maximization of functions \( f: \{0,1\}^n \rightarrow \mathbb{R} \) by the (1+1) EA. This algorithm is often subject to theoretical studies \([3, 5, 9, 11]\). We present results using the common notions \( O, \Omega, \) and \( \Theta \), where for two functions \( s, t: \mathbb{N} \rightarrow \mathbb{R} \) we say that \( s(n) = O(t(n)) \), if there exist constants \( n_0 \in \mathbb{N} \) and \( c \in \mathbb{R}^+ \), such that for all \( n \geq n_0 \) we have that \( s(n) \leq c \cdot t(n) \) holds. We say that \( s(n) = \Omega(t(n)) \), if \( t(n) = \Theta(t(n)) \). Finally, we say \( s(n) = \Theta(t(n)) \) if both, \( s(n) = O(t(n)) \) and \( s(n) = \Theta(t(n)) \) hold.

Algorithm 1 (The (1+1) EA).

1. Choose \( x \in \{0,1\}^n \) uniformly at random.
2. Fix \( p(n) \in (0,1/2] \).
3. \( y := x \).
4. Flip each bit in \( y \) independently with probability \( p(n) \).
5. If \( f(y) > f(x) \), set \( x := y \).
6. Continue at 3.

It is a common experience that simple hill-climbers are often able to find solutions that are at least comparable to those of more sophisticated evolutionary algorithms \([8]\). Setting \( p(n) = 1/n \) implies that during one mutation step on average one bit flips. Thus, the (1+1) EA may be regarded as a kind of randomized hill-climber. In fact, the most recommended fixed choice for the mutation probability \( p(n) \) is \( 1/n \) [1,9]. For linear functions choosing \( p(n) = \Theta(1/n) \) can be proven to be optimal [3]. Sometimes it is conjectured that \( p(n) = \Theta(1/n) \) may be optimal for all functions.

First of all, we present known examples where the choice \( p(n) = 1/n \) for the mutation probability is much worse than the choice \( p(n) = 1/2 \). The choice \( p(n) = 1/2 \) changes the (1+1) EA into a random search. We only consider functions with a unique global maximum. Then the probability of choosing \( x_{\text{opt}} \) equals \( 2^{-n} \) for each step independently of the history. Hence, the expected running time equals \( 2^n \).

The first example is the well-known “needle in a haystack” function, that yields the function value \( 0 \) for all \( x \in \{0,1\}^n \setminus \{(1,\ldots,1)\} \) and \( 1 \) else. A search algorithm gets no information until it finds the global optimum by chance. The mutation probability \( p(n) = 1/n \) is worse than \( p(n) = 1/2 \), since it increases the probability to investigate the same strings again and again. Indeed, Garnier, Kallel, and Schoenauer [5] have shown that the expected running time of the (1+1) EA with \( p(n) = 1/n \) equals \( 2^n/(1 - e^{-1}) \approx 1.582 \cdot 2^n \) and is by a constant factor larger than the expected running time of random search.
The second example is the “trap” function with \( \text{TRAP}(x) = \text{ONEMAX}(x) = x_1 + \cdots + x_n \) for all \( x \in \{0,1\}^n \setminus \{(0,\ldots,0)\} \) and \( \text{TRAP}(0,\ldots,0) = n + 1 \). Hence, \( \text{TRAP} \) differs from the linear function \( \text{ONEMAX} \) only for the all zero string which is optimal. A search strategy for \( \text{ONEMAX} \) should find the all one string quickly. As long as the all zero string is not found by chance, the \((1+1)\) EA works on \( \text{TRAP} \) as on \( \text{ONEMAX} \) and, therefore, gets hints in the wrong direction. For this reason, \( \text{TRAP} \) is called a strongly deceptive function. Droste, Jansen, and Wegener [4] have proved that the expected number of steps of the \((1+1)\) EA with \( p(n) = 1/n \) on \( \text{TRAP} \) is \( \Theta(n^n) \) which is by the exponential factor \( \Theta((n/2)^n) \) larger than the expected time of random search. Random search is impractical even for rather small \( n \). Hence, if random search beats an evolutionary algorithm for some function, both algorithms are too bad for this function. The algorithms will be stopped in applications before the global optimum is found.

We are looking for examples where the running time of the \((1+1)\) EA can be drastically reduced to some polynomial number of steps when setting the mutation probability \( p(n) \) to some other value than \( 1/n \). Such an example is given in the following section.

3 A Non-Trivial Example Function

Here, we introduce an objective function with the following properties. Setting \( p(n) = 1/n \) implies that the \((1+1)\) EA will almost surely need a superpolynomial number of steps to find a global maximum. If \( p(n) \) is set to another value, a global maximum is found within a polynomially bounded number of steps with high probability. The objective function we discuss combines some ideas that can partly be identified in other functions that were already subject of theoretical investigations. The first basic idea is that the need for a large “jump”, i.e., the mutation of several bits simultaneously, causes difficulties for standard \( 1/n \)-mutations. This is the key property of \( \text{JUMP} \) used by Jansen and Wegener [7] to provide an example where crossover is helpful. The second observation is that analyzing the behavior of an EA may become substantially easier, if the function enforces that certain paths are followed almost surely. An example for that are ridge functions introduced (in a different context and for different reasons) by Quick, Rayward-Smith, and Smith [10].

For the sake of simplicity, we assume that \( n = 2^k > 32 \) implying that \( n/4 \geq 2 \log n \). Otherwise, appropriate rounding leads to substantially the same definitions and results. Throughout this paper by \( \log n \) we denote the logarithm to the base 2, i.e., \( \log_2 n \). In order to be able to present a well-structured and understandable definition we partition the search space \( \{0,1\}^n \) into five disjoint sets, namely

\[
A := \{x \in \{0,1\}^n \mid n/4 < ||x||_1 < 3n/4\}, \\
B := \{x \in \{0,1\}^n \mid ||x||_1 = n/4\}, \\
C := \{x \in \{0,1\}^n \mid \exists i \in \{0,1,\ldots,(n/4)−1\} : x = 1^i0^{n−i}\}
\]
\[ D := \left\{ x \in \{0,1\}^n \mid (|x|_1 = \log n) \wedge \left( \sum_{i=1}^{2\log n} x_i = 0 \right) \right\}, \] and

\[ E := \{0,1\}^n - (A \cup B \cup C \cup D), \]

where \(|x|_1\) denotes the number of ones in \(x\) and \(1^i0^{n-i}\) denotes the string with \(i\) consecutive ones followed by \(n-i\) consecutive zeros. Now, we can define the objective function that serves as our example.

**Definition 1.** The function \(\text{PATHTOJUMP} : \{0,1\}^n \to \mathbb{R}\) is defined by

\[
\text{PATHTOJUMP}(x) := \begin{cases} 
 n - |x|_1 & \text{if } x \in A, \\
 (3/4)n + \sum_{i=1}^{n/4} x_i & \text{if } x \in B, \\
 2n - i & \text{if } x \in C \text{ and } x = 1^i0^{n-i}, \\
 2n + 1 & \text{if } x \in D, \\
 \min\{|x|_1, n - |x|_1\} & \text{if } x \in E.
\end{cases}
\]

All strings in \(D\) are globally optimal. In the rest of the paper we use the notation \(X' \prec X''\) (\(X''\) has a higher rank than \(X'\)) to indicate that \(f(x') < f(x'')\) for all \(x' \in X'\) and \(x'' \in X''\). The set \(X''\) is globally better than \(X'\). By definition, we obtain

\[ E < A < B < C < D. \]

A typical run of the (1+1) EA whose mutation probability is not too large looks as follows. We start in \(A\) and, therefore, never reach \(E\). The (1+1) EA almost works like on simple linear functions and has a good chance to reach \(B\). Then it is unlikely to jump to \(C \cup D\). Hence, improvements are made in \(B\) until the best string \(1^{n/4}0^{3n/4}\) is reached there. Here we are close to \(C\) and follow the “path” given by the strings of \(C\) to the string \(0^n\). Finally, a jump to \(D\) is necessary. Here we need a mutation probability which is essentially larger than \(1/n\).

First, we prove that with mutation probability \(p(n) = 1/n\) with a probability close to 1 the (1+1) EA will not find a global optimum within a polynomial number of steps. In fact, we prove something stronger. We show that if the mutation probability \(p(n)\) substantially differs from \(\log n/n\), then the (1+1) EA needs a superpolynomial number of steps for optimizing \(\text{PATHTOJUMP}\) almost surely.

**Theorem 1.** Let \(\alpha(n) := (n-1)/\log n\). If \(\alpha(n) \to 0\) as \(n \to \infty\) or \(\alpha(n) \to \infty\) as \(n \to \infty\), the probability that the (1+1) EA with mutation probability \(p(n)\) needs a superpolynomial number of steps to find a global optimum of \(\text{PATHTOJUMP}\) converges to 1.

**Proof.** By Chernoff's bounds (Hagerup and Rüb [6]), the probability that the initial string belongs to \(A\) equals \(1 - e^{-\Omega(n)}\). Hence, we can assume that the (1+1) EA starts in \(A \cup B \cup C \cup D\) and never reaches strings in \(E\).

**Case 1:** \(\alpha(n) \to \infty\) as \(n \to \infty\). We only investigate the last step where a string \(x \in A \cup B \cup C\) is changed by mutation into a string \(y \in D\) and estimate the
probability of such a mutation. We know by definition that \( x \) contains at least \( n/4 \) zeros and in order to reach some \( y \in D \) it is necessary that at most \( \log n \) of these bits flip. The expected number of flipping bits among the chosen positions equals 
\[
(\lfloor n/4 \rfloor \cdot p(n)) = (1/4) \alpha(n) \log n.
\]
By Chernoff's bounds, the probability that at most \( \log n \) of these bits flip is bounded above by 
\[
e^{-\Omega(\alpha(n) \log n)} = n^{-\Omega(\alpha(n))} \leq n^{-\alpha(n)/2}
\]
for some \( c > 0 \). Hence, the expected running time of the \((1+1)\) EA is at least \( n^{\alpha(n)/2} \) which grows superpolynomially in this case. Also the probability that \( n^{\alpha(n)/2} \) steps are sufficient is bounded above by \( n^{\alpha(n)/2} \) which is superpolynomially small.

**Case 2:** \( \alpha(n) \to 0 \) as \( n \to \infty \) Again we estimate the probability to obtain \( y \in D \) by a mutation step from \( x \in A \cup B \cup C \). By case inspection, we conclude that the Hamming distance between \( x \) and \( y \) is at least \( \log n \). Here the assumption that \( y_1 + \cdots + y_{2\log n} = 0 \) for \( y \in D \) is essential. Hence, it is necessary that at least \( \log n \) bits flip simultaneously. The expected number of flipping bits equals \( \alpha(n) \log n \). By Chernoff's bounds, the probability that the number of successes is by a factor of \( \alpha(n) \log n \) larger than the expected value is bounded above by 
\[
e^{-\Omega(\alpha(n) \log n)} = n^{-\Omega(\alpha(n))} \leq n^{-\alpha(n)/2}
\]
for some \( c < \infty \). Remember that \( \alpha(n) \to 0 \) as \( n \to \infty \). Hence, the expected running time of the \((1+1)\) EA is at least \( n^{-\alpha(n)/2} \) which is superpolynomially increasing. Also, the probability that \( n^{-(c/2) \log \alpha(n)} \) steps are sufficient is bounded above by \( n^{-(c/2) \log \alpha(n)} \) which is superpolynomially small.

We remark that these lower bounds also hold for the \((\mu + \lambda)\) EA and the \((\mu, \lambda)\) EA. As long as \( \mu = e^{\alpha(n)} \), in particular, for populations of polynomial size, all initial strings belong to \( A \) with overwhelming probability and at least one string has to mutate in the way described in the two cases of the proof of Theorem 1.

In the following we investigate the case that \( \alpha(n) \) is a constant. In order to simplify some calculations we switch from \( \log n \) to \( \ln n \) and assume that \( p(n) = (\ln n)/n \). We analyze the random number of steps of the \((1+1)\) EA until an optimal string \( y \in D \) is found and then we look for that constant \( c \) leading to the best result.

**Theorem 2.** Let \( p(n) = (\ln n)/n \). The expected number of steps until the \((1+1)\) EA with mutation probability \( p(n) \) finds a global optimum of the function \( \text{PATHTOJUMP} \) is bounded above by \( O(n^{2+c \ln^{-1} n} + n^{c \log c - \log \ln 2}) \). For \( c = 1/(4 \ln 2) \leq 0.361 \) the expected running time is bounded by \( O(n^{2.361}) \).

**Proof.** In the following we work with the following partitions of \( A, B, C, \) and \( E \):

- \( A_i = \{ x \in A \mid ||x||_1 = i \}, n/4 < i < 3n/4 \)
- \( B_i = \{ x \in B \mid x_1 + \cdots + x_{n/4} = i \}, 0 \leq i \leq n/4 \)
- \( C_i = \{ 1^{i0^{n-i}} \}, 0 \leq i < n/4 \)
- \( E_i = \{ x \in E \mid ||x||_1 = i \text{ or } n - ||x||_1 = i \}, 0 \leq i < n/4 \)

Then we have

\[
E_0 < E_1 < \cdots < E_{(n/4) - 1} < A_{(3n/4) - 1} < \cdots < A_{(n/4) + 2} < A_{(n/4) + 1} < B_0
\]
Our aim is to derive lower bounds $\alpha_i$, $\beta_i$, $\gamma_i$, and $\epsilon_i$ for the probabilities that we reach within one step from $X \in A_i$, $B_i$, $C_i$, and $E_i$ resp. a string $y$ which is element of a set with a higher rank. These probabilities may be called success probabilities, since they measure the probability of increasing the fitness of the current string.

**Claim 1:** $\alpha_i = \Omega((\ln n)/n^c)$, $\epsilon_i = \Omega((\ln n)/n^c)$

**Proof.** If $x \in A_i$, we have a success if exactly one of the $i > n/4$ 1-bits and none of the 0-bits flip. Hence,

$$\alpha_i \geq \frac{c \ln n}{n} \left(1 - \frac{c \ln n}{n}\right)^{n-1} > \frac{1}{4} \frac{c \ln n}{n} \left(1 - \frac{c \ln n}{n}\right)^{n-1}$$

$$= \Omega \left( \ln n e^{-c \ln n} \right) = \Omega \left( \ln n \frac{n^c}{n^c} \right).$$

If $x \in E_i$, even at least $(3n/4)$ 1-bit mutations lead to a success.

**Claim 2:** $\beta_i = \Omega(((n/4) - i)^2(\ln^2 n)/n^{2+c})$, if $i < n/4$

**Proof.** If $x \in B_i$, $i < n/4$, $x$ contains $(n/4) - i$ 0-bits among the first $n/4$ positions and $(n/4) - i$ 1-bits among the last $3n/4$ positions. We have a success if exactly one of these 0-bits and one of these 1-bits flip. Hence,

$$\beta_i \geq \left(\frac{n}{4} - i\right)^2 \left(\frac{c \ln n}{n}\right)^2 \left(1 - \frac{c \ln n}{n}\right)^{n-2}$$

$$= \Omega \left( \left(\frac{n}{4} - i\right) \frac{\ln^2 n}{n^2} e^{-c \ln n} \right) = \Omega \left( \left(\frac{n}{4} - i\right)^2 \frac{\ln^2 n}{n^{2+c}} \right).$$

**Claim 3:** $\beta_{n/4} = \Omega((\ln n)/(n^{1+c}))$, $\gamma_i = \Omega((\ln n)/(n^{1+c}))$, if $i > 0$

**Proof.** If $x \in B_{n/4}$, $x = 1^{n/4}0^{3n/4}$ and, if $x \in C_i$, $x = i^0n^{-i}$. We have a success if exactly the last 1-bit flips. Hence

$$\frac{c \ln n}{n} \left(1 - \frac{c \ln n}{n}\right)^{n-1} = \Omega \left( \frac{\ln n}{n^{1+c}} \right)$$

is a lower bound for $\beta_{n/4}$ and $\gamma_i$.

**Claim 4:** $\gamma_0 = \Omega(n^\log c + \log \ln 2 - c)$

**Proof.** If $x \in C_0$, $x = 0^n$. Hence, the success event contains exactly all events where exactly $\log n$ of the last $n - 2 \log n$ bits flip. Hence

$$\gamma_0 = \left(\frac{n - 2 \log n}{\log n}\right) \cdot \left(\frac{c \ln n}{n}\right)^{\log n} \cdot \left(1 - \frac{c \ln n}{n}\right)^{n-\log n}$$

$$\geq \left(\frac{n - 2 \log n}{\log n}\right)^{\log n} \cdot \left(\frac{c \ln n}{n}\right)^{\log n} \cdot \left(1 - \frac{c \ln n}{n}\right)^n \cdot \left(1 - \frac{c \ln n}{n}\right)^{-\log n}$$
If the success probability of some event equals $q$, the expected waiting time for a success equals $q^{-1}$. In the worst case we start in $E_0$ and each success leads to a string of the next rank class. Hence, the expected running time $E(T(c))$ can be estimated above by

$$E(T(c)) \leq \sum_{i=0}^{(n/4)-1} \varepsilon_i^{-1} + \sum_{i=(n/4)+1}^{(3n/4)-1} \alpha_i^{-1} + \sum_{i=0}^{(n/4)-1} \beta_i^{-1} + \beta_{n/4} + \sum_{i=1}^{(n/4)-1} \gamma_i^{-1} + \gamma_0^{-1}$$

$$= O\left(n^{1+c} \ln^{-1} n\right) + O\left(n^{2+c} \left(\sum_{i=1}^{(n/4)-1} \frac{1}{i^2}\right) \ln^{-2} n\right) + O\left(n^{1+c} \ln^{-1} n\right)$$

$$+ O\left(n^{2+c} \ln^{-1} n\right) + O\left(n^{c-\log c-\log \ln 2}\right)$$

$$= O\left(n^{2+c} \ln^{-1} n + n^{c-\log c-\log \ln 2}\right),$$

since the series $\sum (1/i^2)$ is converging.

In order to choose the best value for $c$ we set

$$2 + c = c - \log c - \log \ln 2$$

which is equivalent to

$$\log c = -2 - \log \ln 2$$

or

$$c = \frac{1}{4\ln 2} \leq 0.361.$$

Theorem 2 shows that the appropriate mutation probability leads to an expected running time which is a polynomial of reasonable degree.

**Theorem 3.** Let $p(n) = (c \ln n)/n$. The probability that the $(1+1)$ EA with mutation probability $p(n)$ finds a global maximum of PATHTOJUMP within $O(n^{3+c} \ln^{-1} n + n^{1+c-\log c-\log \ln 2})$ steps ($O(n^{3.361})$ if $c = 1/(4\ln 2)$) is bounded below by $1 - e^{-\Omega(n)}$.

**Proof.** If the success probability of some event equals $q$, the probability of having no success within $\lceil q^{-1} n \rceil$ steps, equals

$$(1 - q)^{\lceil q^{-1} n \rceil} \leq (1 - q)^n = e^{-\Omega(n)}.$$
Our results show that we can gain a lot by choosing the appropriate mutation probability. The running time (the expected one and even a bound for the running time which holds with the overwhelming probability $1 - e^{-\Omega(n)}$) can decrease from superpolynomial to polynomial (with reasonable degree). But our results do not answer the question how to choose the right mutation probability. In black-box scenarios we do not know enough about the objective function to start any analysis and even in other scenarios the analysis is much too difficult to be carried out before starting the evolutionary algorithm.

4 The Dynamic (1+1) EA

We have seen that the choice of an appropriate mutation probability is advantageous and that we have in general no idea to compute or to estimate the appropriate mutation probability. Moreover, it can be even better to have different mutation probabilities in different phases of the algorithm. Hence, the algorithm has to “try” different mutation probabilities. Bäck [2] distinguishes a static parameter setting from three types of dynamic parameter settings. We choose the simplest one namely dynamic parameter control where the mutation probability only depends on the number of steps performed before. The more general schemes are adaptive parameter control which is guided by the success during the optimization process and self-adaptive parameter control where the change of the mutation probability is guided by an evolutionary algorithm.

Dynamic parameter control is easy to describe and implement. We choose $1/n$ as lower bound on the mutation probability since otherwise the expected number of flipping bits is less than one. We choose $1/2$ as upper bound since otherwise we flip on average more than half of the bits and do not search in the neighborhood of the current string. A phase consists of $\lceil \log n \rceil$ steps $t$, $1 \leq t \leq \lceil \log n \rceil$. The mutation probability in the $t$-th step of a phase equals $2^{-t}/n$.

Algorithm 2 (The dynamic (1+1) EA). The algorithm works as Algorithm 1 but the mutation probability in step $s$ where $s = r \lceil \log n \rceil + t$, $1 \leq t \leq \lceil \log n \rceil$, equals $p_t(n) = 2^{-t}/n$.

Theorem 4. The expected number of steps until the dynamic (1+1) EA finds a global optimum of the function $PATH\text{TO}\text{JUMP}$ is bounded above by $O(n^2 \log n)$.

Proof. We use an approach similar to that of the proof of Theorem 2. By $\alpha'_i$, $\beta'_i$, $\gamma'_i$, and $\varepsilon'_i$ we denote the corresponding success probabilities for a whole phase of $\lceil \log n \rceil$ steps of the dynamic (1+1) EA.

Claim 1: $\alpha'_i = \Omega(1)$, $\varepsilon'_i = \Omega(1)$

Proof. We only consider the step with mutation probability $1/n$. Hence (compare the proof of Theorem 2),

$$\alpha'_i \geq \frac{1}{n} \left( 1 - \frac{1}{n} \right)^{n-1} > \frac{1}{4} \left( 1 - \frac{1}{n} \right)^{n-1} = \Omega(1).$$

The result on $\varepsilon'_i$ follows in a similar way.
Claim 2: $\beta'_i = \Omega(((n/4) - i)^2/n^2)$, if $i < n/4$

Proof. As in the proof of Theorem 2, we have $((n/4) - i)^2$ pairs such that the event that exactly the bits of a pair flip is a success. Hence, using the mutation probability $1/n$

$$\beta'_i \geq \left(\frac{n}{4} - i\right)^2 \left(\frac{1}{n}\right)^2 \left(1 - \frac{1}{n}\right)^{n-2} = \Omega\left(\left(\frac{n}{4} - i\right)^2 \frac{1}{n^2}\right).$$

Claim 3: $\beta'_{n/4} = \Omega(1/n)$, $\gamma_i = \Omega(1/n)$, if $i > 0$

Proof. There is a 1-bit mutation leading to a success. The claim follows for the mutation probability $1/n$.

Claim 4: $\gamma'_0 = \Omega(n^{\log \log n - 1}) = \Omega(1/n^{1.53})$

Proof. The success event contains exactly all events where exactly $\log n$ of the last $n - 2 \log n$ bits flip. Hence, mutation probabilities of the order $(\log n)/n$ are most promising. In the proof of Theorem 2 we have proved the bound $\Omega(n^{\log c + \log \ln 2 - c})$ for the mutation probability $(c \ln n)/n$. This bound is maximal, if $c = 1/\ln 2 \leq 1.45$. Then $\log c = -\log \ln 2$ and the bound equals $\Omega(n^{-1.45})$. But the dynamic $(1+1)$ EA does not necessarily use exactly this mutation probability. We only now that we have for each interval $[c \ln n]/n; (2c \ln n)/n]$ one mutation probability falling into this interval. Here it is optimal to choose $c = 1$ leading to a lower bound of $\Omega(n^{\log \log n - 1}) = \Omega(n^{-1.53})$.

These claims lead to the following upper bound on the expected number $E(P)$ of phases

$$E(P) = O(n) + O\left(n^2 \sum_{i=1}^{(n/4)-1} \frac{1}{i^2}\right) + O\left(n^2\right) + O\left(n^{1.53}\right) = O\left(n^2\right).$$

The run time is bounded above by $\lceil \log n \rceil \cdot P$ which proves the claim. \qed

In the same way as we have obtained Theorem 3 from Theorem 2 we get the following result.

Theorem 5. The probability that the dynamic $(1+1)$ EA finds a global optimum of $\text{PATH}\text{JUMP}$ within $O(n^3 \log n)$ steps is bounded below by $1 - e^{-\Omega(n)}$.

The upper bounds of Theorem 4 and Theorem 5 are better than the corresponding bounds of Theorem 2 and Theorem 3. The dynamic $(1+1)$ EA wastes some steps by using inappropriate mutation probabilities. But this only leads to an additional factor $O(\log n)$. The dynamic $(1+1)$ EA does not always choose the optimal mutation probability (see the proof of Claim 4 in the proof of Theorem 4). This is not essential for $\text{PATH}\text{JUMP}$. But the dynamic $(1+1)$ EA tries different mutation probabilities while the "optimal" static mutation probability is a compromise between optimal mutation probabilities for the $\text{PATH}$-phase and...
the JUMP-phase. In order to rigorously prove that the dynamic (1+1) EA is better than each static (1+1) EA on PATHTOJUMP we need lower bounds for the static (1+1) EA. This implies bounds for the probability of jumping to strings whose rank is much larger than the current string. Such calculations are possible with our methods but we have not performed them yet.

5 Conclusions

We have presented the function PATHTOJUMP which has some interesting properties. The static (1+1) EA with mutation probabilities whose growth order is larger or smaller than \((\log n)/n\) needs superpolynomial time with overwhelming probability. But there are mutation probabilities such that the static (1+1) EA finds an optimum within an expected number of only \(O(n^{2.361})\) steps. Since in general there is no idea how to compute an optimal or good value for the mutation probability, we have presented a dynamic (1+1) EA with a simple dynamic control of the mutation probability. This variant of the (1+1) EA finds an optimum of PATHTOJUMP within an expected number of \(O(n^2 \log n)\) steps.

References

The Genetic Code-Like Transformations and Their Effect on Learning Functions

Hillol Kargupta

School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99163, USA.
hillol@eecs.wsu.edu

Abstract. The natural gene expression process evaluates the fitness of a DNA-string through a sequence of representation transformations. The genetic code defines one such transformation in this process. This paper shows that genetic code-like transformations introduce an interesting property in the representation of a genetic fitness function. It points out that such adaptive transformations can convert some functions with an exponentially large description in Fourier basis to one that is highly suitable for polynomial-size approximation. Such transformations can construct a Fourier representation with only a polynomial number of terms that are exponentially more significant than the rest when fitter chromosomes are given more copies through a redundant, equivalent representation. This is a very desirable property [2, 3] for efficient function-induction from data which is a fundamental problem in learning, data mining, and optimization.

1 Introduction

Gene expression evaluates the fitness of the DNA. It does so by first transforming the DNA sequence to the mRNA. The mRNA is a sequence of four different nucleotides namely, adenine (A), uracil (U), guanine (G), and cytosine (C). Next the mRNA sequence is transformed into protein, a sequence of amino acids. There exists a set of rules that defines the correspondence between nucleotide triplets (known as codons) and the amino acids in proteins. This is known as the genetic code (Table 1). Each codon is comprised of three adjacent nucleotides in an mRNA chain and it produces an amino acid. With a few exceptions the genetic code for most eukaryotic and prokaryotic organisms is the same.

This paper investigates the role of genetic code-like transformations (GCT) in genetic search. It presents a Fourier\(^1\) analysis of GCT in the binary sequence space and notes a quite interesting property of such representation transformations: It shows that there exists a class of GCT that can convert some functions\(^2\) of an exponential description in Fourier basis to a representation with only

\(^1\) The analysis is identical to that using Walsh basis [1]; however, I choose the term Fourier because of its historical [2, 3] use in function approximation literature.

\(^2\) Although the class of such functions is yet to be precisely characterized, this paper develops a general understanding.
Table 1. The universal genetic code.

<table>
<thead>
<tr>
<th>Protein feature</th>
<th>mRNA codons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>GCA GCC GCG GCU</td>
</tr>
<tr>
<td>Cysteine</td>
<td>UGC UGU</td>
</tr>
<tr>
<td>Aspartic acid</td>
<td>GAC GAU</td>
</tr>
<tr>
<td>Glutamic acid</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>UUC UUU</td>
</tr>
<tr>
<td>Glycine</td>
<td>GGA GGC GGG GGU</td>
</tr>
<tr>
<td>Histidine</td>
<td>CAC CAU</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>AUA AUC AUU</td>
</tr>
<tr>
<td>Lysine</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>Leucine</td>
<td>UUA UUG CUA CUC CUG CUU</td>
</tr>
<tr>
<td>Methionine</td>
<td>AUG</td>
</tr>
<tr>
<td>Asparagine</td>
<td>AAC AAU</td>
</tr>
<tr>
<td>Proline</td>
<td>CCA CCC CCG CCU</td>
</tr>
<tr>
<td>Glutamine</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>Arginine</td>
<td>AGA AGG CGA CGC CGG CGU</td>
</tr>
<tr>
<td>Serine</td>
<td>AGC AGU UCA UCC UCG UCU</td>
</tr>
<tr>
<td>Threonine</td>
<td>ACA ACC ACG ACU</td>
</tr>
<tr>
<td>Valine</td>
<td>GUA GUC GUG GUU</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>UGG</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>UAC UAU</td>
</tr>
<tr>
<td>STOP</td>
<td>UAA UAG UGA</td>
</tr>
</tbody>
</table>

da polynomial number of terms that are exponentially more significant than the rest when fitter proteins are given more copies through redundant and equivalent representation. Although the class of such functions is yet to be characterized, the growing interest in learning functions by computing its Fourier spectrum [2, 3] suggests further exploration of the gene expression.

Section 2 discusses related work and Section 3 presents a brief review of Fourier analysis. Section 4 analyzes the effect of GCT on the representation of the genetic fitness function. Section 5 concludes this paper.

2 Related Work

The importance of gene expression in genetic search was realized in the early days of the genetic algorithms (GAs). The “operon” model of the chromosome [4] and its influence on the development of diploid GAs [5] support this observation. The under-specification and over-specification decoding operator of the messy GA was viewed as a mechanism similar to gene signaling [6]. Kauffman [7] offered an interesting perspective of the natural evolution that realizes the importance for gene expression. An empirical study of genetic programming using artificial genetic code is presented in [8].

The “neutral network” theory [9] also considers sequence-to-structure mapping from the perspective of random graph construction. There exists extensive literature that investigates the evolution of the genetic code. An algebraic model...
of the evolution of the genetic code is presented in [10]. For additional work on
the different biological theories on the evolution of the genetic code see [11].

An alternate approach has been developed by Kargupta et al. [12, 13]. This
approach is mainly motivated by a perspective of the gene expression as a mecha­
nism to make genetic search more efficient and scalable. This approach views
gene expression as a sequence of representation transformations and explores
the role of such transformations in the computational complexity of detecting
a representation of the genetic fitness function. This paper further investigates
this line of thoughts and explores the role of the GCTs. It shows that such
transformations offer interesting properties in Fourier representation. Note that
the Fourier basis is functionally complete. Therefore, such transformations have
considerable implications on the complexity of learning functions from observed
data. Next we review the basics of Fourier representation.

3 Fourier Basis: A Brief Review

Fourier bases are orthogonal functions that can be used to represent any function.
Consider the function space over the set of all \( \ell \)-bit boolean feature vectors. The
Fourier basis set that spans this space is comprised of \( 2^\ell \) Fourier functions. A
Fourier basis function is defined as \( \psi_j(x) = (-1)^{\langle j, x \rangle} \). Where \( j, x \in \{0,1\}^\ell \) and
\( j = j_1, j_2, \cdots, j_\ell \), \( x = x_1, x_2, \cdots, x_\ell \) and \( j, x \in \{0,1\}^\ell \); \( x \cdot j \) denotes the inner
product of \( x \) and \( j \). \( \psi_j(x) \) can either be equal to 1 or \(-1\). The string \( j \) is called
a partition. The order of a partition \( j \) is the number of 1-s in \( j \). The \( \psi_j(x) \)
function depends on some \( x_i \) only when \( j_i = 1 \). Therefore a partition can also
be viewed as a representation of a certain subset of \( x_i \)-s; every unique partition
 corresponds to a unique subset of \( x_i \)-s. A function \( f : \{0,1\}^\ell \to \mathbb{R} \), that maps
an \( \ell \)-dimensional space of binary strings to a real-valued range, can be written
using the Fourier basis functions: \( f(x) = \sum_j w_j \psi_j(x) \). where \( w_j \) is the Fourier
Coefficient (FC) corresponding to the partition \( j \); \( w_j = \frac{1}{2^\ell} \sum_x f(x) \psi_j(x) \). The
Fourier coefficient \( w_j \) can be viewed as the relative contribution of the partition
\( j \) to the function value of \( f(x) \). Therefore, the absolute value of \( w_j \) can be used
as the “significance” of the corresponding partition \( j \). If the magnitude of some
\( w_j \) is very small compared to other coefficients then we may consider the \( j \)-th
partition to be insignificant and neglect its contribution.

4 Exploring GCT

The cardinalities of the alphabet sets of the mRNA and the protein are more
than two. However, understanding their underlying computation may require
abstraction. In this section we will do so by assuming that the protein and the
mRNA sequences are binary strings. Our objective is to explore the effect of the
GCT in the binary domain using Fourier analysis. Although strings are binary
we will continue to use the terms mRNA, protein, and genetic code accordingly
for maintaining the link between biology and the current analysis.

4.1 The Notion of GCT

Let us use \( r \) and \( p \) to represent the mRNA and the protein sequences respectively.
Let \( \ell_r \) and \( \ell_p \) be their respective lengths. Just like the natural translation, our
Protein feature | mRNA codon
---|---
1 | 100
1 | 000
1 | 001
1 | 010
0 | 111
0 | 101
0 | 110
0 | 011

Protein feature | mRNA codon
---|---
0 | 100
0 | 000
0 | 001
0 | 010
0 | 111
0 | 101
0 | 110
1 | 011

Fig. 1. (Left) Code A: A GCT for binary representation. Single bit in the protein space maps to 3-bit codons in the mRNA space. (Right) Code B.

artificial translation maps the mRNA to the corresponding protein sequence using the genetic code. The mapping in Translation will be denoted by $\eta_c$ where the subscript $c$ denotes the number of mRNA features that define a codon. If three features are used like natural codons, $c = 3$; $\eta_c$ can be defined as $\eta_c : R^c_r \rightarrow P^c_p$. $R^c_r$ and $P^c_p$ denote the $\ell_r$ and $\ell_p$ dimensional space of all mRNAs and proteins respectively. Note that $\ell_r = c\ell_p$. In our case, $R = P = \{0, 1\}$.

Consider the genetic codes presented in Figure 1. Note that the genetic code may be redundant. In other words, a unique protein feature value may be produced by several mRNA codons. This is also true for natural genetic code (Table 1). As a result, there exist many equivalent mRNA sequences that produce the same protein sequence. All these mRNA sequences have the same genetic fitness since they all map to the same protein sequence. So we can view the space of mRNAs grouped into different equivalence classes. We shall call this characteristic Translation Introduced Equivalence (TIE) and these groups of equivalent mRNAs will be called the TIE classes. Let $R_p$ be the TIE class for the protein sequence $p$. We can also define $R_p$ in the following manner: $R_p = \{r_j|r_j \eta_j^c \rightarrow p\}$. The cardinality of the set $R_p$ depends on the genetic code and the protein sequence $p$. Let $a_0$ and $a_1$ be the total number of codons that map to a protein feature value of 0 and 1 respectively. Let $\ell_p,0$ and $\ell_p,1$ be the number of 0-s and 1-s in $p$ respectively. Then the cardinality of the TIE class is $|R_p| = a_0^{\ell_p,0}a_1^{\ell_p,1}$.

Since one feature in the protein sequence maps to $c$ mRNA features, partitions in the mRNA and the protein spaces can be associated with each other. If $j$ and $j'$ are partitions in the mRNA and the protein spaces respectively then $j'$ is the reflection of $j$ in the protein space when $j'_i = 1$ if and only if $j$ takes a value of 1 at the location(s) corresponding to at least one of the mRNA features associated with $j'_i$. For example, the reflection of the partitions 101000 and 100110 under a genetic code of codon size three are 10 and 11 respectively. Note that different mRNA partitions may have the same reflection in the protein space. If $q$ is the number of ones in $j'$ then it is the reflection of $(2^c - 1)^q$ different partitions
in the mRNA space. The number of 1-s in \( j' \) will be called the absolute order of partition \( j \).

Once the protein sequence is constructed from the mRNA sequence, the protein folds into a three dimensional structure and its shape determines its fitness. Let us use \( f : P^t \rightarrow \mathbb{R}^+ \) for denoting this fitness function that maps the protein sequence to a non-negative real-valued range. Since the protein sequences are produced from the mRNA sequences, we can also define the fitness over the domain of mRNA sequences. Let \( \phi : R^t \rightarrow \mathbb{R}^+ \) be this fitness function defined over the mRNA domain. \( \phi(r) = f(p) = f(\eta_c(r)) \). Therefore, \( \phi(r) \) can be viewed as a different representation of the genetic fitness function \( f(p) \).

Let us now study the representations of \( f(p) \) and \( \phi(r) \). We will be particularly interested in the effect of the representation transformation \( \eta_c \) on the complexity of describing the function. In other words, we would like to know if \( \phi(r) \) has a more concise description compared to that of \( f(p) \). The results will be often illustrated using the Trap and the Needle-in-a-haystack (NH) functions. An \( \ell \)-bit deceptive trap function is defined as, \( f(x) = \ell \) if \( \text{ones}(x) = \ell \); \( f(x) = \ell - 1 \)-\text{ones}(x) otherwise; where \( \text{ones}(x) \) is the number of 1-s in the string \( x \). The NH function is defined as follows: \( f(x, x_{\text{opt}}) = \ell \) if \( x = x_{\text{opt}} \); \( f(x) = 0 \) otherwise.

### 4.2 Individual Fourier Coefficients

The \( j \)-th Fourier coefficient in the mRNA space can be computed as,

\[
w_j = \frac{1}{2^c \ell_\nu} \sum_p f(p) \sum_{r_1 \in R_p} \psi_j(r_1)
\]

The magnitude of the second summation in the above expression may take a value in between 0 and \( a_{l_0} a_{l_1} \) (cardinality of \( R_p \)) depending upon the nature of the set \( R_p \). This imposes a scaling factor to the contribution of every unique protein sequence to the \( j \)-th Fourier coefficient. Let us represent \( j \) using a collection of partitions \( \{j_0, j_1, \ldots, j_g\} \) where \( j_0 \) represents the null partition with all 0-s and every \( j_i \) represents a sub-partition of the 1-contributing positions of \( j \) that contains only those features that belong to the same protein feature. Note that the reflection of any \( j_i \neq 0 \) in the protein space has only one 1.

If \( p_\alpha \) is the protein feature value in a given \( p \) corresponding to the reflection of the \( \alpha \)-partition in the mRNA space then let \( R_{p_\alpha} \) be the set of all mRNA codons that maps to \( p_\alpha \). Let \( R_{j',p} \) be the Cartesian product of \( R_{p_\alpha} \)-s for \( \alpha = 1, 2, \ldots q \). Every member of \( R_{j',p} \) has \( c q \) mRNA features. For example consider \( p = 110 \) and \( j = 110000010 \). So \( j_0 = 000000000, j_1 = 110000000 \) and \( j_2 = 000000010 \). In case of code A, \( R_{p_1} = \{100, 000, 001, 010\} \) and \( R_{p_2} = \{111, 101, 110, 011\} \). Therefore \( R_{101,110} = R_{p_1} \times R_{p_2} \). Let \( q_{p,j',0} \) and \( q_{p,j',1} \) be the number of 0-s and 1-s in \( p \) that are covered by the fixed bits of \( j' \), the reflection of \( j \) in the protein space. Now we can write from Equation 1,

\[
w_j = \frac{1}{2^c \ell_\nu} \sum_p f(p) a_{l_0} a_{l_1} \prod_{\alpha=0,1,\ldots,q} \sum_{r_{j,p} \in R_{p_\alpha}} \psi_{j_\alpha}(r_{j,p})
\]
Note that the basis function \( \psi_{\alpha}(r_{j,p}) \) is well defined for any \( r_{j,p} \in R_{j',p} \) for any \( \alpha \) since the feature values of \( r_{j,p} \) are defined for every defining location of the partition \( j_\alpha \). Let \( e_{j_\alpha,p} \) and \( o_{j_\alpha,p} \) be the number of members in \( R_{p_\alpha} \) that have an even and odd number of ones respectively over the partition \( j_\alpha,p \). For example, if \( \alpha = 110000 \) and \( p = 10 \) then \( e_{110000,10} = 0 \) and \( o_{110000,10} = 1 \) for code B shown in Table 1. Now using Equation 2 we can write,

\[
w_j = \frac{1}{2 \ell_p} \sum_{p} f(p) a_0^{f_p,0 - q_p,j,0} a_1^{f_p,1 - q_p,j,1} \kappa \prod_{\alpha=0,1,\cdots,q} |e_{j_\alpha,p} - o_{j_\alpha,p}| \quad (3)
\]

Where \( \kappa \in \{-1, 1\} \) and \( |e_{j_\alpha,p} - o_{j_\alpha,p}| \) denotes the magnitude of \( (e_{j_\alpha,p} - o_{j_\alpha,p}) \) for all \( \alpha \neq 0 \). The value of \( |e_{j_\alpha,p} - o_{j_\alpha,p}| \) can be determined directly from the genetic code. By definition, for the null partition \( \alpha = 0 \), we set \( |e_{j_0,p} - o_{j_0,p}| = 1 \) as before, this is done to take care of the case when \( j \) is comprised of only 0-s.

### 4.3 Energy of the Fourier Spectrum

The energy of the Fourier spectrum can be defined as, \( E = \sum_j w_j^2 \). Let us now study the change in the overall energy of the spectrum due to the GCT. Using Equation 1 and noting that \( V_j(x) = \sum_{j} \psi_j(x) \) we can write [14],

\[
\sum_j w_j^2 = \frac{1}{2 \ell_p} \sum_{p} f^2(p) a_0^{f_p,0} a_1^{f_p,1} \quad (4)
\]

Let us now specialize this result for code A. For this code, \( a_0 = a_1 = 4 \) and \( c = 3 \). Substituting these values in Equation 4 we get,

\[
E_R = \frac{1}{2 \ell_p} \sum_{p} f^2(p) = E_P
\]

Where \( E_R \) and \( E_P \) are the energies of the mRNA and the protein spaces. The overall energy remains invariant under the transformation code A. Code B however changes the overall energy. For example, the overall energy of the Fourier representation of the Trap function with four variables using code B is 2.2778\( E_P \).

Distribution of the energy among the coefficients of different order is a very interesting property of a representation. Let us define the order-\( k \) energy, \( E^{(k)} = \sum_{j} \psi_{j_\alpha}(j) = k w_j^2 \). We can compute this for both the protein and the mRNA space. Note that the an order-\( k \) partition in the mRNA space may correspond to a lower order partition in the protein space since multiple mRNA features are associated with the same protein feature. A careful study of the effect of the representation transformations on the order-\( k \) energy in the mRNA space may require understanding the properties of the coefficients in the mRNA space that correspond to exactly \( k \) features in the protein space. Let us call it the \textit{absolute order-\( k \) energy}, defined as \( E^{(k)} = \sum_{j} \psi_{j_\alpha}(j) = k w_j^2 \); as defined earlier, \( j' \) is the reflection of \( j \) in the protein space. Just like the association between the partitions in the protein and the mRNA spaces through the concept of reflection, the distribution of energies in these two representations can be linked though
the concept of absolute order-\(k\) energy. Using Equation 2 we can derive [14] the overall energy of the representation for code B.

\[
E_B = \sum_j w_j^2 \approx \frac{1}{2c\ell_p} \sum_p \frac{f^2(p)}{a_{0\ell_p,0}} g(\ell_p, \ell_{p,0}, \ell_p) \tag{5}
\]

Where,

\[
g(\ell_p, \ell_{p,0}, k) = \frac{1}{2c\ell_p} \sum_{q=0}^{k} \min(q, \ell_{p,0}) \sum_{q_p,j,0} \left(\frac{\ell_p - \ell_{p,0}}{q - q_p,j,0}\right) \left(2^q - 1\right)^{a_{0\ell_p,0} - 2q_{p,j,0}}
\]

Where \(0 \leq k \leq \ell_p\). We would like to study the convergence of \(g(\ell_p, \ell_{p,0}, k)\) to 1 as \(k\) increases from 0 through \(\ell_p\). Note that \(g(\ell_p, \ell_{p,0}, k)\) is also a function of \(\ell_{p,0}\), the number of 0-s in a sequence \(p\). Since we are dealing with boolean sequences, \(\ell_{p,0}\) is sufficient to define any particular \(p\).

Note that the sequence with high genetic fitness contribute significantly to the overall energy \(E_R\) since \(f^2(p)\) will be large for them. Moreover the effect of \(f^2(p)\) on the energy gets scaled up by the factor \(a_{0\ell_p,0}\). This essentially means that fitter proteins with large number of 0-s will mainly contribute to \(E_R\). Now note that for proteins with large number of 0-s (i.e. relatively large \(\ell_{p,0}\)) the function \(g(\ell_p, \ell_{p,0}, k)\) approaches 1 very fast (Figure 3(Left)). In other words, the main portion of the overall energy comes from the highly fit proteins that have more number of equivalent mRNA representations (implied by large value of \(\ell_{p,0}\) and bias of code B towards the protein feature 0).

Figure 2 shows the effect of Code A on the absolute-order-energy distribution of the function NH (4-bit in protein representation). It shows that code A does not have any effect on the distribution. Figure 3(Right) shows the same for code...
Fig. 3. (Left) $g(300, \ell_{p,0}, k)$ with respect to increasing $k$ for two boundary cases $\ell_{p,0} = 0$, $\ell_{p,0} = 300$, and the intermediate case $\ell_{p,0} = 150$ using code B. (Right) Distribution of the absolute order energy using Code B for function NH.

B which shows a fast decay in the energy for the same problem. Now let us consider the Trap functions. Note that in this case although the sequence with all 1-s has the highest function value, there are other sequences that have non-zero function value. The sequences with more number of 0-s have relatively high fitness values. This also matches with the bias of the genetic code B. Therefore we should expect a good approximation using the low order coefficients. Figure 4(Left) shows the distribution of absolute order-$k$ energy using the code B and no transformation for a Trap function with $\ell_p = 4$. Note that the absolute order-$k$ energy decreases exponentially [14] for code B. Figure 4(Right) shows the distribution of order-$k$ energy using code B for the Trap function. Note that this is the order-$k$ energy of the 12-bit mRNA representation.

It appears that an approximate match between the bias of the genetic code and the representation of the fitter proteins may not be difficult to achieve. Fitter proteins have larger values of $f^2(p)$. If we assign more number of codons to the most frequent feature value (either 1 or 0 in case of binary strings) used in the fitter proteins, then the corresponding scaling factor ($a_0^{\ell_{p,0}}$ in case of code B) will also be large. For these proteins $g(\ell_p, \ell_{p,0}, k)$ also approaches 1 very fast with respect to $k$. In case of code B, the larger the value of $\ell_{p,0}$ in a protein, the higher the rate of convergence and the larger the scaling factor. On the other hand, if the proteins with frequent feature values that have less number of codons assigned (1 in case of code B) will have a slower convergence rate for $g(\ell_p, \ell_{p,0}, k)$ and smaller scaling factor. In case of code B it will be proteins with smaller values for $\ell_{p,0}$ (i.e. strings with more number of 1-s). Although the convergence rate will be slow, if the fitnesses of these proteins are relatively low then their contribution to the overall energy will be low since the scaling factor will be small for them. Note that if the fitness value is relatively small compared
Fig. 4. (Left) Distribution of the absolute order energy using Codes A and B for the Trap function. (Right) Distribution of the order-k energy using Codes A and B for the Trap function. Note that 4-bit protein space maps to 12-bit mRNA representation.

to the scaling factor, the latter will play a more significant role. For binary representation the issue is assigning a codon distribution among two possible protein features 0 and 1. For representations with higher cardinality the code introduces richer transformations that are yet to be explored.

5 Conclusions

This paper offers some intriguing properties of GCT-s that may be extremely useful for learning functions (e.g. Fig. 5) from data. It shows that there exists a class of GCT-s that can construct a Fourier representation of the fitness function where the low order coefficients are exponentially more significant than the higher order coefficients. This is a very critical property that allows a polynomial complexity approximation of an exponentially long function representation. The implication of this paper on the field of evolutionary computation is important. A technique for efficient and scalable induction of function representation will be useful in almost every application of evolutionary algorithms such as evolving programs, learning classifiers, data mining, and optimization. This work suggests further exploration of the computational role of gene expression. That may ultimately lead us toward unveiling the true power of genetic search.

Acknowledgments

The author acknowledges support from US NSF Grant IIS-9803660 and thanks Alden Wright, Robert Heckendorn, and Dirk Thierens for valuable comments.

References

Fig. 5. Approximation of NH using only up to order-2 coefficients of the original and the transformed representations (Code B). The latter is more accurate because of the exponential decay property introduced by code B.

Abstract. When considering continuous spaces EA, a convenient tool to model these algorithms is perturbation theory. In this paper we present preliminary results, derived from Freidlin-Wentzell theory, related to the convergence of a simple EA model. The main result of this paper yields a bound on sojourn times of the Markov process in subsets centered around the maxima of the fitness function. Exploitation of this result opens the way to convergence speed bounds with respect to some statistical measures on the fitness function (likely related to irregularity).

1 Introduction

While strong results have been obtained in a recent past concerning the convergence behaviour of particular applications of Evolutionary Algorithms (EAs), especially in the discrete case, no generic theory has been proposed to deal at once with a wider variety of frameworks.

In this paper, we will show how classical stochastic analysis tools can be used in order to build a model of EA, first step toward a theoretical toolkit that could apply to a wide range of evolutionary problems.

The main mathematical results we will rely on are inspired by Freidlin and Wentzell’s [1] fundamental work about stochastic perturbations of dynamic systems, a necessary tool to obtain global time-related equivalents on Markov processes. And EA modeling principles are based on Raphaël Cerf’s [8] and Olivier François [10] works on GA convergence. The principles of Olivier François’ MOSES model, using only selection or mutation in a discrete context, have been partly reused in this paper to build a continuous-space model of EA.

Considering a very simple model of EA, presented in section 2, with no crossover and a very basic selection scheme, enables us to focus on EAs’ global behaviour along time, with affordable computational complexity.

With stronger hypotheses on exponential moments in the main theorem of Freidlin and Wentzell’s perturbation theory, we prove a global bound on measures (theorem 2, section 3), in place of an asymptotic bound.

In section 4 this stronger version of the theorem is used to get an exponential bound on probabilities for populations to lie in well-chosen sets, yielding an existence theorem (theorem 3) on time-exponential bounds for sojourn times.
A Markov Model for Evolutionary Algorithms

2.1 Optimization model

In the following, we will consider stochastic processes, or random functions, which are time-indexed families of random variables on a probabilistic space \((\Omega, \mathcal{F}, \mathbb{P})\), and more precisely, Markov processes. We aim at finding the maximum of a function \(h: \mathbb{R}^n \rightarrow \mathbb{R}^+\). We suppose that the problem is consistent:

- \(h\) does not reach its upper bound at infinity, i.e. there does not exist a sequence \(x_i\) of points in \(\mathbb{R}^n\) such that \(\lim_{i \to \infty} |x_i| = \infty\) and \(\lim_{i \to \infty} h(x_i) = \sup_{\mathbb{R}^n} h\)
- the set of maxima \(\text{Argmax}(h) = \{x \in \mathbb{R}^n | h(x) = \sup_{\mathbb{R}^n} h\}\) is finite

An evolutionary algorithm can be defined as an operator acting on populations, i.e. subsets of \(X = \mathbb{R}^n\). Sequences of populations \(\{\xi_t\}\), indexed by the set \(T\) of simulation times, a subset of \(\mathbb{N}\), are produced by iterating this operator. Considering only populations of a fixed size \(d\), an EA will be described as an operator \(R\) from \(X^d\) into itself. A new population \(\xi_{t+1}\) is computed from the current one \(\xi_t\) at time \(t\) by applying \(R\), so that \(\xi_{t+1} = R(\xi_t)\). This algorithm will converge, from an initial population \(\xi_o \in X^d\), if \(\xi_t\) converges towards the maximal set of \(h\) when \(t\) goes to infinity, that is: \(\lim_{t \to \infty} \max \{d(x, \text{Argmax}(h)) | x \in \xi_t\} = 0\).

We can even suppose that the operator itself depends on \(t\), we thus write \(\xi_{t+1} = R_t(\xi_t)\). \(R_t\) is usually a random operator.

2.2 Markov Model

The collection of populations over time is a discrete-time stochastic process \(\xi_t \in X, t \in T\). At this point, assumptions on the random operator \(R_t\) must be made. We will define a reproduction operator \(R_t\) by an elementary transition probability \(<(t, x, r) = \mathbb{P}\{\xi_{t+1} \in r | \xi_t = x\}\), the probability for the population at time \(t + 1\) to be in a set of possible populations \(r\) if the population at time \(t\) was \(x \in X\).

The existence of such a transition probability makes \(\{\xi_t\}\) a Markov process. We deduce from \(\Phi\) a global transition probability \(P(s, x, t, \Gamma) = \mathbb{P}\{\xi_t \in \Gamma | \xi_s = x\}\) to get from \(x \in X\) at time \(s \in T\) into \(\Gamma\) at time \(t \in T\):

\[
P(s, x, s, \Gamma) = \mathbb{I}_\Gamma(x) \quad \text{(that is 1 if } x \in \Gamma, \text{ 0 otherwise)}
\]

\[
\forall t \in T, x \in X, P(s, x, t + 1, \Gamma) = \int_X \Phi(t, y, \Gamma)P(s, x, t, dy)
\]

We will not actually work on these transition probabilities, but rather on the associated densities \(\phi(t, x, \Gamma) = \int_r \phi(t, x, u) du, \text{ and } P(s, x, t, \Gamma) = \int_r \phi(s, x, t, u) du\).

Here are some basic properties (\(\delta\) is the Dirac distribution centered at 0):

\[
p(s, x, s, y) = \delta(y - x), \quad p(s, x, s + 1, y) = \phi(s, x, y)
\]

and \(p(s, x, t + 1, y) = \int_X p(s, x, t, u)\phi(t, u, y) du\)
A operator $D_t$ can be introduced, defined for a measurable function $f$ from a vector space $E$ into $X$ by, $\forall t \in T, x \in X$:

$$D_t f(x) = \int_X f(u) \phi(t, u, x) \, du$$

(1)

If $f$ represents some criterion on the population at time $t$, then $D_t f$ is an estimation of this criterion at time $t + 1$. We look forward in time, using the known values of $f$ on the current population together with the information on possible offsprings to compute this estimation. We can rewrite the above equations using $D_t$:

$$p(s, x, s, y) = \delta(y - x) \text{ and } p(s, x, t + 1, y) = D(p(s, x, t, \bullet))(y)$$

Moreover, since the populations are meant to be only sets, the order of coordinates on $X$ is irrelevant: any permutation must leave $\phi$ unchanged. If we define, $\forall \sigma \in S_d$ (the set of all permutations on $d$ elements), $x_\sigma = (x_\sigma(1), x_\sigma(2), \ldots, x_\sigma(d))$, we can write this condition as: $\forall (x, u) \in X^2, \forall i \in [1, d]$

$$\phi(t, x, u) = \phi(t, x, u_\sigma) \text{ and } \phi(t, x, u) = \phi(t, x_\sigma, u)$$

2.3 Evolutionary Model

To shorten the proofs, we have restricted the model to individuals taken in $\mathbb{R}$, so that $X = \mathbb{R}^d$. This does not affect the model's generality. In the first approach described here, we will only use mutation to generate offsprings, so that one individual has one, and only one, offspring. This condition is very restrictive, and in doing this we are stepping back from the general definition of evolutionary algorithms, involving selection (individuals can have none or many offsprings) and crossover (an individual is born from at least two parents).

The initial population is generated with respect to a random isotropic law $Q$, that is in terms of distribution: $Q(\Gamma) = \int_\Gamma q(u)du$ where $q$ is isotropic. Defining $p(t, x), t \in T, x \in X$, by $\mathbb{P}(\xi_t \in \Gamma) = \int_\Gamma p(t, x) \, dx$, we get:

$$p(0, x) = q(x) \text{ and } p(t + 1, x) = \int_X p(t, u) \phi(t, u, x) \, du$$

The mutation process can be split into two phases: the strength of the mutation of a particular individual, and the "shape" of the mutation itself. If we do not discriminate between individuals, we should fix once for all the shape at a given time. Keeping in mind we should use isotropic functions since populations are unordered sets, we can mutate individuals following a density $g_t$, with $g_t(-u) = g_t(u)$. Let $m(x)$ be a "mutation vector" in $\mathbb{R}^d$, where $m_i(x)$ is our mutation decision on the i-th individual in the population. $m_i(x) = 0$ stands for "no mutation at all", and the probability of mutations grows with $m_i(x)$. We can rewrite $\phi$ as:
\[ \phi(t, x, y) = \prod_{m_i(x)=0} \delta(y_i - x_i) \prod_{m_i(x) \neq 0} g_t(y_i - x_i) \]

or

\[ \phi(t, x, y) = \prod_{i=1}^{d} \frac{1}{m_i(x)} g_t \left( \frac{y_i - x_i}{m_i(x)} \right) \]

A very simple example of decision function can be defined as follows: let \( \alpha, 0 \leq \alpha \leq 1 \) be a threshold value,

\[ m_i(x) = \begin{cases} 0 & \text{if } h(x_i) - a(x) \geq \alpha(b(x) - a(x)) \\ 1 & \text{otherwise} \end{cases} \]

where \( a(x) \) is the lowest value of the fitness function \( h \) on the population \( x \), and \( b(x) \) its highest value. That means the individuals with fitness lower than the threshold \( \alpha \) mutate.

We could also use a variation of elitist selection, by letting \( m_i(x) = 0 \) if \( x_i \) is among the \( \alpha.d \) better individuals in population \( x \), and 1 otherwise.

But there is no need to restrict \( m_i \) to binary values. For instance, \( m_i(x) = \frac{h(x_i)}{\sum_{j=1}^{d} h(x_j)} \) could be used, which represents still another adaptation of the selection concept to our model.

We will need in the following a generic definition of this model:

**Definition 1: EA process**

Using the conventions stated above, we will call "EA process" with parameters \([q, \phi]\) a Markov process \( \{X_t\} \), taking its values in \( X = \mathbb{R}^d \) and its times in \( \mathbb{R}_+ \), with the properties:

\[ \mathbb{P}(X_t \in \Gamma) = \int_{\Gamma} p(t, x) \, dx \quad (2) \]

\[ p(0, x) = q(x) \text{ and } p(t + 1, x) = \int_X p(t, u).\phi(t, u, x) \, du \quad (3) \]

We will call \( q \) the initialization function of the EA, \( \phi \) its transition function, and \( p \) its density.

### 3 Perturbation theory

#### 3.1 Exponential moments

Freidlin and Wentzell’s perturbation theory makes an extensive use of exponential moments of the measures of interest. The measures \( \mu_t \), where \( \mu_t(\Gamma) = \mathbb{P}(\xi_t \in \Gamma) = \int_{\Gamma} p(t, x) \, dy \), are examined here. Their exponential moments are:

\[ H^t(\alpha) = \ln \int_X e^{\langle \alpha, x \rangle} p(t, x) dx \quad (4) \]
As in section 2.2, we can introduce an operator $G_t$, defined for a measurable function $f$ from a vector space $E$ into $X$ by, $\forall t \in T, x \in X$:

$$G_t f(x) = \int_X f(u) \phi(t, x, u) \, du$$

(the variables are switched in comparison to the ones of the $D_t$ definition). $G_t$ does the opposite of $D_t$: instead of looking forward in time, trying to estimate the future value of some criterion $f$, it looks backward, estimating the value of this criterion on the previous population. We then write:

$$H^{t+1}(\alpha) = \ln \int_X p(t, u) G_t \left( e^{(\alpha, *)} \right) (u) \, du$$

Iterating this result, and letting $\Gamma_t f(x) = \ln G_t \left( e^f \right) (x)$, we get:

$$H^t(\alpha) = \ln \int_X q(x) e^{(\Pi_{s=1}^t \Gamma_s) (e^{(\alpha, *)})} (x) \, dx$$

### 3.2 Moments properties

The properties we are interested in apply to the iterates of the $\Gamma$ operator on linear functions. Let us define a function $\gamma_t$ from $\mathbb{R}$ into $\mathbb{R}^+$ by:

$$\gamma_t(s) = \ln \int_{\mathbb{R}^+} e^{s \gamma_t(u)} \, du$$

$\gamma_t$ is actually the exponential moment associated with $g_t$, the mutation law, and summarizes all the useful information on it from the point of view of exponential moments. We get:

$$H^{t+1}(\alpha) = \ln \int_X p(t, u) e^{(\alpha, u)} e^{\sum_{i=1}^{d} \gamma_t(\alpha; m_i(u))} \, du$$

where $d$ is the population's size, $p$ its density at time $t$, and $m$ the mutation decision.

Freidlin and Wentzell’s perturbation theorem gives only asymptotic informations on a probability distribution. However, under somewhat more restrictive conditions and minor changes in the original proof, we have established an upper bound on the distributions, that applies globally instead of asymptotically (see appendix B for proofs).

Let us recall the Legendre’s transform definition:

**Definition 2: Legendre’s transform**

Given $H : X \to \mathbb{R}$ convex, the Legendre’s transform of $H$ is, for $x \in X$:

$$\mathcal{L}H(x) = \sup_{u \in X} \left( \langle u, x \rangle - H(u) \right)$$

**Theorem 1: Uniform bound**

Let $\{\mu^h\}, h \in \mathbb{R}^+$ be a family of probability measures on $X$. For $\alpha \in X$, we define $H^h(\alpha) = \ln \int_X e^{(\alpha, x)} \, d\mu(x)$. $H^h$ is convex. Let us make the following hypotheses:
- for any fixed $\alpha$, $h \mapsto H^h(\alpha)$ is a non-increasing function
- there exists some non-increasing function $\lambda : \mathbb{R}^*_+ \to \mathbb{R}^*_+$ such that:
  - $\lim_{h \to 0} \lambda(h) = \infty$
  - $\forall \alpha \in X, H(\alpha) = \lim_{h \to 0} \frac{1}{\lambda(h)} H^h(\lambda(h)\alpha)$ exists and the limit $H$ is $C^1$

Then for $r \in \mathbb{R}, v \in X$, we have: $\mu^h(\{x \in X, \langle v, x \rangle \geq r\}) \leq e^{-\lambda(h) \inf \{\langle v, x \rangle \geq r\} } \mathcal{L}H(x)$

Note 1: To preserve Freidlin and Wentzell's notations, we have used here a parameter $h$ to index the family of measures, and examined their behavior as $h \to 0$. However, to be consistent with the indexing of populations by $t$ in EAs, results will be stated using $t \to \infty$ in the following.

Note 2: In the following we will respectively speak, for $\mathcal{L}H$ and $\lambda$, of an “action functional” and a normalization factor for the family of measures $\{\mu^h\}$.

We are specially interested in the case $v = \pm e_i$, where $e_i$ is the base vector with null components except for the coordinate $i$: we then have $x_i \leq r$ or $x_i \geq r$.

Let us suppose the conditions of the theorem are fulfilled. We will build hypercubic “balls”, by excluding $2d$ half-spaces. These half-spaces being defined by some relation like $x_i > r$, a permutation on the coordinates doesn’t change anything, and we will get an exponential upper bound on the probability to be outside this ball. Taking into account sections 3.1 and 3.2’s results on the $G$ operator (equation 5), we get:

**Theorem 2: Probability asymptotics**

Let us examine an EA process with isotropic initialization function $q$, and with transition function $\phi$, with:

$$\phi(t, x, y) = \prod_{i=1}^{d} \frac{1}{m_i(x)} g_t \left( \frac{y_i - x_i}{m_i(x)} \right)$$

If there exists:

- an non-decreasing function $\lambda : \mathbb{R}^*_+ \to \mathbb{R}^*_+$, with $\lim_{h \to 0} \lambda(h) = \infty$,
- a function $\alpha \mapsto \overline{H}(\alpha)$ such that $\forall \alpha, \frac{1}{\lambda(t)} H^t(\lambda(t)\alpha) \leq \overline{H}(\alpha)$,

then $\forall \alpha \in X, H(\alpha) = \lim_{h \to 0} \frac{1}{\lambda(h)} H^h(\lambda(h)\alpha)$ exists. And if we let

$A = \{x \in X | \exists i, |x_i - x_{o_i}| \geq r\}$, we get: $P(t, A) \leq 2.d.e^{-\lambda(t) \inf_A \mathcal{L}H(x)}$

4 Convergence speed

4.1 Particular values of $g_t$

In order to use theorem 2, let us assume that our mutation kernel $g_t$ has the following form: $g_t(u) = k_t g(k_t u)$ where $g = \frac{1}{2} \mathbb{I}_{[-1,1]}$. This yields: $\gamma_t = \gamma \left( \frac{s}{k_t} \right)$, where: $\gamma(s) = \ln \frac{1}{2} \int_{-1}^{1} e^{su} du = \ln \sinh s - \ln s$.

Let us recall that $m(u)$ is the mutation decision vector, where $m_i(u)$ stands for the mutation rate of the $i$-th individual in population $u$. As $0 \leq \gamma(s) \leq s$, we obtain:
4.2 How to get an usable action functional \( \mathcal{L} H \)?

The previous relation shows which criterion on \( k_t \) and \( m \) implies the existence of some \( \lambda \) for which the limit exists, and for which theorem 2 is not trivially verified:

- If \( H^0(\alpha) > C|\alpha|^{1+\epsilon} \), the limit \( H(\alpha) \) will not exist.
- If \( \sum_{s=0}^{\infty} \frac{1}{k_t} \) converges, \( H^0(\alpha) \leq H'(\alpha) \leq H^0(\alpha) + |\alpha| \sum_{s=0}^{\infty} \frac{1}{k_t} \), and \( \mathcal{L} H \) is infinite almost everywhere.

The limit will exist if \( \sum_{t=0}^{\infty} \frac{1}{k_t} \langle \alpha, \int_X p(t, u)m(u) \, du \rangle \) is finite. When \( \frac{1}{\lambda(t)} H'(\lambda(t)\alpha) \) converges towards \( H(\alpha) \), the upper bound theorems apply. To get a non-trivial result, we need that \( \infty > \inf_{A_{r,x_0}} \mathcal{L} H(x) > 0 \), which will be true, if \( H \) is convex, for large enough radius \( r \). Finally, we get an exponential bound on probabilities of being outside a disk \( D \) of radius \( r \) around an optimum.

4.3 Sojourn times

The sojourn time of an EA process in a region \( D \) of the populations space \( X \), for a given simulation (that is, a run of the algorithm), can be defined as:

\[
\theta(D) = \sum_{t=0}^{\infty} \mathbb{1}_D(\xi_t)
\]

This random variable can be understood as the number of elementary time units during which the population is inside \( D \). If theorem 2 applies, we get the immediate upper bound:

\[
\begin{align*}
\text{E}(\theta(D)) & = \sum_{t=0}^{\infty} P(t, X\setminus D) \\
& \leq \sum_{t=0}^{\infty} 2.d.e^{-\lambda(t) \inf_{x \in D} \mathcal{L} H(x)}
\end{align*}
\]

that, together with Chebychev’s inequality, provides an estimate of the time after which the EA has “converged” into \( D \):

\[
\text{Theorem 3: Convergence speed}
\]

If theorem 2 (probability asymptotics) applies, then:

\[
\begin{align*}
\mathbb{P} \left( \sum_{0}^{\infty} \mathbb{1}_{X \setminus D}(\xi_t)dt > l \right) & \leq \frac{1}{l} \sum_{t=0}^{\infty} 2.d.e^{-\lambda(t) \inf_{x \in D} \mathcal{L} H(x)}
\end{align*}
\]

In other words, the probability to be outside \( D \) during more than \( l \) time units is bounded by \( \frac{1}{l} \text{E}(\theta(D)) \).
5 Limits and prospects

The use of the convergence theorems presented above is tricky: without an explicit form of $\lambda$ in the general case, we need to treat this problem for every particular case, taking into account the exact mutation operator (that is, the properties of $m$) and the characteristics of the objective function.

The critical role of the values $\frac{1}{k} \left( \alpha, \int_X p(t, u)m(u) \, du \right)$ emphasizes the importance of the regularity of the $m$ decision factor, strongly linked to the regularity of the fitness function $h$. An analysis of the impact of some regularity measure on the values of $\lambda$, $H$ and $\mathcal{L}H$ is first to be done. There are evidences that "fractal" quantities such as Legendre multifractal spectra have common features with quantities involved in this study. Future works will also concern the analysis of irregularities for some constrained classes of fitness functions, as in [11].

Finally, we worked on a voluntarily simplified framework respectively to the traditional models of EAs. Further efforts will also concern a more realistic EA model involving crossover and advanced selection schemes.

References


Appendix A: Notations

- $[m, m']$: set of the integers $i \in \mathbb{N}, m \leq i \leq m'$
- $[x, x']$: set of the reals $u \in \mathbb{R}, x \leq u \leq x'$
- $Y^X$: set of the functions $f : X \mapsto Y$
- $S_d$: permutations group on $[m, m']$
- $1_X$: characteristic function of a set $X$
- $\mathbb{P}(E)$: probability of an event $E$

- $d$: population size
- $X$: set of all individuals
- $p(t, x)$: probability density to have a population $x$ at time $t$
- $q(x)$: probability density for the initial population
- $\phi(t, u, x)$: probability density to produce a population $x$ at time $t + 1$ if the population at time $t$ is $u$

Appendix B: Sketch of the proofs

Uniform bound theorem

We begin by stating here some lemma useful in the proof of our main theorem, and whose proofs, rather straightforward, can be found in [15].

**Lemma 1: Hyperplane separation, real case**

Let $H$ be a convex function on $\mathbb{R}$, such that $H(0) = 0$ and $\mathcal{L}H$ be strictly convex. Let $A$ be a range in $\mathbb{R}$, with $s(A) = \inf_{x \in A} \mathcal{L}H(x)$. With these assumptions, $\forall \gamma \in \mathbb{R}^+$, there exists some $w \in \mathbb{R}$ verifying $\forall x \in A, \langle w, x \rangle - H(w) \geq s(A) - \gamma$

**Lemma 2: Hyperplane separation**

Let $H \in U(X)$, $r \in \mathbb{R}$ and $u \in X$ such that $\forall t \in \mathbb{R}, \forall q \in X$ $\langle v, q \rangle = 0$, we have $H(tv + q) = H(tv - q)$ (that means $v$ is a symmetry axis for $H$). If $A = \{ x \in X, \langle v, x \rangle \geq r \}$, $s(A) = \inf_{x \in A} \mathcal{L}H(x)$, $\forall \gamma \in \mathbb{R}^+$ there exists $w \in X$ verifying; $\forall x \in A, \langle w, x \rangle - H(w) \geq s(A) - \gamma$

**Lemma 3: Limit from below**

Let $H^h : X \mapsto \mathbb{R}$ convex such that $\forall \alpha \in X, \forall (h, h') \in \mathbb{R}^2_+$ with $h' \leq h$, we have $H^{h'}(\alpha) \geq H^h(\alpha)$. Let $\lambda \in (\mathbb{R}^+)^{\mathbb{R}^2}$, strictly decreasing, verifying:

- $\lim_{h \to 0} \lambda(h) = \infty$
- $\forall \alpha \in X, H(\alpha) = \lim_{h \to 0} \frac{1}{\lambda(h)} H^h(\lambda(h)\alpha)$ exists

Then $\forall \alpha \in X, H(\alpha) = \lim_{h \to 0} \frac{1}{\lambda(h)} H^h(\lambda(h)\alpha)$ is reached from below.

We are now able to prove theorem 1:

**Proof** Let $s = \inf_{\langle v, x \rangle > r} \mathcal{L}H(x)$. If $s = \infty$, $\mu^h(\{ x \in X, \langle v, x \rangle > r \}) = 0$

In the other case, let $w \in X$ such that $\forall x \mid \langle v, x \rangle > r, \langle w, x \rangle - H(w) \geq s$. 
Using the exponential Chebychev's inequality, we get:

\[
\mu^h (\{x \in X, \langle v, x \rangle > r \}) \leq \mu^h (\{(w, x) - H(w) \geq s \}) \\
\leq \int_X e^{\lambda(h)((w,x) - H(w) - s)} \leq e^{\lambda(h)} \left( \frac{1}{\lambda(h)} H^h(\lambda(h)w) - H(w) \right) e^{-\lambda(h)s} \leq e^{-\lambda(h)s}
\]

since the \( H \) limit is reached from below. \( \square \)

Application to EAs

\textit{Lemma 4: Exponential moments' properties I}

Let us consider an EA process with density \( p \) and transition function \( \phi \). For \( \alpha \in X \), we define \( H^\alpha(x) = \ln \int_X e^{(\alpha, u)} p(t, y) dy \)

Let \( E \) be a vector space, and \( f \in E^X \) measurable, \( G_t f(x) = \int_X f(u) \phi(t, x, u) du \), and \( \Gamma f(x) = \ln \int_X e^{\Gamma(u)} \phi(t, x, u) du = \ln G_t (e^f) (x) \)

Then \( H'^{\alpha+1}(\alpha) = \ln \int_X p(t, u) G_t e^{\Gamma(u)} (e^{(\alpha, u)}) (u) du \), and \( H'(\alpha) = \ln \int_X q(x) e^{(\Gamma_1 \ldots \Gamma_n)(e^{(\alpha, u)})}(x) dx \).

\textit{Lemma 5: Exponential moments' properties II}

Let us consider an EA process with density \( p \) and transition function \( \phi \) given by the expression:

\[
\phi(t, x, y) = \prod_{i=1}^d \frac{1}{m_i(x)} g_t \left( \frac{y_i - x_i}{m_i(x)} \right)
\]

Then \( H'^{\alpha+1}(\alpha) = \ln \int_X p(t, u) e^{(\alpha, u)} e^{\sum_{i=1}^d \gamma_i(\alpha; m_i(u))} du \), where \( \gamma_i(s) = \ln \int_R e^{s u} g_t(u) du \).

\textit{Lemma 6: Exponential moments' properties III}

Let us consider an EA process with isotropic initialization function \( q \), and with transition function \( \phi = \prod_{i=1}^d \frac{1}{m_i(x)} g_t \left( \frac{y_i - x_i}{m_i(x)} \right) \), with a mutation decision \( m \) verifying: \( \forall \sigma \in S_d, \forall x \in X, \forall i \in \{1, d\}, m_i(x_\sigma) = m_i(x) \) (with \( x_\sigma = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(d)}) \)).

Then \( \forall \sigma \in S_d, H'(\alpha) = H'(\alpha_\sigma) \).

Using lemma 4, 5 and 6, we can now prove theorem 2:

\textbf{Proof} Lemma 5, taking into account that \( \gamma_i(s) \geq 0 \), yields \( H'^{\alpha+1}(\alpha) \geq H'(\alpha) \).

We then use lemma 3 to prove that \( \frac{1}{\lambda(t)} H'(\lambda(t)\alpha) \) is increasing, so it converges if it remains bounded, and theorem 1 applies. Since \( H \) is unchanged when coordinates are permuted, \( \inf_{(x_i, x) \geq r} L H(x) \) does not depend on \( i \).

Let \( A^+_i(r) = \{x \in X \mid x_i \geq r\} \), and \( A^-_i(r) = \{x \in X \mid x_i \leq r\} \). \( \inf_{A^+_i(r)} L H(x) \) does not depend on \( i \), and \( \inf_{A^-_i(r)} L H(x) \) neither.

If \( A = \{x \in X \mid \exists i, |x_i - x_0| \geq r\} \), then \( A = \bigcup_i A^+_i(x_0 + r) \bigcup_i A^-_i(x_0 - r) \), and:

\[
\inf_A L H(x) = \min \left( \inf_{A^+_i(x_0 + r)} L H(x), \inf_{A^-_i(x_0 - r)} L H(x) \right)
\]

We thus get \( P(t, A) \leq 2.d.e^{-\lambda(t) \inf_A L H(x)} \)
Statistical Characteristics of Evolution Strategies *

Yoshiyuki Matsumura, Kazuhiro Ohkura, and Kanji Ueda
Kobe University, Rokkodai, Nada, Kobe 657-8501, Japan,
{matsumu,ohkura,ueda}@mi-2.mech.kobe-u.ac.jp

Abstract. Evolution Strategies (ES) are an approach to numerical optimization that show good optimization performance. The evolutionary behavior of ES has been well-studied on simple problems but not on large complex problems, such as those with highly rugged search spaces, or larger scale problems like those frequently used as benchmark problems for numerical optimization. In this paper, the evolutionary characteristics of ES on complex problems are examined using three different statistical approaches. These are (1) basic statistical measures at the function-value level, (2) Hotelling's $T^2$ for measuring the balance of exploitation and exploration at the individual-code level and (3) principal components analysis at the individual-code level for visualizing the distribution of the population. Among many formulations of ES, the fast-ES and the robust-ES are adopted for the analyses.

1 INTRODUCTION

Evolutionary Computation (EC) has widely been recognized as a robust approach to various optimization problems. There are three main approaches within the field of EC, these are Evolution Strategies (ES)[1], Genetic Algorithms (GA) [8] and Evolutionary Programming(EP)[7]. ES in particular has traditionally been concerned with numerical optimization.

ES has several formulations, but the most recent form is $(\mu, \lambda)$-ES, where $\lambda > \mu \geq 1$. $(\mu, \lambda)$ means that $\mu$ parents generate $\lambda$ offspring through recombination and mutation at each generation. The best $\mu$ offspring are selected deterministically from the $\lambda$ offspring and replace the current set of parents.

The ES research community has at least two major interests. One is pursuing the optimization of performance on benchmark problems or practical applications through empirical experiments on computers[6,12]. Work of this type tends to use only best and average population scores in order to validate the efficiency. This is due to a lack of a systematic methodology for the analysis of complex evolutionary behavior.

Other work has engaged in a more theoretical analysis of the evolutionary process, however this has typically entailed using a simplified ES[2–5,13].

* The authors acknowledge financial support through the "Methodology of Emergent Synthesis" project (96P00702) by JSPS (the Japan Society for the Promotion of Science).
Such approaches only address small simple problems which are mathematically tractable. Work of this type has produced some important results, such as 1/5 success rule. However, despite much work in this area, few results scale up to more practical and complex problems.

This paper attempts to show the evolutionary characteristics of ES on large complex problems through empirical experiments. The following statistical values are employed for this purpose: (1) basic statistical values at the function-value level, these are the average, the standard deviation and the coefficient of variation; (2) Hotelling’s $T^2$ for measuring the balance of exploitation and exploration at the individual-code level; (3) Principal components analysis for visualizing the distribution of the population at the individual-code level. These measures can be applied to problems of any dimensionality. They were adopted for our experiments in the expectation that we could find evolutionary characteristics which are independent of problem dimensionality.

This paper adopts FES (Fast ES) by Yao and Liu[12] and RES (Robust ES) by Ohkura, et al.[10,11] and compares the evolutionary behavior of each. There are two reasons for choosing the two ESs. Firstly, both ESs perform better than CES (Classical ES), particularly on multimodal benchmark problems [12,10,11]. Secondly, preliminary experiments showed that the evolutionary behavior of FES is similar to that of CES, whereas RES has some unique characteristics which are described below.

The rest of this paper is organized as follows. Section 2 briefly explains the computational procedures of FES and RES. Section 3 shows the results of our computer simulations. Finally, the summary of our experiments are given in Section 4.

## 2 COMPUTATIONAL PROCEDURE

### 2.1 Evolution Strategies

The computational procedure of ES can be described as follows[1]:

1. Generate the initial population of $\mu$ individuals, and set $g = 1$. Each individual is taken as a pair of real-valued vectors $(x_i, \eta_i)\forall i \in \{1, \ldots, \mu\}$, where $x_i$ and $\eta_i$ are the $i$-th coordinate value in $R$ and its strategy parameters larger than zero, respectively.
2. Evaluate the objective value for each individual $(x_i, \eta_i)\forall i \in \{1, \ldots, \mu\}$ of the population based on the objective function $f(x_i)$.
3. Each parent $(x_i, \eta_i), i = 1, \ldots, \mu$, creates $A/\mu$ offspring on average, so that a total of $A$ offspring are generated. At that time, offspring are calculated as follows: for $i = 1, \ldots, \mu$, $j = 1, \ldots, n$, and $k = 1, \ldots, \lambda$,

$$\eta'_k(j) = \eta_i(j)e^{\tau' N(0, 1) + \tau N_j(0, 1)}$$  \hspace{1cm} (1)

$$x'_k(j) = x_i(j) + \eta'_k(j)N_j(0, 1)$$  \hspace{1cm} (2)

where $x_i(j), x'_k(j), \eta_i(j)$ and $\eta'_k(j)$ denote the $j$-th component values of the vectors $x_i, x'_k, \eta_i$ and $\eta'_k$, respectively. $N(0, 1)$ denotes a normally distributed
One-dimensional random number with mean zero and standard deviation one. \( N_j(0,1) \) indicates that the random number is generated anew for each value of \( j \). The factors \( r \) and \( r' \) are commonly set to \( \left( \sqrt{2\sqrt{n}} \right)^{-1} \) and \( \left( \sqrt{2n} \right)^{-1} \) [1]. Notice that, when \( \eta_k(j) \) calculated by Equation (1) is smaller than a small positive value \( \epsilon \), i.e., the lower bound, \( \epsilon \) is assigned to \( \eta_k(j) \). Various types of recombination can also be performed before calculating Equations (1) and (2).

4. Calculate the fitness of each offspring \( (x'_i, \eta'_i), \forall i \in \{1, \ldots, \lambda\} \), according to \( f(x'_i) \).
5. Sort offspring \( (x'_i, \eta'_i), \forall i \in \{1, \ldots, \lambda\} \) in non-descending order according to their fitness values, and select the \( \mu \) best offspring out of \( \lambda \) to be parents of the next generation.
6. Stop if the halting criterion is satisfied; otherwise, \( g = g + 1 \) and go to step 3.

2.2 Fast Evolution Strategies

One of the extensions to ES, called FES (Fast ES), was proposed by Yao and Liu[12]. This replaces Gaussian mutation with Cauchy mutation for \( (\mu, \lambda) \)-ES. Cauchy mutation uses the following Cauchy distribution function:

\[
F_t(x) = 1/2 + (1/\pi) \arctan(x/t) \tag{3}
\]

where \( t = 1 \). The success of FES is explained as a result of a larger probability of escaping from local optima, due to the fatter convergence trails of the Cauchy mutation operator. They conducted empirical experiments using many test functions to demonstrate an improvement in performance, especially on multimodal problems.

2.3 Robust Evolution Strategies

When ES is applied to an optimization problem successfully, it shows evolutionary behavior similar to that of other evolutionary algorithms: the focus of the search shifts from a global region onto a local region. This arises from the gradual convergence of the population due to the direct effects of natural selection. Associated with this, \( \eta_i \) gradually becomes minute values. This has been considered a process of “self-adaptation”, which is one of major attractive features of ES. However, ES does not work well without a lower bound \( \epsilon \) on \( \eta_i \), otherwise its “self-adaptive” property is not effective. The relevance of \( \epsilon \) is not limited to a few exceptional cases, but is an important factor for ES. This is called the lower bound problem[11].

RES was proposed in order to overcome this problem. RES is designed so that it can utilize the effect of genetic drift[9] by allowing selectively neutral mutations, which are capable of rapidly increasing or decreasing \( \eta_i \) irrespective of natural selection. It has been confirmed that RES shows more robust self-adaptation than CES or FES. RES follows the same procedure as FES except for the following two points:
Procedure $O_{dup}$  

begin  
$\hat{\eta}_i(0) = \eta_i(0)$;  
for $p=1$ to $m$ do  
$\hat{\eta}_i(p) = \eta_{i(p-1)}(p)$;  
end for  
for $p=0$ to $m$ do  
$\hat{\eta}_k(p) = D(\hat{\eta}_i(p))$;  
end for  
end

Procedure $O_{del}$  

begin  
for $p=0$ to $m-1$ do  
$\hat{\eta}_i(p) = \eta_{i(p+1)}(p)$;  
end for  
for $p=0$ to $m$ do  
$\hat{\eta}_k(p) = D(\hat{\eta}_i(p))$;  
end for  
end

Procedure $O_{inv}$  

begin  
$p = \text{random}(1, \ldots, m)$;  
$\hat{\eta}_i(0) = \eta_{i(p+1)}(0)$;  
$\hat{\eta}_i(p) = \eta_{i(p)}(p)$;  
$\hat{\eta}_k(p) = D(\hat{\eta}_i(p))$;  
end

Fig. 1. Three stochastic mutations for strategy parameters in RES.

- New individual representation that holds redundant (i.e., inactive) strategy parameters. These inactive parameters have no effect on an individual.
- New stochastic mutation mechanisms for changing original (i.e., active) strategy parameters. These mutations replace, swap or copy active strategy parameters with inactive ones.

**Individual Representation** An individual $X_i$ is represented as follows, assuming that $i = 1, 2, \ldots, \mu$, $j = 1, 2, \ldots, n$, $k = 0, 1, \ldots, m$ and $x_i(j), \eta_{ip}(j) \in R$:

$$X_i = [x_i, (\eta_{i0}, \ldots, \eta_{ip}, \ldots, \eta_{im})]$$  \hfill (4)

$$x_i = (x_i(1), \ldots, x_i(j), \ldots, x_i(n))$$  \hfill (5)

$$\eta_{ip} = (\eta_{ip}(1), \ldots, \eta_{ip}(j), \ldots, \eta_{ip}(n))$$  \hfill (6)

Notice that each $x_i(j)$ has $(m + 1)$ strategy parameters.

**Reproduction** Assuming that $D$ is the same mutation as Equation (1), $\eta_{ip}$ are modified stochastically before a parent reproduces, according to the following new mutation operators before a parent reproduces:

- $O_{dup}$: shifts all of $\eta_{ip}(j)$ into the adjacent position of $(p + 1)$ and removes $\eta_{im}(j)$ from the list. Then, it modifies with $D$.
- $O_{del}$: discards $\eta_{i0}(j)$, moves $\eta_{ip}(j)$ to the adjacent position of $(k - 1)$ and places a large value $\eta_L$ at the $m$-th position. ($\eta_L$ is calculated as the smaller value either $\eta_{max}$ or $\sum_{p=1}^{m-1} \eta_{ip}(j)$.) Then, it modifies them with $D$.
- $O_{inv}$: swap $\eta_{i0}(j)$ with one of $\eta_{ip}(j)$, $p = 1, \ldots, m$. Then, it modifies with $D$.

Then, the component values $x_i$ are modified as:

$$x'_k(j) = x_i(j) + \eta_{i0}(j)\delta_j$$  \hfill (7)

where $\delta_j$ is a random number calculated anew for each $j$ based on Cauchy distribution. The other computational steps are the same as those of FES. Notice that RES is equivalent to FES when the probabilities of $O_{dup}$, $O_{del}$ and $O_{inv}$ are set at 1.0, 0.0 and 0.0, respectively.
3 COMPUTER SIMULATIONS

3.1 Test Functions and Conditions

We conducted a series of computer simulations using the test functions given by Yao and Liu [12]. Due to limited space, only the results for two of these test functions are presented. However, it should be noted that the evolutionary characteristics observed using these two functions were also consistently observed with the other test functions. The two test functions for which results will be presented are the sphere function $f_1(x)$ and Ackley’s function $f_2(x)$. These are frequently used as representatives of unimodal and multimodal functions respectively. They are defined as:

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \quad (-100 \leq x_i \leq 100)$$

$$f_2(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos 2\pi x_i\right) + 20 + e \quad (-32 \leq x_i \leq 32)$$

where $n = 30$ and $e$ is the base of natural logarithms$^1$. Both functions have the global minimum 0 at the origin.

The same conditions as [12] were adopted, that is, $(30,200)-(F,R)ES$ with mutation, but no correlated mutation or recombination was used. In the case of RES, $m$ and $\eta_{\text{max}}$ were set at 5 and 3.0, respectively. $O_{\text{dup}}, O_{\text{del}}$ and $O_{\text{inv}}$ were applied at the rates of 0.6, 0.3 and 0.1, respectively. The lower bound $e$ is set at 0, so as to observe only the evolutionary change of strategy parameters.

3.2 Basic Statistics

Firstly we describe the results in which basic statistics were gathered at the function-value level. We calculated (a) the average, (b) the standard deviation and (c) the coefficient of variation$^2$. These were calculated for both parents and offspring, because interesting evolutionary characteristics are observed when both offspring and parent values are examined.

Figures 2 and 3 show the results for $f_1$. With FES both parent and offspring averages converge until about generation 500 and generation 800 respectively after which there is no further practical improvement of the best and average values. This can be seen in Figure 2(a). Such evolutionary behavior is typical of the effect of the lower bound problem on FES. Premature convergence arises because strategy parameters tend to reach minute values too early in the search.

---

$^1$ $n = 30$ was selected because this is the standard dimensional size for test functions in [12]. The same results were obtained through a series of computer simulations, even when a larger dimensional size of $n = 50$ or 100 was employed.

$^2$ The coefficient of variation is the standard deviation divided by the average.
with the result that the population is practically unable to move to better regions of the search space\(^3\). In contrast, Figure 3(a) shows that with RES, the average continually improves up until the last generation in spite of the rather slow decrease in the offspring average.

Figures 2(b) and 2(c) show the transitions of the standard deviation and the coefficient of variation of FES for \(f_1\). Over evolutionary time both parents and offspring become less diverse with respect to their function-value. The convergence seen in Figure 2(a) thus reflects the fact that, as the run progresses, parents generate offspring which are increasingly similar to themselves. In contrast, with RES the coefficients of variation remains above a certain level, as shown in Figures 3(b) and 3(c). This implies that the diversity at the function-value level is maintained throughout the course of the experiment.

The results for \(f_2\) are shown in Figures 4 and 5. With respect to each of the statistical measures, both ESs exhibited similar behavior to that observed for \(f_1\).

### 3.3 Hotelling’s \(T^2\)

Hotelling’s \(T^2\) is a statistical measure of the multivariate distance of each observation from the center of a data set. In an evolutionary population, individuals located far from the center of population can be said to be engaged in exploration, while those located close to the center are engaged in exploitation. Hotelling’s \(T^2\) is useful because it gives a measure of the ratio of population individuals engaged in local search (exploitation) to those engaged in global search (exploration).

It is calculated as follows:

\[
T^2 = \lambda(\lambda - 1)(\bar{x} - \delta)'S^{-1}(\bar{x} - \delta)
\]

where

\[
S = \sum_{i=1}^\lambda (x_i - \bar{x}_i)(x_i - \bar{x}_i)'
\]

\[
\bar{x}_i = \frac{\sum_{i=1}^\lambda x_i}{\lambda}, \quad x_i \in N(\delta, \sigma_\zeta)
\]

where \(N(\delta, \sigma_\zeta)\) is the \(N\)-dimensional normal distribution in which \(\delta\) and \(\sigma_\zeta\) are the average and the standard deviation, respectively, and \(\lambda\) is the number of data points.

Figures 6 through 9 show the value of the \(T^2\) during both ES runs. In the early stages of each run, there is no clear difference between FES and RES with respect to the value of \(T^2\). However a clear difference between the two emerges as the runs progress. With FES, it can be seen that for both \(f_1\) (figures 6 a, b and c) and \(f_2\) (figures 8 a, b and c) there is no real change in the balance between exploitation and exploration over generation. In contrast, with RES the distribution changes in such a way that ultimately 15% of individuals are

\(^3\) It was demonstrated that the best value for a lower bound depends on the fitness landscape as well as on the simulation conditions[11].
located very far from the center of the population, while the most of the others are located very close to the center.

From these results it can be said that FES maintained the same search strategy for the entire duration of the run, whereas the search strategy employed by RES changed over generation. While initially employing the same strategy as FES, RES gradually shifted toward a strategy composed of fine exploitation and broad exploration.

3.4 Principal Components Analysis

Principal components analysis is a rigorous, quantitative method for visualizing multi-dimensional data. The principal components as a whole form the orthogonal basis of a space in which the data can be represented. The first principal component and the second principal component make a plane on which a data set is maximally distributed. If a data set is converged, plotting all the data points will result a line segment parallel to either axis.

This technique was applied in order to observe the distribution of offspring at the individual-code level. The results are shown in Figures 10 through 13. With FES, the the population converges until generation 800 and generation 600 for $f_1$ and $f_2$, respectively, as shown in Figures 10(c) and 12(c). In contrast, with RES the population repeatedly gathers and scatters over the plane for the entire duration of the run. Some snap shots of the RES population are shown in Figures 11 and 13. No stable line segments on the plane (indicating convergence) were observed at any point during the run, for either $f_1$ or $f_2$.

4 SUMMARY

Some statistical characteristics of the behavior of evolution strategies on large complex problems were investigated empirically. The results of our computer simulations can be summarized as follows:

- For both ESs, the evolutionary behavior observed and described below was independent of the modality of the benchmark problems.
- The lower bound problem occurs on FES but not on RES.
- At the level of function-values, Once an FES population has converged, its convergence sometimes results in a situation in which the population becomes trapped by local optima. Offspring are too similar to parents for the population to effectively engage in global search. In contrast, even when an RES population is converged, it is capable of continually finding better solutions. This is because RES generates offspring which are sufficiently different from their parents that global search is facilitated. At the same time, offspring are also generated which are similar to their parents, thus allowing fine-grained local search.
- At the individual-code level, FES continues to use the same search strategy, represented by the distribution of offspring, for the entire duration of a run.
This is the case even when the population has converged on a very small area of the search space. In contrast, while RES commences with the same search strategy as FES, it gradually shifts to a strategy which combines intensive exploitation with ambitious exploration.

At the individual-code level, a converged FES population always generates offspring that are too similar to parents. In contrast to the static behavior of a converged FES population, RES employs a dynamic strategy. RES changes its search strategy when the population starts to converge and alternates between short successive cycles of exploitation-oriented and exploration-oriented search.

References

Fig. 2. FES for $f_1$.  Fig. 3. RES for $f_1$.  Fig. 4. FES for $f_2$.  Fig. 5. RES for $f_2$.

Figs. 2 to 5: Basic statistics. The average, the standard deviation and the coefficient of variation are shown in (a), (b) and (c), respectively.

Fig. 6. FES for $f_1$.  Fig. 7. RES for $f_1$.  Fig. 8. FES for $f_2$.  Fig. 9. RES for $f_2$.

Figs. 6 to 9: Hotelling's $T^2$. The generations from which the data was taken are shown below each graph.
Figs. 10 to 13: The population of RES for $f_1$ on the plane constructed by the first principal component and the second principal component. The generations from which the data was taken are shown below each graph.
Consensus Sequence Plots and Error Thresholds: Tools for Visualising the Structure of Fitness Landscapes

Gabriela Ochoa

COGS – The University of Sussex, Falmer, Brighton BN1 9QH, UK
gabro@cogs.susx.ac.uk

Abstract. This paper investigates the occurrence of error thresholds in genetic algorithms (GAs) running on a wide range of fitness landscape structures. The error threshold, a notion from molecular evolution, is a critical mutation rate beyond which the evolutionary dynamics of a population changes drastically. The paper also introduces Consensus sequence plots, an empirical tool for locating error thresholds on complex landscapes. This plots were borrowed and adapted from theoretical biology. Results suggest that error thresholds occur in GAs but only on landscapes of certain degree of ruggedness or complexity. Moreover, consensus sequence plots can be useful for predicting some features of a landscape such as ruggedness and “step-ness”. We argue that error thresholds and consensus sequence plots, may become useful tools for analyzing evolutionary algorithms and visualising the structure of fitness landscapes.

1 Introduction

The error threshold — a notion from molecular evolution — is the critical mutation rate beyond which structures obtained by an evolutionary process are destroyed more frequently than selection can reproduce them. With mutation rates above this critical value, an optimal solution would not be stable in the population. The main purpose of this paper is to bring the notion of error threshold from molecular evolution to evolutionary computation. In other words, to explore whether a phenomenon similar to error thresholds is found in evolving populations of bit strings using a GA. In a previous work, we demonstrated empirically the existence of error thresholds on simple abstract landscapes using a standard GA (Ochoa & Harvey, 1998), we found also that the magnitude of error thresholds is lower when recombination is used. This paper, extends those findings by studying more complex landscapes, including a real-world engineering problem. A general empirical approach for locating error thresholds on complex landscapes is provided. This approach produces the so-called consensus sequence plots. The existence and characteristics of the error threshold on a given landscape depend upon the structure of the landscape. Thus consensus sequence plot can be useful tools for visualising the structure of a given landscape.

1 The term sequence is here interchangeable with string or genotype.
2 Quasispecies and Error Thresholds

This section introduces the notions of quasispecies and error thresholds from molecular evolution. Quasispecies theory was derived by Eigen and Schuster (1979) to describe the dynamics of replicating nucleic acid molecules under the influence of mutation and selection. The theory was originally developed in the context of pre-biotic evolution (studies of the origin of life), but in a wider sense it describes any population of reproducing organisms. A quasispecies is defined as the stationary population distribution of replicating macromolecules under mutation and selection.

The most prominent feature of the quasispecies model is the existence of an error threshold of replication. If replication were error free, no mutants would arise and evolution would stop. On the other hand, evolution would also be impossible if the error rate of replication were too high (only a few mutations may produce an improvement, but most will lead to deterioration). The notion of error threshold allows us to quantify the resulting minimal replication accuracy that still maintains adaptation. The quasispecies model, as stated originally, considered infinite asexual populations. Later extensions included finite populations and recombination. Most quasispecies studies considered simple landscapes, including single peak, double peak, and flat fitness landscapes. In contrast, the work of Bonhoeffer and Stadler (1993), described below, studied the evolution of quasispecies on two complex fitness landscapes.

3 Consensus Sequence Plots

The work of Bonhoeffer and Stadler (1993) studied the evolution of quasispecies on two correlated fitness landscapes, the Sherrington Kirkpatrick spin glass and the Graph Bipartitioning landscape. The authors described an empirical approach for locating thresholds on complex landscapes. In this paper, Bonhoeffer and Stadler's approach is borrowed and adapted. Instead of the quasispecies model, a GA is used as the underlying model of evolution. The resulting method can be applied for identifying error thresholds in GAs running on general complex landscapes. The approach is to calculate and plot the consensus sequence at equilibrium for a range of mutation rates. The consensus sequence in a population is defined as the sequence of predominant symbols (bits) in each position; it is plotted as follows: if the majority of individuals has a ‘1’ or ‘0’ in a position \( i \) the field is plotted white or black, respectively. The field is plotted grey if the position is undecided. Figure 1, shows an hypothetical population and calculates its consensus sequence. The plot shown in Figure 1 will correspond to a single line in a consensus sequence plot (See Figures 4 - 8). The Equilibrium State is reached when the proportion of different sequences in the population is stationary. This happens when evolution is simulated for a large enough number of generations. In practice, it is considered that the equilibrium is reached when several parameters of the population (e.g. the maximal and average fitness) reach equilibrium. According to Bonhoeffer and Stadler (1993) the error threshold may be approached from below or above, with both methods producing similar results.
Fig. 1. Calculating and plotting the consensus sequence of a population.

Approaching the Error Threshold from Below

To approach the error threshold from below, the simulation starts with a homogeneous population at the global optimum. This approach requires knowing the optimal string beforehand. Then, the population is allowed to reach equilibrium at a constant mutation rate of $\mu = 0.0$. Afterwards, the mutation rate is increased by a fixed, small step and the computation is continued with the current population. This process is repeated until a predefined maximum for the mutation rate is reached. The plot summarizes a single run, there is no averaging of multiple runs.

Approaching the Error Threshold from Above

To approach the error threshold from above, the simulation starts with a random population. Then the population is allowed to reach equilibrium at a constant predefined maximum for the mutation rate $\mu$. Afterwards, the mutation rate is decreased by a fixed small step and the computation continues with the current population. This process is repeated until the mutation rate is 0.0. Figure 2 outlines this algorithm. Notice that, in this case, it is not necessary to know the optimal string. Hence, in principle, this approach can be used for locating the error threshold on any complex landscape. Again the plot summarizes a single run, there is no averaging of multiple runs (see Figures 5 - 8).

For both approaches, the consensus sequence in the population is calculated and plotted at the end of each simulation cycle for each mutation step. The error threshold is characterised by the loss of the consensus sequence, i.e. the genetic information of the population. Beyond the error threshold the consensus sequence is no longer constant in time.

---

2 This value has to be high enough to be above the error threshold for the landscape under study.
Procedure Consensus_Plot { 
    p = p_max; /* Initial (high) mutation rate */ 
    initialize the population (randomly); 
    Run_GA; /* large number of generations */ 
    Calculate and plot the consensus sequence; 
    until (p = 0.0) { 
        p = p - p_step; 
        Run_GA; /* large number of generations */ 
        Calculate and plot the consensus sequence; 
    } 
} 

Fig. 2. Algorithm for producing a consensus sequence plot (from above).

4 Test Problems

Two families of abstract fitness landscapes were selected as test problems: the Royal Staircase and the NK landscape. In addition, a real-world engineering problem, the Wing-Box problem, and the simple Onemax function³, were explored.

Royal Staircase Family of Functions

The Royal Staircase family of functions were proposed by van Nimwegen and Crutchfield (1998) for analyzing epochal evolutionary search. This functions are related to the previous Royal Road functions (Mitchell et al., 1992). Although simple, Royal Staircase functions capture some essential elements found on complex problems, namely, the existence of highly degenerate genotype-to-phenotype maps (i.e. the mapping from genetic specification to fitness is a many-to-one function). Next, we present a description of the Royal Staircase class of fitness functions:

1. Genotypes are specified by binary strings \( s = s_1s_2 \ldots s_L, s_i \in \{0, 1\} \), of length \( L = A'^A' \), where \( A' \) is the number of blocks and \( A' \) the bits per block.
2. Starting from the first position, the number \( I(s) \) of consecutive 1s in a string is counted.
3. The fitness \( f(s) \) of string \( s \) with \( I(s) \) consecutive ones, followed by a zero, is \( f(s) = 1 + \left[ I(s)/K \right] \). The fitness is thus an integer between 1 and \( N + 1 \), corresponding to 1 plus the number of consecutive fully-set blocks starting from the left.
4. The single global optimum is \( s = 1^L \); namely, the string of all 1s.

Fixing \( N \) and \( K \) defines a particular problem or fitness landscape. For the experiments in this paper we selected two values of \( N \) (\( N = 3 \) and \( N = 4 \)) and a single value of \( K \) (\( K = 10 \)), that is, string lengths of 30 and 40 respectively.

³ The Onemax function gives the number of 1s in a bit string. Thus, the fittest string is the string of all ones.
NK Family of Landscapes

Kauffman NK family of fitness landscapes (Kauffman, 1993) are determined by two parameters: N and K. N is the string length (of binary strings) and K is the degree of epistatic interaction between the bits. An interesting property of the NK landscapes is that the ruggedness of the fitness landscape can be tuned by changing the parameter K. As K increases, the number of optima increases and the fitness correlation decreases.

Wing-Box Problem

The Wing-Box problem was formulated as part of the Genetic Algorithms in Manufacturing Engineering (GAME) project at COGS, University of Sussex. An industrial partner, British Aerospace, provided data from a real Airbus wing box for the definition of the problem. When designing aircraft structures, a common problem is to define structures of minimum weight that can withstand a given load. Figure 3 sketches the elements of a wing relevant to this problem. The wing is supported at regular intervals by slid ribs which run parallel to the aircraft’s fuselage. On the upper part of the wing, thin metal panels cover the gap between adjacent ribs. The objective is to find the number of panels and the thickness of each of these panels while minimizing the mass of the wing and ensuring that none of the panels buckle under maximum operational stresses. More details, and the equations for calculating the fitness function, can be found in McIlhagga et al. (1996).

![Fig. 3. Relevant elements of a wing. Wing dimensions are fixed. The variable elements are the number of ribs and the thickness of the top panels.](http://www.cogs.susx.ac.uk/projects/game/)

5 Experiments and Results

All experiments were run using a generational GA with fitness proportional selection and a population of 100 individuals. The genetic operations were 2-point crossover with a rate of 0.6 and the standard bit mutation. Mutation rates were expressed as mutation per genotype, a wide range of mutation values were tested. The string length varied according to the landscape under study. The GA was run in two modes: using mutation only (GA-M); and using both

---

http://www.cogs.susx.ac.uk/projects/game/
mutation and recombination (GA). Consensus sequence plots were produced for all landscapes under study following the method described in section 3.

The landscapes studied were classified (afterwards) into two groups. First, those where the error threshold could be clearly identified. In other words, where there was a distinguishable transition between an "ordered" (selection-dominated) regime and a "disordered" (mutation-dominated) one. Landscapes in this group were, incidentally, those with higher levels of ruggedness or discontinuity, thus we refer to them as ‘rugged’ landscapes. The second group of landscapes showed a very wide error transition band, in this sense the error threshold could not be located. These landscapes were those with low levels of ruggedness and discontinuity, thus we called them ‘smooth’ landscapes.

5.1 Rugged Landscapes

Figure 4 shows results for GA (crossover rate = 0.6) and GA-M (crossover rate = 0.0) on a Royal Staircase function with \( N = 3, K = 10 \). The error threshold was approached from below, starting from a homogeneous population at the global optimum and a mutation rate of \( p = 0.0 \); and from above, starting with a random population and a mutation rate of \( 5.0/L \) (where \( L \) is the string length). The plots illustrate the existence of a stable consensus sequence for mutation rates below the error threshold. In this case the consensus sequence is the single optimum string in the landscape (the string of all ones). Interestingly, error thresholds do not depend on whether they are approached from below or from above. The error thresholds for each fitness level or step can be clearly observed. Notice that error thresholds for all fitness levels are lower when recombination is used (GA).

![Fig. 4. The Consensus sequence on a Royal Staircase function \((N = 3, K = 10)\) for GA and GA-M. The horizontal axis show the consensus bit for each position \(i\), the vertical axis show the mutation rate expressed as mutations per genotype. The error threshold was approached from below and above (Section 3). The mutation step-size used was 0.25/L. Error thresholds are characterized by the loss of the consensus sequence, in this case the string of all ones. Also the intermediate error thresholds for each step or fitness level can be observed.](image-url)

To explore whether increasing the number of blocks would be reflected in the consensus sequence plot, we ran a similar experiment on a Royal Staircase
function with $N = 4$, and $K = 10$, for both GA and GA-M (Figure 5). The error threshold was approached from above. The consensus sequence is again the single optimum string in the landscape. The error thresholds for each fitness level or step can be clearly observed. Notice that a fourth level (step) appears compared to Figure 2. Again error thresholds are lower when recombination is used (GA).

Fig. 5. The consensus sequence on a Royal Staircase function ($N = 4$, $K = 10$), for both GA and GA-M. The horizontal axis show the consensus bit for each position $i$, the vertical axis show the mutation rate expressed as mutations per genotype. The mutation step-size used was $0.25/L$.

Figure 6 illustrates consensus sequence plots for both GA and GA-M on two $NK$ landscapes with $N = 24$; and $K = 6$ (Left), and $K = 12$ (Right). The error threshold is approached from above. Notice that consensus sequences reached by GA and GA-M are different, this is because these landscapes are multimodal, and different runs will most likely lead to different optima. For the more rugged landscape ($K = 12$) the transition is sharper, and the error threshold magnitude is lower when recombination is used (GA). For the less rugged landscape ($K = 6$) the transition seems to be smoother and there is no clear difference between GA and GA-M. As mentioned before, consensus sequence plots summarize a single GA run. However, our results suggest that the error threshold is independent of the initial population, since runs using different seeds produced qualitatively similar results.

Figure 7 shows results on the Wing-Box problem for GA and GA-M. The plots show the existence of a stable consensus sequence for mutation rates below the error threshold. The error threshold is visualized as the transition from a stable consensus sequence to a random sequence of bits. Notice that there is not a clear and single transition; from approximately bit 75 to bit 125 the error threshold looks higher than for the rest of the bits. Otherwise the transition seems to occur around 1.5 to 2.0 mutations per genotype. There is, in this case, no clear difference between GA and GA-M regarding the magnitude of the error threshold.
Fig. 6. The consensus sequence on two NK landscapes: \( N = 24, K = 6 \) (Left) and \( N = 24, K = 12 \) (Right). For both GA and GA-M. The mutation step-size used was 0.05/\( L \). Notice that the error threshold transition is sharper for the more rugged landscape (\( K = 12 \)).

Fig. 7. The consensus sequence on the Wing-Box problem, for both GA and GA-M. The mutation step-size used was 0.1/\( L \).

5.2 Smooth Landscapes

Figure 8 illustrates the consensus sequence on smooth landscapes for both GA and GA-M. Left, \( NK \) landscape with \( N = 24 \) and \( K = 2 \). Right, Onemax function with \( L = 30 \). In this cases, there is no clear error threshold, but instead a wide band of mutation rates where the consensus sequence is still present in the population.

6 Discussion

This paper demonstrates the existence of error thresholds in GAs (with and without recombination) running on several landscapes, including a real-world application. In this way, the notion of error threshold is brought to evolutionary computation. In a previous work, we have shown that error thresholds are related to the more familiar notion of an "optimal" mutation rate in GAs (Ochoa et al., 1999). The implication of this finding for GAs is two-fold. First, in understanding GAs' behavior, as insights about error thresholds will shed light on
optimal mutation rates. Second, practically, as this knowledge may help in devising heuristics for effective setting of evolutionary parameters. Notice that the study of error thresholds in GAs is relevant only when elitism is not used. Elitism is a widespread practice in GAs, however, the opinions on its advantages are divided. A number of researches in the field hold that a non-elitist strategy, which allows temporary deterioration to be accepted, may help to leave the region of attraction of a local optimum and reach a better optimum. Moreover, the general knowledge in the field of evolution strategies (Rechenberg, 1973), suggests that a non-elitist strategy is better suited for optimizing multimodal functions and for achieving self-adaptation of the mutation rates. Hence, we argue that non-elitist evolutionary algorithms are better suited for solving complex search and design problems, and that, therefore, the notion of error threshold is relevant in those scenarios.

This paper also introduced the consensus sequence plots. These plots, borrowed and adapted from theoretical biology (Bonhoeffer & Stadler, 1993), are new to evolutionary computation. They represent a novel way to visualize the structure of fitness landscapes, since features such as the “step-ness” of the Royal Staircase function can be clearly noticed (Figures 4 and 5). Moreover, the degree of ruggedness in a landscape was shown to be revealed by the plot (Figures 6 and 8). Consensus sequence plots may also serve as a tool to differentiate critical (and less critical) areas in the genotype, which may have practical implications when tackling real-world problems. First, it may be possible to infer important knowledge about an applied problem. Second, it may be possible to refine the genotype representations and optimal schedules for mutation rates.

Acknowledgements Thanks are due to H. Buxton and I. Harvey for their guidance and support. Thanks also to M. Sordo for critical reading of this document. The author is funded by CONICIT, Venezuela.
Bibliography


Experiments with Tuneable Fitness Landscapes

Colin R Reeves

School of Mathematical and Information Sciences
Coventry University
Priory Street, Coventry CV1 5FB
UK
C.Reeves@coventry.ac.uk, http://www.inis.coventry.ac.uk/~colinr/

Abstract. Kauffman's NK-landscapes have become a popular tool for investigating properties of heuristic search algorithms. In this paper we carry out some experiments with a more general, but still tuneable, class of landscapes which we call \( t, \theta \) landscapes. These landscapes are characterized by a parameter \( \theta \) which allows interactions at all orders, rather than merely at orders up to a fixed level as is the case with NK-landscapes. This is accomplished by fixing the magnitude and sign of the effects in an experimental design (ED) decomposition of a function. In some cases the epistasis variance is a simple function of \( \theta \), and can be specified in advance.

Further, by choosing some measure of the Hamming landscape associated with these functions, such as the number of local optima or the size of the global optimum's basin, it is possible to tune the landscape by mapping the effects onto a search problem. Some experiments are reported with a GA on these landscapes, with results that are rather surprising, in that the quality of the solution obtained appears to be poorly predicted by the properties of the associated Hamming landscape.

1 Introduction

A question that has given rise to considerable research in the field of evolutionary computation is that of determining the suitability of a given algorithm for solving a given problem. One approach has been to attempt to measure the amount of epistasis in the given problem. Typically, this entails some form of decomposition model [1-4] of the function, using Walsh transforms [5, 6] or an equivalent 'experimental design' interpretation [3, 4]. As is pointed out in [4], in principle we can never 'know whether we know' how epistatic a problem is without examining the Universe. Nevertheless, in practical problems where we can expect the degree of epistasis to be limited, we can often get away with our lack of knowledge and draw reasonably sound conclusions on the basis of a relatively small sample. Indeed, this assumption underlies the whole edifice of statistical experimental design (ED), which has years of demonstrable results to provide empirical justification for such a practice.

However, when asking the question as to how this relates to the likely performance of a particular algorithm, the problem is subtly different. Using a search algorithm involves a choice of representation, a choice of a set of operators, and the choice of a search strategy. It has been shown many times, as by Culberson [7] and Jones [8], for example, that the resulting landscape may differ substantially as a result of these choices, and consequently, the ease or difficulty of a given problem is not an invariant property of the function, but also depends on how the search is to proceed.

Reeves and Wright [4] have claimed that simple measures of epistasis are by themselves insufficient, and that insight into the nature of the epistatic effects is also required. This has not stopped people proposing simple measures, such as Davidor's epistasis variance measure, or Heckendorn's Walsh sums, or Jones' fitness-distance correlation (FDC), nor has it stopped people using them to draw conclusions about the hardness of a problem. For example, Soule and Foster [9] confirmed the hopelessness of trying to estimate epistatic effects from a small
random sample. DeJong et al. [10] use the maximum order of interaction in Kauffman's NK-landscapes, as a proxy for epistasis, while Kallel and Schoenauer [11] use FDC.

2 Background

These matters are discussed in much greater detail in [12]. Briefly, the starting point is the statistical decomposition model explored, together with its connections to experimental design theory, in [3,4].

\[ v_{pqr} = \mu + \alpha_p + \beta_q + (\alpha\beta)_{pq} + \gamma_r + (\alpha\gamma)_{pr} + (\beta\gamma)_{qr} + (\alpha\beta\gamma)_{pqr} \]  

(1)

where \( v_{pqr} \) is the fitness of the string \((p, q, r)\). The coefficients \( \mu, \alpha, \beta \) etc. are known as the 'effects'. Those involving just one letter indicate 'main effects' (due to one component only), those involving \( k > 1 \) letters represent 'interactions' of order \((k - 1)\). The coefficient \( \mu \) is simply the overall mean of all the \( v_{pqr} \) values.

2.1 Walsh representation

As shown in [3] this way of decomposing the fitness of all the strings in a Universe is equivalent to the Walsh decomposition. The 'mapping' from the Walsh coefficient numbers to the appropriate 'effect' is straightforward: the 'interaction level' of a particular Walsh coefficient is given by the number of 1s in its binary equivalent, and the specific factors involved by the positions of those 1s. For example, the coefficient \( w_{13} \) in a 4-bit problem: the binary equivalent of 13 is 1101, so this represents a 3-factor interaction, and the factors (reading from right to left) are \((A, C, D)\), i.e., \( w_{13} \) relates to the interaction \((\alpha\gamma\delta)\).

We note at this point a difference in terminology: in ED a \( k \)th order interaction means one between \((k + 1)\) factors (genes), whereas a Walsh coefficient of order \(k\) is usually taken to be between \(k\) factors. We shall adopt the latter convention.

3 Measuring Epistasis

As pointed out in [4], the question of how to measure epistasis is a difficult one in practice. Only if one knows the complete Universe is it possible to know infallibly the degree of non-linearity that exists. Nevertheless, a measure has been suggested based on the ED decomposition [1,2], as refined in [3,4]. This reduces to the ratio of the squared sum of interaction effects to the squared sum of all effects. Heckendorn et al. [6] define a closely related measure based on the absolute value of interaction effects of the same order. (Because of their use of Walsh function representations, Heckendorn et al. called these the Walsh sums.)

3.1 Davidor's 'variance' measure

Davidor's 'variance' measure, as modified by Reeves and Wright [4], can be written as

\[ \eta = \frac{\sum (v - \xi)^2}{\sum (v - \bar{v})^2} \]  

(2)

where the sum is over all strings and

\[ \xi_{pqr...} = \mu + \alpha_p + \beta_q + \gamma_r + \ldots \]  

(3)

is the sum of the linear effects (Walsh coefficients of order 1). The mean value of all strings in the Universe is denoted by \( \bar{v} \). For example, in the 3-bit case above, we have

\[ \eta = \frac{\sum_{pqr}[v_{pqr} - (\mu + \alpha_p + \beta_q + \gamma_r)]^2}{\sum (v_{pqr} - \bar{v})^2} \]
where the numerator easily reduces to
\[ \sum_{pqr} [(\alpha\beta)^2_{pq} + (\alpha\gamma)^2_{pr} + (\beta\gamma)^2_{qr} + (\alpha\beta\gamma)^2_{pqr}] \]
while the denominator is
\[ \sum_{pqr} [\alpha^2_p + \beta^2_p + \gamma^2_p + (\alpha\beta)^2_{pq} + (\alpha\gamma)^2_{pr} + (\beta\gamma)^2_{qr} + (\alpha\beta\gamma)^2_{pqr}] \].

In fact this reduction is quite general [12], and we can re-formulate the epistasis variance measure as the sum of the squared interaction effects; i.e.
\[ \eta = \frac{\sum \text{(interaction effects)}^2}{\sum \text{(all effects)}^2} \]

Thus, on the face of it, the more and the larger the interactions there are in a function, the higher the value of \( \eta \). Kauffman's \( NK \)-landscapes [14] for example, are usually characterised simply by the value \( K \), which defines the maximum order of interaction as being between \( K + 1 \) genes. However, the value of \( \eta \) itself cannot easily be controlled by changing the value of \( K \). Table 1 below shows that \( \eta \) can vary substantially for the same value of \( K \).

Table 1. The table shows the values of \( \eta \) measured for a set of 10 \( NK \)-landscapes with \( N = 10 \) and different values of \( K \). It is clear that there is a wide dispersion of values even for the same value of \( K \), although as \( K \) becomes larger \( \eta \) approached its ceiling of 1 and the scope for variation is less.

<table>
<thead>
<tr>
<th>( K )</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.40</td>
<td>0.09</td>
</tr>
<tr>
<td>2</td>
<td>0.62</td>
<td>0.13</td>
</tr>
<tr>
<td>3</td>
<td>0.77</td>
<td>0.08</td>
</tr>
<tr>
<td>4</td>
<td>0.86</td>
<td>0.06</td>
</tr>
</tbody>
</table>

An even more problematic question is what the value of \( \eta \) actually means, in terms of the difficulty of a given problem instance. As we have already said above, and expounded at greater length elsewhere [12], the fact that \( \eta \) reduces to a normalised sum of squared Walsh coefficients implies that it is invariant to a change in sign of a coefficient. Yet the sign can be of crucial importance in determining how difficult the actual problem is.

Other measures, such as the fitness-distance correlation measure [16, 15] and the 'amplitude spectrum' of Stadler and Wagner [18] have been suggested, in part to correct the neglect in the epistasis variance of the effect of the operator used in the search algorithm. However, while they do capture more of the flavour of the landscape, rather than merely that of the function \( v \), these approaches face similar problems; an analysis can be found in [12, 13].

4 Tuneable Landscapes

We note firstly that the above has not assumed anything about the number of alleles available to each gene. Although in what follows we shall consider only binary alphabets, in principle there is nothing in the method developed that requires this restriction.

4.1 An analysis of \( NK \)-landscapes

Kauffman's \( NK \)-landscapes [14] are by now rather familiar, so we shall assume the method of construction is known. (In fact, as hinted previously, there is a problem with Kauffman's
designations of a landscape at this point. Strictly speaking they are just functions, but we note the unspoken convention that in the absence of some specification of the operators, a Hamming landscape is assumed by default.)

Heckendorn and Whitley [17] have shown that NK-landscapes have some interesting properties: for instance, the number of non-zero effects at different interaction levels (i.e., Walsh coefficients of given order) is bounded above by the quantity

$$\min \left\{ \binom{N}{R}, \binom{K+1}{R} \right\} \quad \text{for } R \leq K + 1$$

and may in fact be still smaller in practice. The distribution of the non-zero effects has not previously been investigated. In Figure 1 we show some histograms obtained from 100 randomly generated NK-landscapes with $N = 10$ and $K = 1, \ldots, 6$, where for each bit the bits with which it interacts are assigned randomly. Statistics were gathered on the magnitudes of all the non-zero effects. (These effects are easily calculated by a well-known algorithm due to Yates [19,20].) From the point of view of potential epistasis, it is the non-zero interactions that are relevant.

If we examine these histograms, it is interesting to observe that the histograms for effects of different orders are relatively well-separated. We can also note that for $K = 1$, the main effects are on average larger than the non-zero 2-factor interactions. Further, at each increase in $K$, the new interactions tend to be relatively small, the major contribution of the increase in $K$ being to increase the magnitude of lower-order interactions. Thus 2-factor interactions are most significant when $K = 2$ and 3, but 3-factor interactions increase in importance through $K = 3, 4, 5$, being overtaken in turn by 4-factor interactions as $K = 4, 5, 6$ while 5- and 6-factor interactions can be observed to be "catching up" as $K$ increases further. Such a pattern is certainly one that would be expected to cause what [4] call "malign epistasis" (or deception), as some of the signs are bound to point in the "wrong" direction.

From the point of view of practical optimization problems, one wonders whether this sort of behaviour is likely to be normal, or if it is some sense aberrant. Certainly in traditional ED problems in the statistical literature, the general pattern is often rather different. One tends to see a hierarchy in which main effects have the largest magnitudes, followed by 2-factor interactions, 3-factor interactions etc. Real problems usually also involve uncertainty, and the desired outcome is often to have a situation that can be maintained easily even if the realized settings turn out to vary from the optimum. Cases where high-order interactions have a large influence are likely to lead to instability in engineering design, for example, and may not be chosen even if they are theoretically optimal.

4.2 An alternative

NK-landscapes are interesting and have led to some intriguing discoveries [14]. However, the above analysis suggests that an alternative class of tuneable landscapes can be developed that will be more flexible, and thus be capable of simulating more closely at least some of the type of optimization problems that are encountered in the real world. Using the ED/Walsh representation, we now propose another class that we call the $\ell, \theta-$landscape. The function is characterized by a parameter $\theta$ which allows interactions at all orders, rather than merely at orders up to a fixed level as is the case with Kauffman's NK-landscapes. The value $\ell$ is of course the string length, and is the same as $N$. Figure 4.2 presents a template for constructing such a landscape.

The functions $\mathcal{J}_k$ in Fig. 4.2 are indexed by $k$, which is the order of interaction; the other parameter (or parameter vector) $\theta$ is used to 'shape' the effects. A simple choice would be

$$\mathcal{J}(\theta, k) = U(0, \theta^k)$$

with $\theta$ a scalar and $U(\cdot, \cdot)$ the uniform distribution, which would model a smooth decay (if $\theta < 1$), or a smooth increase (if $\theta > 1$) in the maximum (and mean) size of the interaction
Fig. 1. Magnitude of $NK$ effects. Each histogram is based on the values found over 100 randomly generated $NK$-landscapes with $N = 10$ and values of $K$ from 1 to 6. There are 10 main effects (denoted by 1 in the legend), 45 2-factor interactions (denoted by 2), 120 3-factor interactions etc. However, for $NK$-landscapes there is a (generally much smaller) limit on the number of non-zero effects that can be generated. The true probability distribution is thus a mixed discrete-continuous density, and the histograms above simply plot the continuous part.

1. Define the mean and all main effects by drawing a random variate from some distribution $F(\theta)$, where $\theta$ is a parameter vector. Without loss of generality these values will be related to the case where the allele value is 1.
2. Define $k$-th order interactions by drawing a random variate from a distribution $F_k(\theta)$—again these are related to the case where allele values are 1s.
3. Allocate + and − signs to the interaction effects according to some prescription.
4. (Optional) Finally adjust the 'baseline' by modifying the coefficient $\mu$ so that the string with the minimal value in the Universe has value zero.

Fig. 2. A template for a class of tuneable landscapes
effects. The latter would create conditions for malign epistasis, or deception. The former would model a variety of conditions, depending on the exact value of $\theta$. We note here that a similar but less general procedure was used in [6], where the size of the interactions was constrained to follow a sharply decaying pattern. The other interesting step is the prescription in step 3. As also explained in [4], if the sign allocated to an interaction is aligned with its corresponding main effects, the effect on epistasis is benign. However, if the sign is working against the main effects, the outcome may be malign. We may allocate the signs at random, creating landscapes with a mixture of epistatic effects, or we may do so deterministically or otherwise in order to create particular types of function. Finally step 4 avoids negative fitness values, which may be convenient for some search algorithms.

4.3 The uniform $\ell, \theta$ landscape

In the case that the distribution is uniform as suggested above, it is relatively simple to compute an approximation to the expected value of the epistasis measure $\eta$. The Walsh coefficients of order $k$ are distributed as $U(0, \theta^k)$, and they enter the calculation of $\eta$ as the square of their magnitudes [12]. Thus, we need

$$E[X^2] = \theta^{2k}/3$$

where $X$ stands for an interaction of $k$ genes, so that the denominator of $\eta$ is

$$\frac{1}{3} \left[ \ell \theta^2 + \left( \frac{\ell}{2} \right) \theta^4 + \left( \frac{\ell}{3} \right) \theta^6 + \ldots + \theta^{2\ell} \right] = \frac{1}{3} \left[ (1 + \theta^2)^\ell - 1 \right]$$

while the numerator is

$$\frac{1}{3} \left[ (1 + \theta^2)^\ell - 1 - \ell \theta^2 \right].$$

Now $E[U/(U+V)]$ for random variables $U$ and $V$ is not in general the same as $E[U]/E[U+V]$. However, the error can be estimated using the method of stochastic differentials, and it is not hard to show that it is likely to be small here, so that

$$\eta \approx \frac{(1 + \theta^2)^\ell - 1 - \ell \theta^2}{(1 + \theta^2)^\ell - 1}. \quad (4)$$

Of course, this still does not predict the exact value of $\eta$ in a particular instance, but it is a modest advance on the case of the $NK$-functions, where the dependence of $\eta$ on $K$ can only be stated in very vague and general terms.

5 Epistasis is Not Enough

There are many properties of a landscape that determine the difficulty of solution in any particular case. These include the number of local optima, their depth and width, and the size of their basins of attraction. Of course, these properties depend on the nature of the operator used, which defines the neighbourhood graph for the points of the search space, and on the navigation strategy used by the search [13].

What is of interest is how these properties relate to the difficulty of solution using a GA, since, as we have shown above, functions of more or less difficulty may have the same values for Davidson’s variance measure $\eta$ (and for Walsh sums, since in both cases the sign of the interactions terms is effectively ignored), or for FDC. In the work reported below, we wanted to explore the practical effect of this restriction.
5.1 Experimental results

Initially, it was intended to hand-craft the signs of the interaction effects in order to produce sets of equivalent problems to study: 'equivalent' in the sense that \( \eta \) was the same, although having different properties in other respects. This was done for some small instances, but in general the question of how to identify the 'correct' signs to change is difficult in its own right. It was thus decided to treat this question itself as a search problem. We can define suitable objectives and then use a sequential search method to alter the signs in such a way as to cause increasing difficulty to a hill-climber on a Hamming landscape.

The objectives used were as follows: we looked for landscapes in which a change in sign led to:

- a reduction in the size of the basin of attraction of the global optimum, or
- an increase in the total number of local optima, or
- a reduction in the correlation coefficient between the size of the basins and the fitness of their local optima.

The rationale for the first 2 objectives is clear—the narrower the global basin, and the more local optima there are, the harder it is for a simple Hamming hill-climber to succeed. The third was prompted by Kauffman's observation (in a minimization context) that often 'the widest basins drain the deepest valleys' [14]. This effect can be measured by the correlation coefficient between basin size and fitness. Thus a simple search is more likely to end on a poor local optimum than on a good one if this correlation is reduced. Finally no change in sign of a Walsh coefficient was made unless at least two of these criteria were met.

It is clearly possible to change the orientation of the above prescription in an endeavour to make the problem easier—i.e., to reduce the number of local optima, to widen the global basin, and to increase the correlation between basin size and fitness. It is important to stress that the resulting landscapes all have the same measure of epistasis as the original one (and incidentally may often have the same FDC). Thus we have a family of landscapes defined merely by changing the sign of some Walsh coefficients. However, because of the difference in properties that ensue, in the case of a Hamming landscape hill-climber we can expect these landscapes to form a set of various difficulties. For the case of iterated steepest ascent (starting from independent random points), it is only to be expected that the global basin size will be the biggest factor in the success of the hill-climber in finding the global optimum—a result that is backed up by experiments (not reported here for lack of space).

The interesting question is how (or even whether) this and the other properties will affect the performance of a GA in the same clear-cut way. Two sets of experiments were carried out. Firstly, we generated an initial \( NK \)-landscape with \( N = 10 \) and \( K = 4 \) and determined the number of local optima and the size of their basins of attraction using Jones's 'reverse hill-climbing' technique [8]. The criteria outlined above could thus be evaluated exactly. Then a complete search sweep was carried out to determine landscapes of potentially increasing difficulty, in which each effect (taken in standard binary order) was in turn a candidate for alteration of its sign. The criteria were applied as indicated, and a change in sign accepted if at least two were satisfied. Cases where all three criteria apply simultaneously seemed to be few—at least for the relatively small instance investigated. The process was also used in reverse to generate landscapes of potentially decreasing difficulty.

On inspection of the results it was clear that the changes were not monotonic, as a sign change can lead to a completely new position for the local optimum. A typical pattern was for a succession of landscapes with progressively lower global basin sizes and correlations, followed by a shift of the local optimum to a new position; the number of local optima increased, and the correlation measure reduced, but the new basin size was larger than before. This pattern would then repeat. A similar effect was observed in the opposite direction when trying to make the landscapes easier.

The process could obviously continue until no sign change makes any difference (a sort of local optimum in the landscape of possible landscapes), but a sufficient number of alternative
landscapes (more than 60) were generated by a single sweep in each ‘direction’ (i.e. increasing or decreasing the potential difficulty), with a satisfactorily large range of values for the global basin size etc.

In the second set of experiments, the same procedure was followed using a $\ell, \theta$-landscape with $\ell = 10$, where $\theta$ was chosen to obtain approximately the same value of $\eta$ as that observed for the $NK$ case, $\eta = 0.896$. Equation (4) was used to obtain $\theta = 0.68$, and the resulting function had an actual value of $\eta = 0.894$—an extremely close agreement to what was predicted. There is evidence therefore that our confidence in Equation (4) is well-founded.

Despite having almost identical values of $\eta$, the initial $\ell, \theta$ landscape had twice as many local optima, and a narrower global basin. The subsequent modifications generated instances with global basins between 3.9% and 45% of the search space in the $NK$ case, and between 2.0% and 19% in the $\ell, \theta$ case. The numbers of local optima varied between 8 and 29 ($NK$) and 28 and 54 ($\ell, \theta$); the correlation coefficients varied between 0.522 and 0.978 and 0.545 and 0.943 respectively. Thus, *ceteris paribus*, for a Hamming hill-climber the $\ell, \theta$ landscapes should certainly be more difficult than the $NK$ ones.

5.2 Analysis

We have thus generated a large number of landscapes, all ostensibly the same as far as the epistasis variance is concerned. It is important to realize that the amplitude spectra of [18] would detect no difference either. The size of the global basin is a good predictor of how easy it is to find the global optimum for a hill-climber. But are there any patterns to the ease or difficulty of discovering high-quality solutions with a GA?

Thus a GA was run on each of the landscapes in the generated set in order to evaluate its effectiveness. A steady-state GA was used, with an initial population of 32 strings, which was allowed to run for 1024 evaluations in total. (The reasoning being that 1024 corresponds here to complete enumeration.) Fitness was based on linear ranking; at each generation one new offspring was produced using one-point crossover and mutation at a rate of .01 per bit. This offspring replace one of the worst 50% of the current population. The overall effectiveness of the GA was measured by running it 100 times with different initial populations, and counting the percentage that found the global optimum and the average % difference from the global optimum of the best solution obtained. Use was made of common random number seeds so that the 100 initial populations were the same for each of the landscapes examined.

The effectiveness of the GA was associated with the properties of each landscape by computing a simple correlation coefficient between these and the performance measures. Table 5.2 displays these values for both the $NK$ and $\ell, \theta$ landscapes described above. Group 1 landscapes were obtained by looking for increases in difficulty, group 2 when the search criteria were reversed.

The results are in some cases surprising. If we focus on the combined groups initially, we see that the chance of finding the global optimum of the various $NK$ landscapes by this GA is not associated with any of the landscape properties measured. (No correlation coefficient here is statistically significant.) However, there is an association for the case of the $\ell, \theta$ landscapes. Moreover, the association is what we would predict: the larger the global basin, the smaller the number of local optima, the higher the correlation between basin size and fitness, the better the chance of finding the global optimum. However, if we are concerned about the average % difference from the global optimum, we would hope that these correlations would be in the opposite sense, since in this case small values are good.

As can be seen, this is not so: it appears that for both types of landscape a greater difference is associated with the properties that we expected to betoken easier landscapes. This does hide some interesting anomalies: for the group 1 cases for $\ell, \theta$, the correlations are in the direction we would expect, so further investigation is needed.
Table 2. Correlation coefficients between GA performance measures and landscape properties.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Global basin # local opt r(size/fitness)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NK</td>
<td>Group 1</td>
</tr>
<tr>
<td>% g-opt</td>
<td>-0.393</td>
</tr>
<tr>
<td>%diff</td>
<td>0.813</td>
</tr>
<tr>
<td>% g-opt</td>
<td>0.149</td>
</tr>
<tr>
<td>%diff</td>
<td>0.658</td>
</tr>
<tr>
<td>% g-opt</td>
<td>0.207</td>
</tr>
<tr>
<td>%diff</td>
<td>0.811</td>
</tr>
<tr>
<td>% g-opt</td>
<td>0.807</td>
</tr>
<tr>
<td>%diff</td>
<td>-0.657</td>
</tr>
<tr>
<td>% g-opt</td>
<td>0.666</td>
</tr>
<tr>
<td>%diff</td>
<td>0.854</td>
</tr>
<tr>
<td>% g-opt</td>
<td>0.692</td>
</tr>
<tr>
<td>%diff</td>
<td>0.716</td>
</tr>
</tbody>
</table>

6 Conclusion

This paper has described work on several different aspects of landscape analysis and design. We have empirically investigated some of the properties of the Walsh coefficients of NK functions, and have defined an alternative class whose Walsh coefficients are generated directly, so that the epistasis variance \( \eta \) can be predicted with much greater precision than in the case of NK functions. We have also found that for the 'default' Hamming landscape of such functions, the NK landscapes seem easier to optimize. We have demonstrated a methodology whereby many landscapes having the same value of \( \eta \) can be tuned to have different properties, and we have empirically analysed the performance of a GA on these functions. It would seem that the properties of the Hamming landscape are useful in predicting the chance of this GA finding the global optimum, but that the reverse appears to be the case if we wish merely to attain a high-quality solution in a reasonable amount of computation. The reasons for this result clearly need further study.

References


Abstract. Persistence is a characteristic that distinguishes between different types of objects in our world: alive or dead, stable or unstable. The fact that some objects are able to persist, both in static and dynamic ways, and others are not, appears to be a cornerstone of why interesting things happen in our universe. Hence, the class of dynamical systems in which persistence plays an important role should be a very interesting class of systems. Persistence also appears to be an important aspect of the dynamics displayed by evolutionary search algorithms. In this paper I shall introduce a new measure of static persistence that is more sophisticated than the measure used in previous work. It is shown how this new measure may be useful in distinguishing between dynamics that are performing search and those that are not.

1 Introduction

This paper is part of on-going research into the relationships between evolutionary search, self-organisation and persistence within dynamical systems [4, 5]. In a previous paper it was conjectured that evolutionary search requires the accumulation and maintenance of persistence. Persistence was defined to be the degree to which the state of a dynamical system had or would remain the same over the time of a recorded trace of the system's behaviour. For an \( N \) dimensional dynamical system running for \( M \) time steps such a trace would consist of an \( N \times M \) matrix of values recording the state of each dimension of the system at each of the \( M \) time steps.

In [5] I described a simple measure of static persistence that worked on such traces of finite state dynamical systems (FSDSs). The measure calculated the persistence versus time values for each dimension and then took the average to calculate the persistence versus time graph for the whole system. For each dimension the degree of persistence at any one moment in time was measured as the percentage of the whole time of the trace for which the current state of the dimension had, or would be, the same. This simple calculation for each dimension is depicted in figure 1(a).

This measure was then used to explore the original conjecture by looking at the persistence behaviour of various different dynamical systems. If the conjecture were true we would expect to find the situation as depicted in figure
1(b). Evolutionary search processes (ESPs), the resulting dynamics of running a given evolutionary search algorithm on a given search space, were compared to some non-search dynamics and there was a clear difference in their characteristic behaviours as observed through the persistence measure. In this paper I wish to introduce a more sophisticated persistence measure that has improved foundations and wider applicability.

![Persistence vs. Time Graph](image)

**Fig. 1.** Distinguishing between ESP like and non-ESP like processes.

### 1.1 The dynamics of evolutionary search processes

Given an evolutionary search algorithm, we can use it to define a number of different possible dynamical systems. For example the state of each of the loci of the genomes that make up the individuals in the GA’s population could be used to determine the sub-state of each dimension of a dynamical system. Or one could simply look at the loci of the elite member of the population or indeed, one could use the average value for each loci of the genome. In this paper we shall always be looking at traces made using the values of the loci of the fittest member of the population.

The kind of dynamics created by evolutionary search processes, are interesting not because they reach definite basins of attraction, nor because they keep on moving but rather because the state of the system (the state of the population) is temporarily attracted to regions of the state space before moving on elsewhere. Sometimes this movement is to fitter regions of the state space (where more of the population is fit) or it is movement along neutral networks to regions of similar fitness. It is only in the simplest of situations that the dynamics of ESPs will become permanently stuck at a particular attractor. Indeed, it is only in these simplest of situations that most of our traditional approaches to analyzing dynamical systems become truly useful.

However, it is precisely the transient dynamics of ESPs that are interesting. It is here that GAs move away from local optimum and/or along neutral networks. This research is part of an attempt to better understand these transient dynamics.
by looking at one of their characteristics: persistence. The conjecture that this research is exploring is that, while the state of an ESP will continue to change, its degree of persistence will increase over time. While the results in the previous work were encouraging, the persistence measure used was very simple and had a few serious drawbacks.

1.2 Concerns with the original measure

The first concern with the original measure is that it was only interested in whether or not the value of a given dimension was the same or not. When using a dynamical system with only a few possible states for each dimension this is not a problem, and hence the measure was applicable to GAs operating over binary strings or cellular automaton whose cells only take on a few different states. However, in general it meant that the measure could not tell the difference between a change of state from 100 to 95 or from 100 to 5.

The persistence measure described here overcomes this failing and is able to recognise the difference between a small and a large change in state, thereby a system in which the state could take values between 0 and 100 but actually fluctuates between 100 and 95 would have higher persistence than a system whose state fluctuated equally often but between 100 and 5.

The second, and yet related, problem with the previous measure was that the persistence values calculated were very dependent on the representation used. What we would like to say is that a particular finite state dynamical system has a particular persistence behaviour. By persistence behaviour I am referring to the set of all implied persistence versus time graphs that a given system is likely to produce, each such graph having its respective probability of occurring on a given run of the FSDS. However, we can often choose to represent the states of a given FSDS in many different ways and, with the old measure, most of these would have a different persistence behaviour from each other. Thus the relationship between the FSDS and the persistence behaviour was rather arbitrary and therefore not very useful.

1.3 Adding a metric for each dimension

The way that we shall solve these concerns with the persistence measure is in part by tightening up the definition of the kinds of things to which it can be applied. We shall insist that when describing the FSDS on which the measure is to be applied that one also specify a distance metric for each dimension of the state space over which the FSDS is operating. This way we shall say that the persistence behaviour is unique for that particular FSDS operating over that particular metric space. Any equivalent FSDS operating over an isometric state space will therefore have the same persistence behaviour.

We also want to fix the dimensionality of the modeled dynamical system so that we can measure persistence for each dimension independently. So we will insist that the state space of the FSDS be constructed out of a set of $n$ subspaces,
$\{M_1, M_2, ..., M_n\}$ such that each $M_i$ is a metric space with an associated distance function $d_i$.

We shall label such a space an $M^n$ space, to indicate that it is an $n$-dimensional metric space. So now our dynamical systems are uniquely defined by the pair $(T, M^n)$, where $T$ describes the transitions of the FSDS. Let’s call the set of so define dynamical systems the set $\text{DSMS}$ to stand for the set of dynamical systems over metric spaces. So let’s take a look at the definition of persistence that assigns a unique persistence behaviour to each $(T, M^n)$ member of $\text{DSMS}$.

2 Persistence as a property of members of $\text{DSMS}$

Given such a $\text{DSMS}$, $D$, with $n$ dimensions, the persistence measure can be applied to any possible trace of $D$. Let the set $\{t_1, ..., t_l\}$ be the indexes of a trace, $W$ of $D$ that lasts for $l$ time steps. So during this trace $D$ took the states $s_{t_1}, s_{t_2}, ..., s_{t_l}$ in that order.

We are calculating the persistence value of the trace $W$ at time $i$, let’s call this $\mathcal{P}(W, i)$. The dynamical system was in state $s_{t_i}$ at that moment of time. As we have mentioned already, we want the persistence values of each dimension to be independent of each other and the persistence value of the whole system to be a simple average of the persistence within each dimension. So:

$$\mathcal{P}(W, i) = \frac{1}{n} \sum_{j} \mathcal{P}_j(W, i)$$

So we need to calculate the persistence measure within each dimension first. As $D$ was in state $s_{t_i}$ at time $i$, so it’s $j$th dimension was in state $w_{t_{i,j}} \in S_j$ at that time, where $S_j$ is the set of possible values for the state of the $j$th dimension. Hence the same trace, $W$ of the system, as viewed in only the $j$th dimension, $W_j$ looks like $w_{t_{i,1}}, w_{t_{i,2}}, ..., w_{t_{i,i}}$.

Given that the set $S_j$ is finite, and that it has an associated distance metric, $d_j$, then for each member $s \in S_j$ we can talk about the expected distance between $s$ and a randomly picked member of $S_j$. We shall call this distance $\hat{s}$. In effect this is the distance that we would expect a random trace to move from $s$ in one time step.

We are going to take the position of saying that the state of the $j$th dimension, $W_j$ can only be said to be still persisting (in both directions of time) if between $i$ and the current moment of time the state of the system has never moved further than it is moving less far than $\hat{s}$. In other words, a given previous (or future) state of the system can only be said to be still persisting if the state of the system has never moved as far as one would expect by the movement of a random jump in the $j$th dimension. In other words less far than one would expect by the movement of a random jump in $S_j$. Assuming that a random jump has equal chance of picking each member of $S_j$, then $\hat{s}$ can be defined as follows:

$$\hat{s} = \frac{1}{|S_j|} \sum_{r \in S_j} d(s, r)$$
So we wish to calculate the persistence value of the jth dimension at time i. To do this we shall define a second sequence \( U \) of \( I \) values that will hold the maximum distance, up to \( \overline{w}_j \) moved so far between each point of the trace and the point we are interested in. So we define the points of \( U \) as follows:

\[
\forall 1 \leq k \leq i \quad u_k = \min\{\overline{w}_{t_i,j}, \max\{d(w_{t_r,j}, w_{t_i,j})\} \mid \forall k \leq r \leq i\}
\]

and:

\[
\forall i < k \leq m \quad u_k = \min\{\overline{w}_{t_i,j}, \max\{d(w_{t_r,j}, w_{t_i,j})\} \mid \forall i \leq r \leq k\}
\]

So now from this sequence \( U \) we can calculate the persistence value as follows:

\[
P_j(W, i) = \frac{1}{(l, \overline{w}_{t_i,j})} \sum_{k=1}^{k \leq l} \overline{w}_{t_i,j} - u_k
\]

which simply calculates the area of the shaded region in figure 2 and we have scaled this area to be a value between 0 and 1.0. So once we have calculated the persistence value for each dimension, then we simply take the average of these to get the persistence value of the whole system at that time in the trace. So:

\[
P(W, i) = \frac{1}{n} \sum_{j=1}^{n} P_j(W, i)
\]

as stated above, and so this final value will also be between 0 and 1.0. In this way we can calculate the persistence value for each point of the trace. As we are working with dynamical systems in the set DSMS, hence the values of the distance functions are fixed by the defined metric space, \( M^n \). The possible trace behaviours of each such system is fixed by the dynamics of the FSDS defined by \( T \). As we are basing our persistence measure soley on these distance functions and trace behaviours therefore the persistence behaviour of any dynamical system in DSMS is a fixed property of that dynamical system.
2.1 Natural members of DSMS

Many dynamical systems are natural members of DSMS in the sense that the obvious way in which we may observe them includes a natural n-dimensional metric space, an $M^n$. In an observed system each dimension of the DSMS space is a degree of freedom of the observed system. While we may chose the number of degrees of freedom to which we are going to model a given phenomenon, once the model is described as an FSDS there is normally a natural metric for each of the decided upon degrees of freedom. Examples include the metric spaces that come with distance, time and temperature, all of which are effectively going to be sub-spaces of the real line with its normal distance metric, and with binary strings there's the natural metric space defined by the Hamming distance. Hence, most FSDSs that we deal with do not just involve the dynamics of movement within a disconnected set of objects, rather they describe movement within an $M^n$ space. The naive mistake in the earlier work [5] was to lose the metric relationships of the set of states over which the FSDS is behaving.

3 Preliminary results with the new persistence measure

![Graphs](image)

Fig. 3. (a) Random trace and (b) random walks as seen by the new persistence measure.

So let's take a look at some graphs from this new persistence measure. The first kind of persistence graph to look at is one generated using a random trace. We can see such a graph in figure 3a) which clearly shows that a random trace has a very flat persistence graph whose value is close to zero persistence. We can use this as a baseline against which to compare all other traces.

The next kind of dynamics to look at are those created by ESPs that simply perform random walks. Figure 3b) shows the persistence graph calculated from the dynamics of a random walk on a large search space. The search space is the same as the real world problem in [1, 3] relating to an octopod robot. Each line is generated with a different mutation rate. The rates used are 0.02, 0.03, 0.04,
0.08, 0.16 and 0.32. The bottom, flat line with a constant persistence value of zero is the random trace again, put here to demonstrate how close to zero the persistence value of the random trace is. The next lowest line, closest to that of random search, is the persistence graph of the random walk with mutation rate 0.32, the greatest mutation rate. The line with the highest persistence value is the graph of the random walk with the mutation rate of 0.02, the lowest value. The remaining lines are in order of increasing mutation rate. The higher the mutation rate the closer the line is to that of random search.

![Persistence graphs](image)

**Fig. 4.** The persistence graphs of (a) a GA and (b) a cross-over based evolutionary search algorithm on the wing box real world problem in [2].

We can see in figure 3b) that nearer the edges of the graph the persistence value decreases. This dropping off of the persistence value can be understood in a similar manner as was discussed in [5] for the previous persistence measure. Near the edges of the trace the persistence measure has no knowledge of how the ESP might have behaved before the beginning of the trace or after the end of the trace. Therefore the closer a time step is to the edges of the trace the less accurately the measure can 'know' that the state has or will persisted for and therefore there is a natural tendency for the edges of the persistence graph to be lower. However, the lines presented in this graph represent the persistence characteristic of dynamics generated with mutation rates that are known to be constant. Therefore, this kind of symmetric curved line represents the persistence graph of the dynamics of an ESP with constant mutation rate.

### 3.1 Persistence of a genuinely searching ESP

Next we shall look at some persistence graphs of traces generated by ESPs that are genuinely doing search. Figure 4 shows the persistence graphs of two different evolutionary search algorithms running on a real world problem (the problem is to optimise the design of an aircraft wing, see [2]) and we can clearly see an increase in the level of persistence over the time of the graphs, this time in
Fig. 5. The persistence graphs of (a) a mutation based evolutionary search algorithm on the wing box problem [2], and a genetic algorithm on an real world problem to control an octopod robot [1,3]

a non-symmetric manner. The fitness of the state of the ESP is also plotted, although the actual values have been scaled to be plotted over the same range as the persistence graph. In the wing box problem we are trying to minimise the fitness value. Figure 5 shows two more examples of the persistence measure being used on traces from genuinely searching ESPs (the problem in figure 5(b) is that of optimising the neural network controller for an octopod robot, see [1,3]). In all four graphs as fitness gets better, so the persistence value gets higher. We can think of this as an indication that the effective mutation rate of the ESP decreased over time as the fitness of the state of the ESP gets better.

It is worth remembering that the persistence value is calculated without any knowledge of the fitness values of the states of the trace. The persistence measure only needs to be given the trace of the values that encode the solutions. From this data it is able to create a persistence versus time graph that reflects the basic structure of the fitness versus time graph. The connection between the two is almost certainly to do with the notion of the effective mutation rate of the ESP. Unfortunately, there is not enough space to discuss this issue further here. What is clear however, is that the persistence measure is able to reveal aspects the searching behaviour of the ESP.

3.2 A non-search dynamic

To contrast with the search dynamics that have been looked at so far, let’s take a look at the persistence within the dynamics of the Lorenz attractor. While the Lorenz attractor as defined by its equations is not an FSDS, we can still take a discrete trace from it and thereby apply this measure to it. However, we first need to discuss how the persistence measure can be applied to any arbitrary trace of a dynamical system.

The issue that needs to be resolved is that of the expected behaviour of a random search. The persistence measure is defined in relation to a probability
distribution across the space of possible states of the FSDS. This distribution effectively describes the behaviour of random search over the same space as that of the ESP, and therefore we can use it to determine the degree to which the trace being looked at is not a random trace. The Lorenz attractor, as with many dynamical systems, is defined over the real line, or rather each of the three variables of the dynamical system can take any real value. Clearly we cannot use a uniform distribution over an infinite set as our random distribution. We therefore have two choices as how to define a random distribution against which to meaningfully measure the persistence of the Lorenz attractor. Either we use a distribution like a Gaussian distribution, or we consider the Lorenz attractor over a bounded sub-set of the real line so that we can still use a uniform distribution over the considered set. Here we shall only look at this second approach.

We shall first look at an automatically generated bound that is simply determined by taking the maximum and minimum values found within the trace for each of the n dimensions of the dynamical system, thereby enclosing the trace within the smallest n dimensional cube (or rather rectangle) that will hold it. The random distribution is now defined to be a uniform distribution over this n dimensional cube.

Fig. 6. Various persistence graphs of traces generated with the Lorenz attractor.
Figure 6(a) shows the result of using this automatic bounding technique to generate a persistence graph from a trace of the Lorenz attractor. In figure 6(b) we shall also look at two manually chosen bounds for the same trace. The top line uses a bound of -1000.0 to 1000.0 for each dimension whereas the middle line uses a bound of -100.0 to 100.0. The lowest line is again that generated by using the automatic bound. Despite their differences, all three of the resulting persistence graphs are clearly different in nature from those of ESPs seen so far. This supports the idea that search dynamics can be recognised as being different from non-search dynamics, with respect to persistence.

Finally we shall take a brief look at a trace that moves into the Lorenz attractor from a distance. Figure 6(c) shows the trace being used and figure 6(d) shows the resulting persistence graph, generated using the automatic bounding technique described above. What is interesting here is that the resulting persistence graph looks much more like that of a search process. Starting from a very low level the persistence value climbs up to a higher value. We can maybe understand this in terms of it being the dynamics of ‘finding’ the Lorenz attractor. Interestingly, once the attractor has been reached we then see a vaguely cyclic persistence graph that is not at all like that of a search ESP.

3.3 Conclusions

A new persistence measure was introduced, its foundations explained and some preliminary results discussed. It was argued that it is the transient dynamics of evolutionary search algorithms that are most interesting and hardest to analyse. The persistence measure presented was introduced as a possible tool to observer these transient dynamics. From the results we can see that the persistence measure is capturing some interesting features of the dynamics being looked at. This is most clearly illustrated by the way that the persistence graph is able to mimic aspects of the fitness graph without having any knowledge of the fitness values. Obviously this measure requires further evaluation, but it has already demonstrated interesting promise.

References

An Analysis of Dynamic Severity and Population Size

Karsten Weicker
University of Stuttgart, Institute of Computer Science,
Breitwiesenstr. 20-22, 70565 Stuttgart, Germany,
email: Karsten.Weicker@informatik.uni-stuttgart.de

Abstract. This work introduces a general mathematical framework for non-stationary fitness functions which enables the exact definition of certain problem properties. The properties' influence on the severity of the dynamics is analyzed and discussed. Various different classes of dynamic problems are identified based on the properties. Eventually, for an exemplary model search space and a \((1, \lambda)\)-strategy, the interrelation of the offspring population size and the success rate is analyzed. Several algorithmic techniques for dynamic problems are compared for the different problem classes.

1 Introduction

Dynamic optimization problems are an area of increasing importance and increasing interest. This is reflected in the number of publications at recent conferences as well as in emerging applications in industry, e.g. control tasks and dynamic scheduling. In research, dynamic problems are primarily used to demonstrate adaptation of evolutionary algorithms and to motivate manifold different techniques to cope with dynamics. Nevertheless, we observe a significant lack of established foundations to examine and compare problems as well as techniques.

This work presents one possible approach to reach a common framework for experimental as well as theoretical examinations of different dynamic problems. The framework is used in this work for a theoretical analysis of the influence of population size and certain techniques to cope with dynamic problems on the success rate of algorithms based on local mutation operators.

2 Related Work

Optimization of dynamic problems is primarily driven by the experimental analysis of special optimization techniques on different problems. One classical benchmark function is the moving peaks problem which was recently implemented differently in various problem generators (Grefenstette, 1999; Branke, 1999b; Morrison & De Jong, 1999). Those problems are usually determined by the position of the peaks and their height and width. Nevertheless the parameterization of the dynamics differs from problem generator to problem generator. Also there
is no common basis for the comparison of different dynamic function optimization problems. An overview on dynamic problems as well as techniques is given in (Branke, 1999a).

Two standard techniques used to improve evolutionary optimization of dynamic problems are memory for previous solutions and an increase of the population diversity. Memory can be introduced either for each individual by using a diploid representation (e.g. Goldberg & Smith, 1987; Lewis, Hart, & Ritchie, 1998) or by an explicit global memory (e.g. Mori, Imanishi, Kita, & Nishikawa, 1997; Branke, 1999b; Trojanowski & Michalewicz, 1999). One possibility to increase the diversity are random immigrants within a hypermutation operator (Cobb & Grefenstette, 1993; Cobb, 1990; Grefenstette, 1999).

One focus of this work is the analysis of the population size for dynamic problems and local mutation operators. The population size was already often target of various examinations. De Jong and Spears (1991) and Deb and Agrawal (1999) analyzed the interaction of population size and genetic operators. Mahfoud (1994) derived a lower bound for the population size in order to guarantee niching in sharing techniques. Miller (1997) deduced optimal population sizes for noisy fitness functions. An overview on topics in population sizing can be found in (Smith, 1997).

Other theoretical investigations of evolutionary algorithms and dynamic problems may be found in Rowe (1999) and Ronnewinkel, Wilke, and Martinetz (2000).

3 Dynamic Problem Framework

This section introduces a general framework to describe and characterize dynamic fitness functions. The goal of this framework is to establish a basis for comparison and classification of non-stationary functions as well as for theoretical results on problem hardness and algorithms’ power. Such results are possible since the framework enables the exact definition of problem properties.

The basic idea of the following definition is that each dynamic function consists of several static functions where for each static function a dynamic rule is given. The dynamics rule is defined by a sequence of isometric, e.g. distance preserving, coordinate transformations and fitness rescalings. The possible coordinate transformations are translations and rotations around a center.

Definition 1 (Dynamic fitness function (maximization)). Let \( \Omega \) be the search space with distance metric \( d: \Omega \times \Omega \rightarrow \mathbb{R} \). A dynamic fitness function \( F = (F^{(t)})_{t \in \mathbb{N}} \) with \( F^{(t)}: \Omega \rightarrow \mathbb{R} \) for \( t \in \mathbb{N} \) is defined by \( n \in \mathbb{N} \) components consisting of a static fitness function \( f_i: \Omega \rightarrow \mathbb{R} \) (\( 1 \leq i \leq n \)) with optimum at \( 0 \in \Omega \), \( f_i(0) = 1 \), and a dynamics rule with

- coordinate transformations \( \left( c_i^{(t)} \right)_{t \in \mathbb{N}} \) with \( c_i^{(t)}: \Omega \rightarrow \Omega \)

where \( d(c_i^{(t)}(\omega_1), c_i^{(t)}(\omega_2)) = d(\omega_1, \omega_2) \) for all \( \omega_1, \omega_2 \in \Omega \) and

- fitness rescalings \( \left( s_i^{(t)} \right)_{t \in \mathbb{N}} \) with \( s_i^{(t)} \in \mathbb{R}_0^+ \).
The resulting dynamic fitness function is defined as

\[ F^{(t)}(\omega) = \max \left\{ S^{(t)}_1 f_1(C^{(t)}_1(\omega)), \ldots, S^{(t)}_n f_n(C^{(t)}_n(\omega)) \right\} \]

where \( C^{(t)}_i = C^{(0,t)}_i \) and \( C^{(t_1,t_2)}_i = c^{(t_2)}_i \circ \ldots \circ c^{(t_1+1)}_i \) are the accumulated coordinate transformations and \( S^{(t)}_i = S^{(0,t)}_i \) and \( S^{(t_1,t_2)}_i = \prod_{t=t_1+1}^{t_2} s^{(t)}_i \) are the accumulated fitness rescalings.

The placement of each optimum at position 0 and the initial optimal fitness 1 has been chosen for simplification of the analysis only. It does not put any constraint on the function since the rescaling of the fitness value and the positioning may be achieved by the first dynamics rule. Furthermore it is assumed that at each time step only one of the component fitness functions attains the optimal fitness value, i.e. there is at each time step only one global optimum.

Before problem properties are considered in detail a few examples are given how problems from recent publications may be studied in the given framework. One example are the moving peaks problems (e.g. Branke, 1999b; Morrison & De Jong, 1999) which can be realized by one component function for each peak and an additional function \( f_0 \equiv 0 \). Then, the motion of the peaks can be given by the coordinate transformations and the peak heights may be changed by fitness rescaling. Note, that this framework allows no random influences, i.e. each fitness function produced by above paper's problem generators defines a new dynamic fitness function within the framework. The fitness function by Trojanowski and Michalewicz (1999) divides the search space in different segments which hold each a peak where the height of the peaks is changing according to a schedule. In this framework the static functions may be defined on the according segment only. Where most other dynamic problems exhibit only coordinate translations, Weicker and Weicker (1999) presented a fitness function with rotation as coordinate transformations which may be reproduced easily within the framework.

Using Definition 1 several problem properties may be identified which influence the hardness of a dynamic problem. The following definition formalizes a few basic problem properties inherent in the dynamics of the problem.

**Definition 2 (Basic problem properties).** Let \( F_i = (f_i, (c^{(t)}_i)_{t \in \mathbb{N}}, (s^{(t)}_i)_{t \in \mathbb{N}}) \) and \( F_j = (f_j, (c^{(t)}_j)_{t \in \mathbb{N}}, (s^{(t)}_j)_{t \in \mathbb{N}}) \) be two components of a dynamic fitness function \( F \). Then the following properties are defined with respect to the coordinate transitions and a set of time steps \( T \subseteq \mathbb{N} \).

- with constant dynamics: \( \text{const}_c(F_i) \) iff \( \forall t \in T \, c^{(t)}_i = c^{(t+1)}_i \)
- stationary: \( \text{stat}_c(F_i) \) iff \( \forall t \in T \, c^{(t)}_i = \text{id} \)
- weak periodic: \( \text{wperiod}_c(F_i) \) iff \( \forall t_1, t_2 \in T, t_1 < t_2 \, C^{(t_1,t_2)}_i = \text{id} \)
- homogeneous: \( \text{hom}_c(F_i, F_j) \) iff \( \forall t \in T \, c^{(t)}_i = c^{(t)}_j \)

The properties \( \text{const}_s(F_i), \text{stat}_s(F_i), \text{wperiod}_s(F_i), \) and \( \text{hom}_s(F_i, F_j) \) are analogously defined with respect to the fitness rescalings where \( c \) is replaced by \( s \) and \( \text{id} \) by 1. Additionally the following property is defined using the fitness rescalings.
alternating: \( \text{alter}_s(F_i) \) iff \( \forall t \in T \ 1 < \frac{S_i^{(t)}}{S_i^{(t+1)}} < \frac{S_j^{(1)}}{S_j^{(1+1)}} \)

In general, a dynamic fitness function is denoted to have constant dynamics if it holds that \( \forall 1 \leq i \leq n \ (\text{const}_c(F_i) \land \text{const}_s(F_i)) \). This is also true for the other defined properties.

The next section defines certain severity measures and analyzes how the severity is affected by the problem properties.

4 Problem Properties and Severity

This section presents in the following definition four different severity measures and discusses the influence of the basic problem properties on the severity in the remainder of the section.

Definition 3 (Severity of dynamics). Let \( \Omega \) be a search space, \( \text{dia} \in \mathbb{R} \) the maximal distance within \( \Omega \), and \( F \) a dynamic fitness function consisting of components \( \left(f_i, (c_i^{(t)})_{t \in \mathbb{N}}, (s_i^{(t)})_{t \in \mathbb{N}}\right) \) where \( 1 \leq i \leq n \). Then the following severity measures are defined

- minimal general severity: \( \text{minsev}(F^{(t)}) = \min_{\omega \in \Omega} \max_{\omega} \maxcomp(\omega) \)
- maximal general severity: \( \text{maxsev}(F^{(t)}) = \max_{\omega \in \Omega} \max_{\omega} \maxcomp(\omega) \)
- average general severity: \( \text{avgsev}(F^{(t)}) = \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \maxcomp(\omega) \)

where \( \maxcomp(\omega) = \max_{1 \leq i \leq n} \frac{d(c_i^{(t)}(\omega), c_i^{(t)}(\omega))}{\text{dia}} \). Moreover, the severity of the optimum is defined as \( \text{optsev}(F^{(t)}) = \frac{d(\text{opt}^{(t)}, \text{opt}^{(t)})}{\text{dia}} \) where for each \( \text{opt}^{(t)} \) there exists a \( i \ (1 \leq i \leq n) \) such that \( \text{opt}^{(t)} = C_i^{(t)}(0) \) and for all \( j \in \{1, \ldots, n\} \setminus \{i\} \) it holds that \( S_i^{(t)} f_i(C_i^{(t)}(0)) > S_j^{(t)} f_j(C_j^{(t)}(0)) \).

First, the influence of the problem properties on the general severity measures is analyzed. The general severity is determined by the transformations in the coordinate space only. As long as there are only linear translations in a homogeneous problem from one generation to the next, the average severity equals both the minimal and the maximal severity for each component function. In an inhomogeneous problem the differences are determined by the differences in component functions. As soon as a rotating transformation is introduced the minimal and the maximal severity differ within one component function since the minimal severity can be zero (in case of rotation only) and the maximal severity can be rather large depending on the rotation angle and the size of the search space.

Nevertheless the severity of the optimum is not only determined by the general severity but also by the property “alternating” of the fitness rescalings. If a problem is alternating at a time step \( t \) the optimum is changing from one component function to another component function, i.e. the severity is not predictable from the coordinate transformations only. Note that the
alternating property is strongly correlated to the stationarity and homogeneity
\((\text{stat}_s \lor \text{homo}_s \Rightarrow \neg \text{alternate})\)

The following problem classes are identified for complete problems as well as
periods of problems. The ordering does not imply a strict increase in problem
difficulty.

**Class 0** (both parts stationary): static optimization task (severity 0)

**Class 1** (constant and homogeneous coordinate translations, not alternating):
pure tracking task with constant severity, i.e. the static fitness landscape is
moving as a whole

**Class 2** (constant and inhomogeneous coordinate translations, not alternating):
pure tracking task with constant severity but in a changing environment

**Class 3** (inconstant coordinate translations, no jumping): either not so easily
predictable tracking task or rather chaotic or random problem

**Class 4** (rotating coordinate translations, not alternating): tracking task with
different inherent degrees of severity

**Class 5** (alternating, stationary coordinate translations): oscillation between
several static optima, the severity depends on the position of the component
functions' optima

**Class 6** (alternating, non-stationary coordinate translation): tracking and
determination of moving and oscillating optima, the severity depends on the
component functions' dynamics

For each problem class different optimization techniques are useful. In the
next section the interdependencies are analyzed between the population size
and some memorizing rsp. diversity preserving techniques. The impact of these
results on the optimization of the problem classes is discussed at the end of the
next section.

5 Analysis of Techniques and Population Size

This section analyzes one generation of an optimizer in a dynamic environment.
In particular, the influence of the population size on the probability to get close
to the next position of the optimum is of interest.

The analysis uses an abstract problem as well as an abstract optimizer. The
problem is only considered as far as the problem properties above are concerned.
For the algorithm only some underlying working principles are assumed instead
of analyzing a concrete instance of evolutionary computation.

The search space is a two-dimensional raster \(1000 \times 1000\). We assume the
current optimum to be placed close to the center such that no newly created
individual by a local mutation lies outside the search space. Furthermore we
consider a distance metric which is defined by vertical and horizontal crossings
of raster boundaries (cf. Figure 1). Simplifying it is assumed that the optimum
is moved horizontally.

In the analysis the following numbers are of interest. \(4i\) points have exactly
distance \(i\) to a given point and the number of points which are at most \(i\) steps
off is \(2i(i + 1) + 1\).
First, a pure mutation based local search algorithm is analyzed. It is assumed that the algorithm performs a mutation with step size $1 \leq i \leq 20$ with a probability according to the binomial distribution.

$$
Pr[i] = \frac{1}{240-1} \binom{41}{20 + i}
$$

(1)

The maximal step size is 20. Moreover, each mutation reaches a new point in the search space ($Pr[0] = 0$). Given that the parent individual is situated at last generations optimum which moves by severity $s$, the number of points within a distance $d$ from the new optimum and an exact distance $s - d < d' \leq s + d$ equals

$$
N_{d,s}(d') = 2d + 1 - 2 \left[ \frac{s + d + 1 - d'}{2} \right]
$$

(cf. left part of Figure 1). Then, the probability to hit a point within distance $d$ from the new optimum by creating one offspring equals

$$
Pr[< d(s)] = \sum_{s-d \leq d' \leq s+d} N_{d,s}(d')Pr[d']
$$

Note, that no recombination is considered in the scope of this work. Assumed that the algorithm uses a $(1, \lambda)$ strategy, the probability that at least one offspring is within a distance $d$ of the optimum results as

$$
Pr[< d(s) \text{ of } \lambda] = 1 - (1 - Pr[< d(s)])^\lambda
$$

The results in Figure 1 (right) show the dependence of the tracking probability of the population size. With increasing severity of the optimum a large
population size becomes more important. This result is especially of interest for pure tracking tasks like in class 1 and class 2 problems. It gives a guideline for choosing an appropriate population size. In some algorithms with local mutation an adaptation of the step size is used (e.g. in evolution strategies). However, this is not considered in this analysis. Note, that the results do not scale with increasing step size since the basic probability for hitting a point in the search space decreases. For problems of class 3 or 4 the determination of the population size is difficult since the severity is varying. For the same reason (self-)adaptive techniques must be questioned critically for class 3 and 4.

Second, an external memory is added to the algorithm in which certain good solutions from previous generations are stored. A memory size of 40 individuals is assumed. In order to analyze the behavior with memory two cases are distinguished: the successful case where the optimum (or a point close to the optimum) is contained in the memory and the unsuccessful case. Furthermore, two replacement strategies are considered. On the one hand, each generation 5 individuals are chosen randomly from the memory and inserted into the off-
spring population and, on the other hand, 10% of the offspring population are chosen from the memory. Now the probability from above can be modified for the successful case with the replacement number $k \in \{5, \lfloor \frac{\lambda}{10} \rfloor \}$

$$\Pr[\sim d(s) \text{ of } \lambda(\text{memory})] = 1 - (1 - \Pr[\sim d(s)])^{\lambda-k} \left(\frac{39}{40}\right)^k$$

and for the unsuccessful case

$$\Pr[\sim d(s) \text{ of } \lambda(\text{memory})] = 1 - (1 - \Pr[\sim d(s)])^{\lambda-k}.$$

Figure 2 shows the results for the algorithm with memory. Obviously with low severity the tracking probability worsens even in the successful cases with small population sizes. But already with severity 5 the algorithm replacing 10% of the population shows small advantages. Nevertheless, the worsening in case of memory without the optimum are still bigger as the gains in the successful case. If only small severity values occur, memory seems only to be useful for problems of class 5 since there is a high chance that the optimum is contained in the memory. However, as soon as bigger severity values occur the diagrams show almost negligible worsenings in the unsuccessful case and significant improvements in the successful case. As a consequence memory is always useful in problems of classes 5 and 6 with high severity even if the memory of problems in class 6 might have a rather small chance of containing the optimum.

Eventually, the method of random immigrants is analyzed in this theoretical setup on its direct influence in finding the optimum or getting close to the optimum. We consider again two scenarios: on the one hand 5 random individuals are injected and on the other hand 20% of the population are replaced randomly. The tracking probability can be derived similarly to the previous case where the basic probability depends on the size of the search space and the minimal allowed distance to the optimum. The results are shown in Figure 3. As expected
this method shows a significant drawback with low severity and small expected
distance to the optimum. Nevertheless, with high severity and a large allowed
distance to the optimum this method can help to increase the probability con­
siderably. This shows that the method is indeed able to find the correct region
with a certain probability. If we still assume in the memory model that only
one individual out of 40 is a successful individual, the random immigrants are
able to beat the memory model in the successful case. The assumption on the
memory is not unlikely since the tracking region covers only approx. 2% of the
search space. This result can explain the good performance of the mechanism in
certain problems – usually with recombination which helps to combine several
individuals getting anywhere close to the optimum. With respect to the problem
classes, this method should be considered for class 6 with a rather high severity
since it has a guaranteed improvement in contrary to the memory approach.

6 Conclusion

This work presents a framework for the classification and comparison of dynamic
problems. This framework is used for an analysis how the offspring population
size and two special techniques for dynamic problems affect the tracking prob­
ability of a search algorithm based on a local mutation operator. Within the
context of a (1, λ)-strategy, the analysis gives concrete information for which
problem class which algorithmic technique and which population size should be
at least considered. Since especially in dynamic problems the populations size
can be a critical parameter – more evaluations can impose higher dynamics –
more investigations are necessary in that direction.

Besides the concrete results in this work, the framework for dynamic problems
enables more theoretical investigations of correlations between problem proper­
ties, parameter settings, algorithmic techniques, and adaptation measures. In
particular, future work will consider more complex models of algorithms in­
corporating recombination operators and other performance measures covering
different aspects of adaptation.

References

Branke, J. (1999a). Evolutionary approaches to dynamic optimization problems: A
survey. In J. Branke & T. Bäck (Eds.), Evolutionary algorithms for dynamic
optimization problems (pp. 134–137). (part of GECCO Workshops, A. Wu
(ed.))

Branke, J. (1999b). Memory enhanced evolutionary algorithms for changing optimiza­
tion problems. In 1999 Congress on Evolutionary Computation (pp. 1875–1882).

operator in genetic algorithms having continuous, time-dependent nonstationary


Abstract

Permutations can represent search problems when all points in the search space have unique evaluations. Given a particular set of N evaluations we have N! search algorithms and N! possible functions. A general No Free Lunch result holds for this finite set of N! functions. Furthermore, it is proven that the average description length over the set of N! functions must be O(N lg N). Thus if the size of the search space is exponentially large with respect to a parameter set which specifies a point in the search space, then the description length of the set of N! functions must also be exponential on average. Summary statistics are identical for all instances of the set of N! functions, including mean, variance, skew and other r-moment statistics. These summary statistics can be used to show that any set of N! functions must obey a set of identical constraints which holds over the set of Walsh coefficients. This also imposes mild constraints on schema information for the set of N! functions. When \( N = 2^L \) subsets of the N! functions are related via Gray codes which partition N! into equivalence classes of size \( 2^L \).

1 Functions as Permutations

Consider a set composed of \( N \) unique values; these \( N \) values can be mapped to a set \( V \) of evaluations for some objective function. Let \( X \) be the set of points and \( V \) the set of evaluations; we then define a one-to-one objective function \( f \):

\[
f(x) = v, \quad v \in V, x \in X.
\]

Construct a set \( \Pi \) of all permutation over the values in \( V \). In this case, the set \( \Pi \) represents all objective functions which can be constructed over \( V \). We
will also use $\Pi_1$ to represent all search algorithms which can be applied to the set of evaluation functions which can be constructed over $V$. We will represent an algorithm by the order in which it samples the values in $V$. If different search procedures enumerate the points in $V$ in the same order, then they form part of an equivalence class. Resampling of points is ignored. Hence, in this context, algorithms as well as functions are permutations. It is easy to show that a specialization of the "No Free Lunch" theorem result holds [14] [9] [2]. On average, no algorithm is better than random enumeration in locating the global optimum. If algorithms are executed for only $m$ steps, every algorithm find the same set of best so-far solutions over all functions.

Focusing attention on a well defined set of permutations of finite length, allows one to make more detailed comments about the No Free Lunch result as it pertains to this set.

**Observation:** The set of $N!$ functions corresponding to $\Pi_1$ have a description length of $O(N \lg N)$ bits on average, where $N$ is the number of points in the search space.

**Proof:** The proof follows the well known proof demonstrating that the best sorting algorithms have complexity $O(N \lg N)$. First, since we have $N!$ functions, we would like to "tag" each function with a bit string which uniquely identifies that function. We then make each of these tags a leaf in a binary tree. The "tag" acts as an address which tells us to go left or right at each point in the tree in order to reach a leaf node corresponding to that function. But the "tag" also uniquely identifies the function. The tree is constructed in a balanced fashion so that the height of the tree corresponds to the number of bits needed to tag each function. Since there are $N!$ leaves in the tree, the height of the tree must be $O(\lg(N!)) = O(N \lg N)$. Thus $O(N \lg N)$ bits are required to uniquely label each function. **QED.**

Note that the number of bits required to construct a full enumeration of any permutation of $N$ elements is also $O(N \lg N)$ bits, since there are $N$ elements and $\lg N$ bits are needed to distinguish each element. Thus, most of these functions have exponential description. To be NP-Complete, the description length must be polynomial. This means that an NP-Complete problem class *cannot* be used to generate all $N!$ functions. This includes NK-Landscapes [8] and MAXSAT, for example.

This is, of course, one of the major concerns about No Free Lunch results. Do "No Free Lunch" results really apply to sets of functions which are of practical interest? Yet this same concern is often overlooked when theoretical researchers wish to make mathematical observations about search. For example, proofs relating the number of expected optima over all possible functions [11], or the expected path length to a local optimum over all possible functions [13] under local search are computed with respect to the set of $N!$ functions.

### 1.1 Walsh Analysis

It is next shown that the set of Walsh coefficients are constrained with respect to the set of all possible members of the set of $N!$ functions. From this it
follows that summary statistics such as variance, skew, kurtosis are not useful for guiding search.

Every real-valued function \( f \) over an \( L \)-bit string can be expressed as a weighted sum of a set of \( 2^L \) orthogonal functions called \textbf{Walsh functions}.

\[
f(x) = \sum_{j=0}^{2^L-1} w_j \psi_j(x)
\]

where the Walsh functions are denoted \( \psi_j : B^L \rightarrow \{-1, 1\} \). The weights \( w_j \in \mathcal{R} \) are called \textbf{Walsh coefficients}.

The indices of both Walsh functions and coefficients may be expressed as binary strings or the equivalent integer. Treating the indices \( x, j \) as binary vectors, we can compute the Walsh function as follows:

\[
\psi_j(x) = (-1)^{j^T x}.
\]

The \( 2^L \) Walsh coefficients can likewise be computed by a Walsh transform:

\[
w_j = \frac{1}{2^L} \sum_{i=0}^{2^L-1} f(i) \psi_j(i)
\]

The calculation of Walsh coefficients can be thought of in terms of matrix multiplication. Let \( \vec{f} \) be a column vector of \( 2^L \) elements where the \( i^{th} \) element is the evaluation of function \( f(i) \). Similarly define a column vector \( \vec{w} \) for the Walsh coefficients. If \( M \) is a \( 2^L \times 2^L \) matrix where \( M_{i,j} = \psi_j(i) \), then:

\[
\vec{w} = \frac{1}{2^L} \vec{f}^T M
\]

For example, if we have a 3 bit function with the \( 2^3 \) function evaluations labeled \( f_0..f_7 \), then the Walsh coefficient calculation would be:

\[
\vec{w} = \frac{1}{8} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix}
\]

An important property of Walsh coefficients is that \( w_j \) measures the contribution to the evaluation function by the interaction of the \( 1 \)'s in \( j \). We number the bit positions from right to left, starting at 0. Thus, \( w_{00001} \) measures the linear contribution to the evaluation function associated with bit position 0, while \( w_{0101} \) measures the nonlinear interaction between the bits in positions 0 and 2. This nonlinearity is often considered to be an important feature in determining problem difficulty for stochastic search algorithms [3, 4, 12].
2 Summary Statistics for Problem Instances

Walsh analysis can be used to compute summary statistics for fitness distributions of discrete optimization problems. Note that the fitness distribution is the distribution formed by evaluating all possible inputs to a problem. So, for a problem defined over $2^L$ possible inputs, the distribution would be comprised of all $2^L$ evaluations of the inputs. Goldberg and Rudnick [5] have used Walsh coefficients to calculate fitness variance for fitness distributions and schemata. Heckendorn, Rana and Whitley [6] show how higher order statistics such as skew and kurtosis can be also be computed from the Walsh coefficients using a general formula for computing the $r^{th}$ moment for any fitness distribution; they also show that for certain special types of functions, the Walsh coefficients and summary statistics can be computed in polynomial time.

Since $\psi_0(x) = 1$ for all inputs, the $w_0$ coefficient is the mean of all fitnesses. Given the mean, the formula used to compute the $r^{th}$ moment for a discrete random variable $X$ is:

$$\mu_r = E[(X - \mu)^r] = \sum_{x \in X} (x - \mu)^r p(x)$$

For our purposes, the function $p(x) = \frac{1}{2^L}$ since we are enumerating a function over binary strings and each point occurs $\frac{1}{2^L}$ times. The function then becomes:

$$\mu_r = \sum_{x \in X} \frac{(x - \mu)^r}{2^L}$$

The $r^{th}$ moment over the distribution of fitness for all $2^L$ possible input strings is:

$$\mu_r = \frac{1}{2^L} \sum_{x = 0}^{2^L - 1} (f(x) - \mu)^r$$

Since $f(x) = \sum_{i=0}^{2^L-1} w_i \psi_i(x)$,

$$\mu_r = \frac{1}{2^L} \sum_{x = 0}^{2^L - 1} \left( \sum_{i=0}^{2^L-1} w_i \psi_i(x) - \mu \right)^r$$

Since $\mu = w_0$, and $\psi_0(x) = 1 \ \forall x$:

$$\mu_r = \frac{1}{2^L} \sum_{x = 0}^{2^L - 1} \left( \sum_{i=1}^{2^L-1} w_i \psi_i(x) \right)^r$$

Now create a set of $r$ indices $a_j$ for $1 \leq j \leq r$, and expand the formula:

$$\mu_r = \frac{1}{2^L} \sum_{x = 0}^{2^L - 1} \left( \sum_{a_1 = 1}^{2^L-1} w_{a_1} \psi_{a_1}(x) \right) \left( \sum_{a_2 = 1}^{2^L-1} w_{a_2} \psi_{a_2}(x) \right) \cdots \left( \sum_{a_r = 1}^{2^L-1} w_{a_r} \psi_{a_r}(x) \right)$$
By simple algebra these equations simplify to the following:

$$\mu_r = \sum_{a_1 \oplus a_2 \oplus \ldots \oplus a_r = 0} w_{a_1} w_{a_2} \ldots w_{a_r}, \quad a_i \neq 0$$

(2)

To summarize this formula, given the set of nonzero Walsh coefficients, we can compute the $r^{th}$ moment for the fitness distribution using products of the Walsh coefficients such that the exclusive-or of the indices is zero.

This formula allows us to compute the variance, skew and kurtosis for any fitness distribution provided we are given the Walsh coefficients.

$$\text{variance} = \mu_2 = \sigma^2 \quad \text{skew} = \frac{\mu_3}{\sigma^3} \quad \text{kurtosis} = \frac{\mu_4}{\sigma^4}$$

For example, since $a_1 \oplus a_2 = 0$ if and only if $a_1 = a_2$ the variance for any function is given by

$$\sum_{i=1}^{2^L-1} w_i w_i$$

Heckendorn, Rana and Whitley [6] show that in the special case of Embedded Landscapes [7] there are only a polynomial number of nonzero Walsh coefficients and we can identify and compute them in polynomial time. MAX3SAT and NK-Landscapes are instances of an Embedded Landscape. Thus, an exact Walsh analysis can be done in polynomial time. And as a result all the summary statistics can be computed exactly from equation 2 for all of these classes of problems in polynomial time.

It at first seems surprising that one can characterize NP-Complete problems in such statistical detail. Since we can exactly compute these summary statistics about NP-Complete problems in polynomial time, is there any useful information which can be extracted from these summary statistics to guide search? We have previously noted [6] that if $P \neq NP$, then in the general case the $r^{th}$ moment summary statistics cannot provide any information which can be used to guide any search algorithm to an optimal solution, or even to an above average solution in the search space. The current paper decouples the usefulness of these summary statistics from the question of $P$ vs $NP$.

2.1 On Summary Statistics and Search

We again consider the set $\Pi$ of all permutations over the set of values $V$.

**Observation:** The set of summary statistics, such as variance, skew and other higher moment summary statistics, provide no useful information for selecting one search algorithm over another, or for guiding any particular search algorithm, when searching a specific objective function constructed from $V$.

**Proof:** The set of summary statistics are identical for the set $V$. Since all permutations, $\Pi$, are constructions over $V$, and these permutations describe all possible functions over $V$, No Free Lunch holds over $\Pi$. This implies that summary statistics cannot be used to guide search, since the summary statistics are identical over a set for which a No Free Lunch result holds. \textbf{QED.}
This also places interesting restrictions on the information which can be extracted from Walsh coefficients about objective functions constructed from \( V \), since the values of the Walsh coefficients themselves are directly constrained by the summary statistics.

### 2.2 Statistics, Neighborhood and Walsh Analysis

We next look at the relationship between summary statistics, neighborhood structure and Walsh analysis.

A Gray code is a wrapped construction which forms a circuit such that all adjacent integers are Hamming Distance 1 neighbors and endpoints are also neighbors. This circuit can be “shifted” such that any value can be assigned to the string of all zero bits; the result is still a Gray code. Normally, when bit strings are “DeGrayed” they are converted to binary strings, then converted to integers, and finally mapped to real space. “Shifting” [10] can be applied after the bits have been converted to integers, but before the integers are mapped to real values. Any integer constant between 0 and \( 2^L - 1 \) \((\text{mod} \ 2^L)\) can be added to the decoded integer generated from the bit string. We may view “shifting” as a change in representation for a single function. But it also changes the permutation in \( \Pi \) and thus also results in a new function. Thus, when \( N = 2^L \) then \( 2^L \) of the \( N! \) functions form an equivalence class; one function can be transformed into another via a change in representation using a shifted Gray Code: this equivalence class corresponds to different Gray codes of the same real valued function. When the real-valued space is modeled as a 1-dimensional circuit, the real valued space is unchanged under this change in representation.

We have recently shown that repeating neighborhood structures exists under shifting. Since a reflected Gray code is a symmetric reflection, flipping the leading bit of each string does not change the Hamming distance-1 neighborhood. This corresponds to a shifts by \( 2^{L-1} \). This means that the neighborhood structure repeats.

**Theorem:** In any reflected Gray code representation of a 1-dimensional function, or parameter of a multidimensional function, there are only \( 2^{L-2} \) unique shifts.

For a proof see Barbulescu, Watson and Whitley [1]. The neighborhood structure repeats every quartile of the parameter space under shifting. For example, in a 4-bit neighborhood defined over 16 points, there are only 4 unique Hamming distance-1 neighborhoods.

Let \( \pi \in \Pi \) be that permutation where the values from \( V \) are in sorted order from small to large. To compute the Walsh coefficients for any other function in the set \( \Pi \), we can apply a reordering permutation to either \( \pi \) to generate the new function/permutation, or we can apply the same reordering permutation to the rows of the Walsh Transform matrix. Note that under matrix multiplication, there is no difference between permuting the function/permutation vector and permuting the rows of the matrix. Let \( \pi' \) be the permuted function (in column vector form) and \( M' \) the corresponding matrix where the permutation \( \pi' \) is used to reorder the rows of \( M \).
Observation: \( \bar{w} = \left[ \frac{1}{2^L}(\pi')^T M \right] = \left[ \frac{1}{2^L}\pi^TM' \right] \)

We can also view the Walsh Matrix as a recursively defined structure. A Walsh Transform of dimension \( 2^d \) can be expressed as a composition of four \( 2^{d-1} \) dimensional transforms. The following notation denotes a \( 2^{d-i} \) transform as \( W_i \). In addition, because we wish to manipulate the rows of the Walsh matrix, the four "rows" of the recursively decompose Walsh matrix can be expressed as M x N matrices; these are labeled using \( W_{2A}, W_{2B}, W_{2C}, W_{2D} \) as shown:

\[
W = \begin{bmatrix}
W_1 & W_1 \\
W_1 & -W_1
\end{bmatrix} = \begin{bmatrix}
W_2 & W_2 & W_2 & W_2 \\
W_2 & -W_2 & W_2 & -W_2 \\
W_2 & W_2 & -W_2 & -W_2 \\
W_2 & -W_2 & -W_2 & W_2
\end{bmatrix} = \begin{bmatrix}
W_{2A} \\
W_{2B} \\
W_{2C} \\
W_{2D}
\end{bmatrix}
\]

Now, we can shift the rows of the \( W_2 \) form of the Walsh Matrix, and this exactly corresponds to shifting the input function by \( 2^L-2 \). However this shift is in Binary space. The shift in Gray space leaves the Hamming neighborhood structure unchanged. Under Binary, a shift of \( 2^{L-1} \) leaves the neighborhood unchanged, but an examination of any small space (e.g. 2 to 10 bits) shows that a shift of \( 2^{L-2} \) changes the neighborhood under Binary. But there are similarities between the two representations. Study of the Gray shifts shows that the decomposed Walsh matrix is reversed at various points under shifting. This is consistent with the fact that the Gray code is a reflected code. This motivates the following observation.

Let \( \equiv_n \) denote a binary relation between Walsh Matrices, such that the resulting set of Walsh coefficients corresponds to functions with an identical neighborhood structure in Hamming space. The decomposed Walsh matrix can be shifted by \( 2^{L-1} \) and the decomposed Walsh matrix can be reversed without changing the neighborhood structure. Thus,

\[
\begin{bmatrix}
W_{2A} \\
W_{2B} \\
W_{2C} \\
W_{2D}
\end{bmatrix} \equiv_n \begin{bmatrix}
W_{2C} \\
W_{2D} \\
W_{2A} \\
W_{2B}
\end{bmatrix} \equiv_n \begin{bmatrix}
W_{2B} \\
W_{2A} \\
W_{2D} \\
W_{2C}
\end{bmatrix} \equiv_n \begin{bmatrix}
W_{2D} \\
W_{2C} \\
W_{2B} \\
W_{2A}
\end{bmatrix}
\]

These reordering can be done by applying exclusive-or between the encoding an a target string and by reordering the first two bits. Neither of these operators have any impact on the neighborhood structure. There are of course other reorderings of the Walsh matrix that leave the neighborhood structure unchanged, including some shifts of the Gray representation.

2.3 Empirical Examples from Walsh Analysis

The point of the last section is that there are several different changes in representation that do not change the structure of the Hamming neighborhood. Does
Table 1: Walsh coefficients for the 16 shifts of the permutation [1 .. 16].

<table>
<thead>
<tr>
<th>Gray</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>$w_{10}$</th>
<th>$w_{11}$</th>
<th>$w_{12}$</th>
<th>$w_{13}$</th>
<th>$w_{14}$</th>
<th>$w_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>12</td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8.5</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8.5</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-3</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>9</td>
<td>8.5</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>13</td>
<td>8.5</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>8.5</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-0.5</td>
</tr>
<tr>
<td>6</td>
<td>8.5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>8.5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-0.5</td>
</tr>
<tr>
<td>3</td>
<td>8.5</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>8.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>8.5</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>15</td>
<td>8.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>$w_{10}$</th>
<th>$w_{11}$</th>
<th>$w_{12}$</th>
<th>$w_{13}$</th>
<th>$w_{14}$</th>
<th>$w_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.5</td>
<td>-5</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8.5</td>
<td>.5</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$0^R$</td>
<td>8.5</td>
<td>.5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$8^R$</td>
<td>8.5</td>
<td>-5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Let $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$ be the sorted permutation composed of the integers from 1 to 16. (This permutation is the Binary ordering. The permutation has been mapped to Gray-coded strings, then those strings must be reordered according to their Binary interpretation for the Walsh analysis to be correct.) Table 1 gives the Walsh coefficients for all possible shifts of this function under Gray code. There are really only 4 different Hamming neighborhood structures over the 16 possible shifts. Shifts at $i+0$, $i+4$, $i+8$, $i+12$ are identical for $i = 1$ to 4. There are strong similarities between the Walsh coefficients for these identical neighborhoods. Table 1 also shows the Walsh coefficients for 4 equivalent neighborhoods under Binary corresponding to equation 3.

Table 2 gives four randomly selected permutations and the corresponding Walsh coefficients. The Walsh coefficients are now more diverse. Nevertheless, the summary statistics always remain the same for all possible permutations. In this example, a simple calculation shows that the variance is 21.25 for all func-
Permutations A, B, C, D

<table>
<thead>
<tr>
<th>ID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>14</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>10</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>13</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>11</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td>11</td>
</tr>
</tbody>
</table>

Walsh Coefficients for Permutations A, B, C, D

<table>
<thead>
<tr>
<th>ID</th>
<th>w_0</th>
<th>w_1</th>
<th>w_2</th>
<th>w_3</th>
<th>w_4</th>
<th>w_5</th>
<th>w_6</th>
<th>w_7</th>
<th>w_8</th>
<th>w_9</th>
<th>w_10</th>
<th>w_11</th>
<th>w_12</th>
<th>w_13</th>
<th>w_14</th>
<th>w_15</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8.5</td>
<td>-4</td>
<td>0.75</td>
<td>-1.0</td>
<td>.875</td>
<td>.375</td>
<td>.625</td>
<td>.375</td>
<td>-1.25</td>
<td>-0.25</td>
<td>0</td>
<td>.25</td>
<td>-1.25</td>
<td>.375</td>
<td>.125</td>
<td>-.825</td>
</tr>
<tr>
<td>B</td>
<td>8.5</td>
<td>1.75</td>
<td>-1.25</td>
<td>1.0 -2.125</td>
<td>.125</td>
<td>1.125</td>
<td>.375</td>
<td>-1.0</td>
<td>0.75</td>
<td>1.5</td>
<td>0.75</td>
<td>-125-1.625</td>
<td>1.375</td>
<td>.875</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>8.5</td>
<td>0</td>
<td>1.125</td>
<td>-0.625</td>
<td>-1.75</td>
<td>-.30</td>
<td>1.375</td>
<td>-1.125</td>
<td>.375</td>
<td>.375</td>
<td>.5</td>
<td>.25</td>
<td>-1.375</td>
<td>.875</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>D</td>
<td>8.5</td>
<td>3.5</td>
<td>.125</td>
<td>.625</td>
<td>-.125</td>
<td>-1.375</td>
<td>.25</td>
<td>.5</td>
<td>-.375</td>
<td>-.125</td>
<td>0</td>
<td>-1.75</td>
<td>1.5</td>
<td>0</td>
<td>-.375</td>
<td>-.875</td>
</tr>
</tbody>
</table>

Table 2: Walsh coefficients for random permutations of the elements [1 .. 16].

Table 3: Statistical analysis of Walsh coefficients for random permutations of the elements [1 .. 16].

A discussion of the limitations and applications of Walsh coefficients for random permutations.

3 Discussion and Conclusions

In as much as Walsh analysis can be used to compute information about schema fitness information, constraints on what we can learn about a function from Walsh analysis have some impact on what we can or cannot learn about a function from schema fitness information.

Often Walsh analysis is dismissed as not being of practical interest. However, Heckendorn, Rana and Whitley [7] have shown that it is possible to do Walsh analysis in polynomial time for all functions which can be modeled as Embedded landscapes. Embedded landscapes include not only NP-Complete functions such as MAX3SAT and NK-Landscapes, but also common separable test functions such as Rastrigins functions. Any function that can be decomposed into a polynomial number of nonlinear interactions where each decomposed subfunction is of polynomial size is an Embedded landscape. Heckendorn, Rana and Whitley [6] also show that for all Embedded landscapes it is possible to compute summary statistics in polynomial time.

This paper specifically looks at functions as permutations. It is observed that the No Free Lunch results hold for this finite set, that the average description length for this finite set of problems must be exponential on average, and that the summary statistics must be identical for all members of this set. This places constraints on the set of values which can be possible Walsh coefficients. More concretely, it is shown that summary statistics cannot be used to guide search, since No Free Lunch results holds over a set for which the summary statistics are identical. Also different representations of the same function can have identical neighborhood structure; this also appears to be reflected in the similarity of the Walsh coefficients. But more work is needed to fully understand this relationship.
ACKNOWLEDGEMENT: The observation that summary statistics are identical under shifting of Gray codes was made by Soraya Rana.

References


Genetic Programming
Distributed Hybrid Genetic Programming
for Learning Boolean Functions

Stefan Droste*, Dominic Heutelbeck*, and Ingo Wegener*

FB Informatik, LS 2, Univ. Dortmund, 44221 Dortmund, Germany
{droste,heutelbe,wegener}@ls2.cs.uni-dortmund.de

Abstract. When genetic programming (GP) is used to find programs with Boolean inputs and outputs, ordered binary decision diagrams (OBDDs) are often used successfully. In all known OBDD-based GP-systems the variable ordering, a crucial factor for the size of OBDDs, is preset to an optimal ordering of the known test function. Certainly this cannot be done in practical applications, where the function to learn and hence its optimal variable ordering are unknown.

Here, the first GP-system is presented that evolves the variable ordering of the OBDDs and the OBDDs itself by using a distributed hybrid approach. For the experiments presented the unavoidable size increase compared to the optimal variable ordering is quite small. Hence, this approach is a big step towards learning well-generalizing Boolean functions.

1 Introduction

A major goal in genetic programming (GP) ([8]) is to find programs that reproduce a set of given training examples and have good generalizing properties. This means that the resulting program should closely resemble the output values of the underlying function for the inputs not included in the training set, too.

One approach to achieve this is the principle of Occam's Razor, i.e., one tries to find the simplest function that outputs the correct values for the training set. Here one assumes, that functions with small representations yield a better generalization. Using a result from learning theory it was shown in [6] how the generalization quality of small programs found by GP can be lower bounded. Because redundant code can make it very difficult to measure the size of S-expressions, ordered binary decision diagrams (OBDDs) were used in this approach having an easy to compute minimal representation, called reduced.

OBDDs, introduced in [3], have proved to be the state-of-the-art data structure for Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$, i.e., $f \in B_n$: on the one hand, they allow the representation of many important Boolean functions in size polynomial in $n$, on the other hand, many algorithms with polynomial runtime in the size of the OBDDs are known for manipulating OBDDs (see [14]).

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).
Because of these advantages, OBDDs have been used successfully in GP-systems in [15], [5], and [7]. But all these systems do not only base their runs on the given training set, but also on a known optimal variable ordering for the given benchmark function. The variable ordering has a crucial influence on the size of an OBDD, i.e., depending on it a function can have a polynomially- or exponentially-sized OBDD. Because in all former approaches only known test functions were used, the optimal variable ordering was known in advance, which is naturally not the case in practical applications. Because there are important functions like the multiplexer-function, where only a very small fraction of all variable orderings allows even a good approximation with polynomially sized OBDDs, it is necessary to adapt the variable ordering.

Here, we present the first GP-system, where the variable ordering and the OBDDs itself are evolved using a distributed hybrid approach. In the next two sections we formally define OBDDs and discuss the variable ordering problem and its implications for OBDD-based GP. Then we describe a new GP-system that uses well-known heuristics for the variable ordering problem and methods from distributed evolutionary algorithms. Finally, we present some empirical results, showing that the unavoidable loss in quality with respect to a system using the optimal variable ordering is quite small.

2 Ordered binary decision diagrams

Definition 1. Let \( \pi \) be a permutation on \( \{1, \ldots, n\} \) (called variable ordering). A \( \pi \)-OBDD is a directed acyclic graph \( O = (V, E) \) with one source and two sinks, labelled by the Boolean constants 0 and 1. Every inner node is labelled by one of the Boolean variables \( x_1, \ldots, x_n \) and has two outgoing edges leading to the 0- and 1-successor. If an edge leads from an \( x_i \)-node to an \( x_j \)-node, then \( \pi^{-1}(i) \) has to be smaller than \( \pi^{-1}(j) \), i.e., the edges have to respect the variable ordering.

In order to evaluate the function \( f \) represented by a \( \pi \)-OBDD for an input \( (a_1, \ldots, a_n) \in \{0, 1\}^n \), one starts at the source and recursively goes to the 0- resp. 1-successor, if the actual node is labelled by \( x_i \) and \( a_i = 0 \) resp. \( a_i = 1 \). Then \( f(a) \) is equal to the label of the finally reached sink.

The size of \( O \) is the number of its inner nodes. An OBDD is a \( \pi \)-OBDD for an arbitrary \( \pi \). An OBDD is reduced, if it has no node with identical 0- and 1-successor and contains no isomorphic subgraphs.

One can prove, that for a given \( f \in B_n \) and a fixed \( \pi \) the reduced \( \pi \)-OBDD of \( f \) is unique up to isomorphism. Let \( \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\} \) be a set of indices and \( a \in \{0, 1\}^k \). The function \( f_{|x_{i_1}=a_1, \ldots, x_{i_k}=a_k} : \{0, 1\}^{n-k} \rightarrow \{0, 1\} \) is the restriction of \( f \), where for every \( j \in \{1, \ldots, k\} \) the variable \( x_{i_j} \) is set to \( a_j \). A function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) depends essentially on \( x_i \), iff \( f_{|x_i=0} \neq f_{|x_i=1} \). One can show that a reduced \( \pi \)-OBDD representing \( f \in B_n \) contains exactly as many nodes with label \( x_i \), as there are different subfunctions \( f_{x_{\pi(1)}=a_1, \ldots, x_{\pi(j-1)}=a_{j-1}} \) depending essentially on \( x_i \), for all \( (a_1, \ldots, a_{j-1}) \in \{0, 1\}^{j-1} \) with \( j = \pi^{-1}(i) \), i.e., the size of a reduced \( \pi \)-OBDD is directly related to the structure of the function it represents.
Hence, the representation of Boolean functions by reduced OBDDs eliminates redundant code and automatically discovers useful subfunctions in a similar manner as automatically defined functions ([8]), as parts of the OBDD can be evaluated for different inputs. One receives these benefits without further expense by the application of reduced OBDDs for representation in a GP-system.

The problem we want to solve at least approximately is the following:

**Definition 2.** In the minimum consistent OBDD problem we have as input a training set \( T \subseteq \{(x, f(x)) \mid x \in \{0,1\}^n\} \) and want to compute the minimal \( \pi \)-OBDD, that outputs \( f(x) \) for all \( x \) with \( (x, f(x)) \in T \), over all variable orderings \( \pi \), where \( f: \{0,1\}^n \mapsto \{0,1\} \) is the underlying function.

Certainly our main goal is to find the function \( f \), but all we know about it is the training set \( T \). Assuming that the principle of Occam’s razor is valid for the functions we want to find, a solution to the minimum consistent OBDD problem would be a well generalizing function. In the next section we argue why the variable ordering is essential for the minimum consistent OBDD problem.

### 3 The variable ordering problem and OBDD-based GP

It is well-known, that the size of a \( \pi \)-OBDD depends heavily on \( \pi \). A good example for this fact is the multiplexer-function, one of the major benchmark functions for GP-systems that try to learn Boolean functions:

**Definition 3.** The multiplexer function on \( n = k+2^k \) \((k \in \mathbb{N})\) Boolean variables is the function \( MUX_n(a_0, \ldots, a_{k-1}, d_0, \ldots, d_{2^k-1}) = d_{|a|} \), where \( |a| \) is the number whose binary representation is \( (a_0, \ldots, a_{k-1}) \).

If \( \pi \) orders the variables as \( a_0, \ldots, a_{k-1}, d_0, \ldots, d_{2^k-1} \), the OBDD for \( MUX_n \) has size \( 2^k - 1 + 2^k \), i.e., linear in \( n \). But for the reverse order the OBDD for \( MUX_n \) has size at least \( 2^{2^k} - 1 = \Omega(2^n) \), i.e., exponential in \( n \). For an example see Figure 1. Furthermore, \( MUX_n \) is almost ugly, i.e., the fraction of variable orderings leading to non-polynomially-sized OBDDs converges to 1 ([14]). So randomly choosing \( \pi \) will lead to non-polynomial \( \pi \)-OBDD size with high probability for large \( n \).

Trying to exactly compute an optimal variable ordering is not a choice, since the computation of an optimal variable ordering is NP-complete ([2]) and even finding a variable ordering \( \pi \), such that the size of the resulting \( \pi \)-OBDD approximates the optimal size over all variable orderings up to a constant factor, cannot be done in polynomial-time, if \( NP \neq P \) ([12])

Considering a GP system that searches for small OBDDs fitting a random set of training examples for \( MUX_n \), [9] provide the following theorem:

**Theorem 1.** For every large enough \( n = k+2^k \), if we choose \( m = k^{O(1)} \) training examples of \( MUX_n \) under the uniform distribution and choose a random ordering \( \pi \) of the variables, then with probability at least \( 1 - k^{-1/2} \), there is no \( \pi \)-OBDD of size \( \frac{1}{10} m/\log m \) matching the given training examples.
This theorem implies, that if we start a OBDD-based GP run even with a random variable ordering with high probability it is impossible to find a small OBDD matching the training examples. In contradiction to the early OBDD-based GP-systems we consider the knowledge of the optimal variable ordering as unknown, since the cited results show that it is hard to obtain, even if the complete function is known. Hence, we have to optimize the variable ordering during the GP run. A possibility to do this is presented in the next section.

4 A distributed hybrid GP-system

Because one has to optimize the variable ordering during the GP run to approximately solve the minimum consistent OBDD problem, we have to decide how to do this in our GP-system. The usual GP approach would be to add independent variable orderings to each individual and to tailor the genetic operators to this new representation. But since this approach would imply the reordering of OBDDs in almost every single genetic operation, this would lead to inefficient genetic operators with exponential worst-case run-time. For OBDDs of the same variable ordering we know efficient genetic operators ([5]). Hence, we should try to do as few reorderings as possible without losing too much genetic diversity.

Therefore, we use a distributed approach similar to the distributed GA from [13]. In our approach all OBDDs in a subpopulation have the same variable ordering, but the subpopulations can have different ones. This fact allows us to use efficient genetic operators in the subpopulations. When migration between

---

**Fig. 1.** Two OBDDs representing the function MUX₆, where the edges to 0-successors are dotted and complemented edges are marked by a *. a) With an optimal variable ordering. b) With a bad variable ordering.
the subpopulations occurs every $M$-th generation, the migration strategy decides how to choose the new variable ordering of each subpopulation. In order to exchange good individuals as fast as possible we use a completely connected topology, i.e., every subpopulation sends migrants to every other subpopulation.

Because this alone would limit the number of variable orderings to the number of subpopulations, every $N$-th generation we use a heuristic to optimize the variable ordering in each subpopulation separately. The following heuristics are suitable for our setting: sifting ([11]), group sifting ([10]), simulated annealing ([1]), and genetic algorithms ([4]).

In order to exactly describe our algorithm, we make the following definitions:

**Definition 4.**

a) Let $\mu \in \mathbb{N}$ be the population size and $\lambda \in \mathbb{N}$ the number of offspring.

b) Let $I = \{i_1, \ldots, i_k\}$ be a multi-set of $k$ subpopulations, where $i_j = \{O_j^1, \ldots, O_j^\mu\}$ is a multi-set with $\pi_j$-OBDDs $O_j^l$ for all $j \in \{1, \ldots, k\}$ and $l \in \{1, \ldots, k\}$.

c) Let $B \in \mathbb{N}$ be the migration rate, $M \in \mathbb{N}$ the length of the migration interval, and $N \in \mathbb{N}$ the length of the reordering interval.

d) Let $I_{n_j} \ (1 \leq j \leq k)$ be the lists of incoming OBDDs.

Then the rough outline of our GP-system is as follows, where all sets of individuals are multi-sets, i.e., we allow duplicates:

**Algorithm 1** Distributed Hybrid GP-system

**Input:** the training set $T$.

1. **Initialization:** Choose a uniformly distributed variable ordering $\pi_j$ and a random initial population $i_j = \{O_j^1, \ldots, O_j^\mu\}$ for all $j \in \{1, \ldots, k\}$. Set $g = 0$.

2. **Reordering:** If $g \mod N = 0$: Execute the chosen variable ordering optimization heuristic on $i_j$ for every $j \in \{1, \ldots, k\}$.

3. **Generation of offspring:** For every $j \in \{1, \ldots, k\}$ generate $\lambda$ offspring $O_j^{\mu+1}, \ldots, O_j^{\mu+\lambda}$ from $i_j$ by doing mutation resp. recombination with probability $p$ resp. $1 - p$.

4. **Selection:** For every $j \in \{1, \ldots, k\}$ let $i_j$ be the $\mu$ individuals with the highest fitness values from $O_j^1, \ldots, O_j^\mu$.

5. **Selection of migrants:** If $g \mod M = 0$: For every $j \in \{1, \ldots, k\}$ set $I_{n_j} = \emptyset$. For every $j \in \{1, \ldots, k\}$ and $j' \neq j$ choose a set of $B$ individuals $a = \{a_1, \ldots, a_B\}$ fitness proportionally from $i_j = \{O_j^1, \ldots, O_j^\mu\}$ and set $I_{n_j}(i_{j'}) = I_{n_j}(i_{j'}) \cup a$.

6. **Calculate new variable ordering:** If $g \mod M = 0$: For every $j \in \{1, \ldots, k\}$ let $\pi_j = \text{migration\_strategy}(j)$.

7. **Migration:** If $g \mod M = 0$: For every $j \in \{1, \ldots, k\}$ let $I_{n_j} = \{a_1, \ldots, a_\nu\}$ for $\nu = B \cdot (k - 1)$, delete $\nu$ randomly under the uniform distribution chosen individuals in $i_j$ and insert the $\pi_j$-OBDD $O_{\mu+\lambda+1}$ with $f_{O_{\mu+\lambda+1}} = f_{a_l}$ for every $l \in \{1, \ldots, \nu\}$ into $i_j$.

8. **Main loop:** Set $g = g + 1$ and go to step 2, until $g > G$.

9. **Output:** Output the smallest consistent OBDD of the last generation.

Now we describe the different steps of our GP-system in more detail. Because initialization and recombination are based on [5], more details can be found there.
4.1 Representation

In the j-th subpopulation all individuals are represented by reduced $\pi_j$-OBDDs. Because they all share the same variable ordering, we use a reduced shared binary decision diagram (SBDD) for every subpopulation. An SBDD representing

$$g: \{0,1\}^n \rightarrow \{0,1\}^m$$

is an OBDD with m sources representing the coordinate functions $g_i$ for $g = (g_1, \ldots, g_m)$. In OBDD-based GP the population of m reduced $\pi$-OBDDs representing functions $g_1, \ldots, g_m$ from $B_n$ is identified with the multi-output function $g = (g_1, \ldots, g_m)$ and stored in the reduced $\pi$-SBDD representing $g$. By doing this, the individuals share isomorphic sub-graphs. Experiments have shown, that in an OBDD-based GP-system the SBDD size will go down fast in comparison to the sum of the sizes of the OBDDs, because the individuals become more and more similar. This observation still holds, if one avoids duplicates.

Furthermore, we use OBDDs with complementary edges for representation, which allow memory savings of a factor up to two by using a complementary flag bit for every edge labelled by 0 and pointers referencing the sources. During evaluation a complementary flag displays if the referenced subOBDD is negated. Hence, to represent a subfunction and its complement only one subOBDD is necessary, whereby we only consider OBDDs with a 1-sink. By using the OBDD-package CUDD all our $\pi$-SBDDs are reduced and use complementary edges.

These are syntactic aspects of our representation, but we also make a semantic restriction: we only use OBDDs that are consistent with the given training set, i.e., have the correct output values for the training set $T$. This is done by explicitly creating consistent OBDDs during initialization and testing new offspring to be consistent, otherwise replacing them by one of the parents. This method reduces the size of the search space by a factor of $2^{|T|}$, allowing us to measure the fitness of an individual by its size only. This was shown empirically in [5] to be advantageous for test problems of our kind.

4.2 Initialization

While the variable orderings $\pi_j$ are chosen independently using the uniform distribution from all possible permutations of $\{1, \ldots, n\}$, the $\pi_j$-OBDDs itself are created as follows (for easier notation we assume that $\pi_j = id$): starting from the source with label $x_1$, for every actual node labelled $x_i$ the number of different outputs of the training examples consistent with the path to the actual node is computed, where a path is identified with its corresponding partial assignment of the variables. If the number of these outputs is two, the procedure is called recursively to create the 0- and 1-successor with label $x_{i+1}$; if it is one or zero, a random subOBDD is returned by creating the 0- and 1-successor with labels $x_{i+\delta_0}$ and $x_{i+\delta_1}$, where $\delta_0$ and $\delta_1$ are geometrically distributed with parameter $\alpha$. If $i + \delta_0$ resp. $i + \delta_1$ is at least $n + 1$ the corresponding sink is returned, or a random one, if the actual path is not consistent with any training example.

Thus, the way consistent OBDDs are randomly generated is influenced by the parameter $\alpha$: for $\alpha = 1$ the resulting function is uniformly distributed from all functions being consistent with the training set, for $\alpha = 0$ every path being
not consistent with the training set leads to a randomly chosen sink via at most one additional inner node.

4.3 Reordering

All our heuristics for optimizing the variable ordering work on the whole \(\pi_j\)-SBDD representing the \(j\)-th subpopulation. If we would apply the heuristic on every \(\pi_j\)-OBDD we would eventually get smaller OBDDs, but then the subpopulation would consist of OBDDs of different variable orderings. Hence, we apply the chosen heuristic on the whole SBDD. Hereby we hope to achieve an approximation of the optimal variable orderings of the individual OBDDs. To see how a heuristic can look like, we give a short description of sifting ([11]):

First all variables are sorted according to the number of nodes in the SBDD labelled by it. Then, starting with the variable with the lowest number, the variable is stepwise swapped with its neighbours: first to the near end of the variable ordering and then to the far end. Because the SBDD-size after such a swap can be computed quite efficiently, the variable is put to the position where the SBDD size was minimal. This procedure is repeated for the variable with the second-lowest number and so on. To avoid blow-up, this process is stopped if the SBDD-size grows beyond a factor of \(c\) (we choose \(c = 2\)).

4.4 Recombination

In recombination and mutation the parents are chosen proportionally to the normalized fitness \(1/(1 + s)\), where \(s\) is the size of the OBDD. Recombination of two \(\pi_j\)-OBDDs chooses uniformly a node \(v_a\) in the first parent and then a node \(v_b\) in the second parent from all nodes having a label at least that of \(v_a\) according to \(\pi_j\). Then the subOBDD starting with \(v_a\) is replaced by the subOBDD starting with \(v_b\): As there can be many paths to \(v_a\), we choose one of the paths randomly and replace the edge to \(v_a\) by an edge to \(v_b\). For all other paths to \(v_a\) we replace the edge to \(v_a\) with probability \(1/2\), hence considering the role of shared subOBDDs as ADFs. If this new OBDD is not consistent with the training set, it is replaced by one of its parents. If the offspring is already in the population, this procedure is repeated up to ten times, to avoid duplicates.

4.5 Mutation

For mutating a \(\pi_j\)-OBDD a node \(v_a\) of the OBDD to be mutated is chosen randomly under uniform distribution. For a path leading to \(v_a\) it is checked, if its 0- or 1-successor is relevant for consistency with the training set. If not, it is replaced by the other successor. Otherwise, a \(\pi_j\)-OBDD with source \(v_b\) is created randomly using the same algorithm as was applied during initialization for an empty training set, where all nodes have a label at least \(v_a\) with respect to \(\pi_j\). On one randomly chosen path to \(v_a\) the edge to \(v_a\) is replaced by an edge to \(v_b\). On all other paths this is done with probability \(1/2\). Again, a not-consistent offspring is replaced by its parent and this procedure is repeated up to ten times, if the offspring is already in the population.
4.6 Migration strategy

The migration strategy decides how to choose the new variable ordering of the $j$-th population after migration has taken place. Because changing the variable ordering can cause an exponential blow-up, we choose an introverted strategy by changing the variable orderings of all incoming OBDDs to the variable ordering of the $j$th subpopulation, i.e., $\text{migration\_strategy}(j) = \pi_j$.

5 Experimental results

For our experiments we use only the multiplexer-function, because we know by Theorem 1 that it is one of the hardest functions, when it comes to finding small OBDDs that approximate it or even a random sampling of it. So we want to see if our GP-system is capable of finding small OBDDs, where the inputs of the training set are randomly and independently chosen for every run. Furthermore, we are interested in the generalization capabilities of the OBDDs found. Hence, we also investigate the fraction of all inputs, where the smallest OBDD found has the same output as the multiplexer function. We emphasize that no knowledge whatsoever of the multiplexer function influences our GP-system.

<table>
<thead>
<tr>
<th>Number of subpopulations</th>
<th>$k = 4$</th>
<th>Size of subpopulations</th>
<th>$\mu = 40$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of generations</td>
<td>$G = 3000$</td>
<td>Migration rate</td>
<td>$B = 5$</td>
</tr>
<tr>
<td>Length of migration interval</td>
<td>$M = 10$</td>
<td>Length of reordering interval</td>
<td>$N = 20$</td>
</tr>
<tr>
<td>Reordering heuristic</td>
<td>group sifting</td>
<td>Initial size parameter</td>
<td>$\alpha = 0.2$</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>$p = 0.1$</td>
<td>Size of training set</td>
<td>$T = 512$</td>
</tr>
</tbody>
</table>

Fig. 2. Parameter settings for our experiments

The parameters of our GP-system in the experiments are set as shown in Figure 2, where the size of the training set is chosen to match the examples of older OBDD-based GP-systems. The results are shown in Figures 3 and 4, where we choose $n = 20$ and average over 10 runs. We compare our GP-system with three GP-systems, where the variable ordering is fixed to an optimal and a random variable ordering. These systems use only one population of size 160 and no variable ordering heuristic, but the same genetic operators as in our system.

We see in Figure 3, that our GP-system, although being worse than the system using an optimal variable ordering, produces smaller OBDDs than using a random variable ordering: after 3000 generations the average size of the smallest OBDD found is 126.97 in comparison to 185.10 (where a minimal OBDD for MUX$_{20}$ has size 32, as we also count the sink here). Taking the results from Figure 3 and Figure 4 we see that Occam’s razor seems to be valid for MUX$_n$, because the generalization capabilities of the found OBDDs behave according to their sizes: while using the best variable ordering by far results in the best generalization capabilities, our GP-system with sifting outperforms a fixed GP-system with fixed random variable ordering (56.73% in comparison to 54.72%
hits after 3000 generations). But one can also notice, that our system is more capable to reduce the size of the OBDDs than to increase the hit rate. One could conclude that the excellent hit rates of the previous OBDD-based GP-systems are based on the information about the variable ordering.

Hence, our distributed hybrid GP-system empirically improves static approaches using a random variable ordering. The static approach with the optimal variable ordering allows no fair comparison, as in practical applications the variable ordering is unknown and even approximations are hard to compute.
6 Conclusion

OBDDs are a very efficient data structure for Boolean functions and are therefore a successfully used representation in GP. But every OBDD-based GP-system so far uses the additional information of an optimal variable ordering of the function to learn, which is only known if the test function is known and has strong influence on the size of the OBDDs. Hence, we presented a distributed hybrid GP-system, that for the first time evolves the variable ordering and the OBDDs itself. Empirical results show that this approach is advantageous to a GP-system, where the variable ordering is randomly fixed, and also more successful than a simple hybrid approach, in which the number of subpopulations is set to one and the population size is set to 160. Hence, this is a great step towards the practical applicability of GP-systems with OBDDs, since there is no additional necessary input needed beside the training set.

References

Genetic Programming with Dynamic Fitness for a Remote Sensing Application

Cyril FONLUPT and Denis ROBILLIARD

Laboratoire d'Informatique du Littoral
Université du Littoral
BP 719
62228 CALAIS Cedex
FRANCE
phone: +33-3-21-97-00-46
fax: +33-3-21-19-06-61
email: {fonlupt,robillia}@lil.univ-littoral.fr

Abstract. The "ocean color" problem consists in evaluating ocean components concentrations (phytoplankton, sediment, yellow substance) from sunlight reflectance or luminance values at selected wavelengths in the visible band. The interest of this application increases with the availability of new satellite sensors. Moreover monitoring phytoplankton concentrations is a key point for a wide set of problems ranging from greenhouse effect to industrial fishing and signaling toxic algae blooms. To our knowledge, it is the first attempt at this regression problem with Genetic Programming (GP). We show that GP outperforms traditional polynomial fits and rivals artificial neural nets in the case of open ocean waters. We improve on previous works by also solving a range of coastal waters types, providing detailed results on estimation errors. Some experiments were conducted with a dynamic fitness GP algorithm in order to speed up computing time through a process of progressive learning.

1 Introduction

One of the interests of Genetic Programming (GP) lies in its ability to adapt to many types of problems. Within the image analysis field, it has been successfully applied to several difficult tasks, like features extraction, image recognition [1], automatic detector [12], image discrimination [18]... Most of these works use GP primitives based on statistics of pixel data, like averaging pixel values in a \( n \times n \) box, or using standard deviations on a pixel grid [16]. All these works but Daida's [6, 7] are based on the visual spectrum but do not use multi-spectral data. In this paper, we present the first results of an on-going study dealing with multi-spectral remote sensing data analysis, to solve the ocean color problem. The objective of this application is evaluation of phytoplankton concentration in oceanic and coastal waters, using remote sensing measurements of the reflected sunlight. These measurements are made along some wavelengths in the visible band, hence the name "ocean color". We also deal with a more technical matter:
speeding up GP execution time. It is well-known that most of the CPU time, during a GP run, is usually spent on the fitness evaluation. Thus, depending on the problem, the computation time requirements may become quickly impractical. This fitness evaluation problem has been addressed by many people, for example by using a staged approach [12], by limiting the fitness evaluation to a subpopulation [3], by stopping the fitness evaluation as soon as a given threshold of bad individuals has been reached [9]. We experiment a GP algorithm where new cases are gradually added to the set of fitness cases during the run. The idea beyond is that it is very unlikely to quickly find fit individuals for all cases. So, in order to allow the opportunity for a soft adaptation to a more and more complex learning situation, our GP variant starts with a subset of fitness cases, and once a given fitness threshold is reached, new fitness cases are then added. This process is iterated until all fitness cases are viewed or the maximum number of generations is reached.

Section 2 introduces the ocean color problem. Section 3 describes the algorithms we use, and presents some non GP schemes that are currently used or proposed to solve the ocean color. Section 4 presents the results and some comparisons with those other known methods. In section 5 we sum up the lessons brought by these experiments and we sketch how future works could be directed both towards refining the ocean color application, and towards using dynamic fitness cases in other applications.

2 The ocean color problem

Being able to monitor the evolution of ocean water characteristics is an important challenge. The knowledge of phytoplankton concentration is especially interesting, since it allows quantitative assessment of ocean and coastal waters primary production. In turn, this primary production plays a key role for the evaluation of the global carbon cycle and is thus of great scientific concern, notably to understand the so-called greenhouse effect. Phytoplankton is also the base of the marine life food chain, with indirect consequences on the fishing industry. Moreover some species affect human health, through the release of toxins that may be absorbed by edible sea-shells. The availability of new satellite missions, with good resolution (300m) and week-to-week occurrence, has raised new opportunities for monitoring ocean characteristics. The "ocean color" problem consists in retrieving phytoplankton, sediment or yellow substance concentrations, from luminance (or reflectance) values as they are measured by a satellite spectrometer. To provide information on water components, the sunlight has to penetrate into the sea, and be reflected on those components we are interested in. Due to the optical properties of water and its components, interesting information are mostly available around the center of the visible band, hence the name ocean color. We sketch the ocean color problem in Fig. 1.

More formally, the signal \( L \) emitted by the sun, is partly absorbed and partly reflected by the sea water and its components. The energy level of the reflected part of \( L \) can be measured by a satellite spectrometer. It is usually done on a
set of specified wavelengths, depending on the spectrometer used, e.g. 412.5 nm, 442.5 nm, 490 nm, 510 nm, 560 nm, 620 nm and 665 nm for NASA "SeaWiFS" sensor\(^1\). Thus, if we denote \( C \) the phytoplankton concentration (in fact the "chlorophyl-a" concentration), \( S \) the sediment concentration, and \( Y \) the yellow substance (dissolved organic substances) concentration, the direct problem is to find a function \( f \) that computes the vector of reflected energy levels at the monitored wavelengths:

\[
\begin{align*}
 f(L, C, S, Y) &= (L_{412.5}, L_{442.5}, L_{490}, L_{510}, L_{560}, L_{620}, L_{665})
\end{align*}
\]

In the above expression, \( L_{412.5} \) indicates the amount of reflected energy around wavelength 412.5 nm, and so on. Solutions have been proposed to solve this direct problem, notably Morel’s model [14,15,10], and Chami, Dilligear and Santer’s model [4,5]. Note that these models do use a lot more parameters to allow for a range of sedimentary particle sizes and other details (an in-depth discussion on these models would be outside the scope of the paper).

However, in the ocean color problem we face the inverse problem, that is finding the components concentration from the measured reflected energy levels. In this paper, we focus on obtaining only the phytoplankton concentration (\( C \)). We furthermore simplify the problem by assuming a standardized level and direction of the sun light. The other works that we compare to also used this simplified setting. So we are left with a symbolic regression problem, searching for a function \( f' \) such that:

\[
\begin{align*}
 f'(L_{412.5}, L_{442.5}, L_{490}, L_{510}, L_{560}, L_{620}, L_{665}) &= C
\end{align*}
\]

The validity of \( f' \) should be tested under two standard oceanographic cases:

- The K1: this corresponds to the open ocean, where phytoplankton dominates, and sediment and yellow substance are negligible [10]. Satisfactory

\(^1\) An on-line presentation of the NASA SeaWiFS project is available at http://seawifs.gsfc.nasa.gov/SEAWIFS.html
models already exist for the K1, notably the artificial neural nets we refer to in our results section. Nonetheless, we were curious of the performance of GP on this problem.

- The K2: this corresponds to coastal waters, where sediment and yellow substance may have to be taken into account, depending on the specific geographic area that is monitored. It is seen as a much harder problem than K1, and still mostly unexplored. In this study we have considered only a subset of the general K2 problem: waters where phytoplankton concentration ranges from 0 to 50 \( mg.m^{-3} \), while sediment concentration varies from 0 to 30 \( mg.l^{-1} \), with negligible yellow matter concentration. As for phytoplankton and sediment, this setup is typical of the English Channel and North Sea. Future works should be directed at solving cases with significant levels of yellow matter.

In both K1 and K2 case, atmospheric noise was not considered, and this will also be an important point to take into account in future works.

3 The GP methodology

3.1 Overview

We tried both standard runs of GP and a “progressive” multi-stage variant, inspired from the one that Howard and Roberts have proposed in [12]. In these experiments rough algorithms are expected to be quickly developed, and then refined in later stages. To obtain this progressive development, we partition the fitness cases in several subsets, called “classes”, of same size. The details of class partition are described in the next subsection. In the standard run, all classes are used from the start of the run. In the progressive variant, the algorithm starts with one class, and when the best individual of the population reaches a given threshold of fitness, a new class is iteratively added. This is summed up in Tab. 1.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generate initial random population</td>
</tr>
<tr>
<td>2</td>
<td>Insert one class of fitness cases into the fitness cases pool (FCP)</td>
</tr>
<tr>
<td>3</td>
<td>Until termination criterion is satisfied loop</td>
</tr>
<tr>
<td>4</td>
<td>Perform genetic operations (crossover, ...)</td>
</tr>
<tr>
<td>5</td>
<td>Evaluate fitness of individuals in population using FCP</td>
</tr>
<tr>
<td>6</td>
<td>If adjusted fitness of best individual &gt; threshold then</td>
</tr>
<tr>
<td></td>
<td>Insert next fitness cases class into FCP</td>
</tr>
<tr>
<td>7</td>
<td>End if</td>
</tr>
<tr>
<td>8</td>
<td>End loop</td>
</tr>
</tbody>
</table>

**Table 1.** Pseudo-code for the progressive GP variant
The experiments were implemented with the Lilgp library\(^2\), from the GARAGe group at the University of Michigan. This library was slightly modified to implement the progressive learning process.

We compare our results on K1 waters with those from two other methods available in the literature [10]. The first traditional model is based on polynomial approximations (see also [11,4]). The second one is based on artificial neural nets. For K2 waters, the most advanced published study seems to be [17], but it provides no numerical results to allow comparisons with our study.

### 3.2 Learning sets

It is difficult to obtain training sets based on real measures, since one has to organise sea cruises and satellite data gathering with perfect synchronization. Thus, one main interest of having simulation codes for the direct problem is the ability to generate training and test data sets.

In the K1 case, we first tested our progressive algorithm versus standard GP, on a data set generated with a software based on Chami et al.'s model ([4, 5]). The fitness cases set is composed of 91 elements, divided into 7 training classes of increasing concentration, and the test set is 309 elements. In a second experiment, we compared our progressive algorithm with polynomial fits and artificial neural nets, on the same data set that was used by [10] to train these methods. This set is based on a simulation of Morel's model ([14,15]). We used 1000 learning cases, randomly divided into 20 training classes for progressive learning, and 1000 tests cases for validation.

In the K2 case, we created our data with Chami et al.'s model. The fitness cases set is composed of 91 elements, and the test set is 196 elements. The fitness cases were divided into 7 classes of increasing sediment concentration, while phytoplankton concentrations were evenly represented in all classes.

The terminal inputs for the GP process were the energy levels \(\{L_{412.5}, L_{442.5}, L_{490}, L_{510}, L_{550}\}\) for Morel's model, with two more inputs \(\{L_{620}, L_{665}\}\) in case of Chami et al.'s model\(^3\).

### 3.3 Fitness evaluation

The raw fitness of programs is given by the relative RMS error on the training set:

\[
\text{relative RMS} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left( \frac{C_{\text{computed}} - C_{\text{expected}}}{C_{\text{expected}}} \right)^2}
\]

The fitness threshold in progressive learning experiments is based on adjusted fitness:

\(^2\) URL [http://isl.cps.msu.edu/GA/software/lil-gp](http://isl.cps.msu.edu/GA/software/lil-gp)

\(^3\) As said above, these are standard SeaWiFS sensor spectral bands
adjusted fitness = \frac{1}{1 + \text{relative RMS}}

Notice that we have to re-evaluate every member of the population when we add a new class of fitness cases in our progressive learning experiments.

We always use a test set independent from the learning set to compute the relative RMS values given as results in the next section. Relative RMS error allows to get rid of absolute value considerations. This is especially important in the K1, because huge sea areas may have very low phytoplankton concentrations, and thus we need an accurate inverse model for these low values.

4 Experimental Results

4.1 K1 waters

Comparing standard and progressive GP. For this first experiment, we tried standard GP, and our progressive algorithm based on an adjusted fitness threshold. Both algorithms were given 10 runs, on a data set that simulates open ocean conditions. The parameter setting is given in Tab. 2, and Tab. 3 sums up the results. Relative RMS is shown with the following format: best (mean / standard deviation). In this case it appears that not only the progressive variant needs less computing time, as it was expected by construction, but also offers results that are better than standard runs.

Table 2. Parameter setting: K1 case, standard versus progressive GP.

<table>
<thead>
<tr>
<th>Objective</th>
<th>compute phytoplankton concentration $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set</td>
<td>${+, -, *, /}$</td>
</tr>
<tr>
<td>Fitness case</td>
<td>91 results of radiative transfer simulation</td>
</tr>
<tr>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum number of generations</td>
<td>100</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>80 %</td>
</tr>
<tr>
<td>Copy probability</td>
<td>15 %</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>5 %</td>
</tr>
<tr>
<td>Selection</td>
<td>tournament of size 7</td>
</tr>
<tr>
<td>Maximum tree depth</td>
<td>10</td>
</tr>
<tr>
<td>Progressive fitness threshold</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Progressive GP versus traditional methods. In this second experiment, we used the same data set as [10], and we have given more time for the progressive GP algorithm to build its model. GP results are computed on 5 runs. The
Table 3. Results on K1 case, standard versus progressive GP.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>relative RMS (%)</th>
<th>mean time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>6.5 (88.3 / 136)</td>
<td>575</td>
</tr>
<tr>
<td>Progressive</td>
<td>1.7 (37.9 / 26.2)</td>
<td>412</td>
</tr>
</tbody>
</table>

The parameter setting is given in Tab. 4, and Tab. 5 sums up the results. Relative RMS is shown with the following format: best (mean / standard deviation). The GP results are a clear improvement on polynomial models, and very close to artificial neural nets.

Table 4. Parameter setting : K1 case, progressive GP versus other methods.

<table>
<thead>
<tr>
<th>Objective</th>
<th>compute phytoplankton concentration $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set</td>
<td>${ +, -, *, / }$</td>
</tr>
<tr>
<td>Fitness case</td>
<td>1000 results of Morel’s model simulation</td>
</tr>
<tr>
<td>Population size</td>
<td>5000</td>
</tr>
<tr>
<td>Maximum number of generations</td>
<td>500</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>85 %</td>
</tr>
<tr>
<td>Copy probability</td>
<td>5 %</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>10 %</td>
</tr>
<tr>
<td>Selection</td>
<td>tournament of size 5</td>
</tr>
<tr>
<td>Maximum tree depth</td>
<td>18</td>
</tr>
<tr>
<td>Progressive fitness threshold</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 5. Results on K1 case, progressive GP versus other methods.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>relative RMS (%)</th>
<th>mean time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progressive</td>
<td>0.26 (0.8694 / 0.0019)</td>
<td>10 h</td>
</tr>
<tr>
<td>Polynomial fit #1</td>
<td>8.59</td>
<td>-</td>
</tr>
<tr>
<td>Polynomial fit #2</td>
<td>5.14</td>
<td>-</td>
</tr>
<tr>
<td>Artificial Neural Net</td>
<td>0.14</td>
<td>-</td>
</tr>
</tbody>
</table>

4.2 K2 waters

The parameters setting for coastal waters case is given in Tab. 6. We tried again standard versus progressive GP. Every algorithm was given 10 runs. Table 7
sums up the results obtained in the K2 case. Relative RMS is shown with the following format: best (mean / standard deviation).

**Table 6. Parameter setting for K2 case.**

<table>
<thead>
<tr>
<th>Objective</th>
<th>compute phytoplankton concentration C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set</td>
<td>{ +, -, *, /, exp, log }</td>
</tr>
<tr>
<td>Fitness case</td>
<td>91 results of radiative transfer simulation</td>
</tr>
<tr>
<td>Population size</td>
<td>500</td>
</tr>
<tr>
<td>Maximum number of generations</td>
<td>700</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>65 %</td>
</tr>
<tr>
<td>Copy probability</td>
<td>15 %</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>20 %</td>
</tr>
<tr>
<td>Selection</td>
<td>tournament of size 7</td>
</tr>
<tr>
<td>Maximum tree depth</td>
<td>15</td>
</tr>
<tr>
<td>Progressive fitness threshold</td>
<td>0.7</td>
</tr>
</tbody>
</table>

**Table 7. Results for the K2 case**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>relative RMS (%)</th>
<th>mean time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>9.3 (109.5 / 95.5)</td>
<td>3270</td>
</tr>
<tr>
<td>Progressive</td>
<td>6.7 (2538.3 / 7750.2)</td>
<td>3041</td>
</tr>
</tbody>
</table>

In this experiment, only 5 over 10 runs of the progressive variant algorithm achieved to do their learning on the whole set of fitness cases. Indeed the other half of the runs, for some class of fitness cases, never developed an individual with adjusted fitness as big as the threshold. Thus they learned only part of the fitness cases set, and shown wide errors when applied to the test set: these are “failed” runs. This results in the very high mean RMS error and standard deviation that can be found on line “Progressive” in Tab. 7. We tried a more “comfortable” setting (runs with population size 1000 and number of generation 1000) without much improvement in this regard. For this progressive variant, the gain in computing time is less significant than in the K1 case. The best individual shows an improvement in performance when compared to the standard GP run, although the huge standard deviation (due to these “failed” runs) prevents generalization.

The evolution of best and mean adjusted fitness in progressive GP is shown in Fig. 2. One can see the perturbation that accompanies the introduction of each new fitness cases class. For the K1 problem, the smoothness of the curve suggests that first GP “guesses” were good, and only one class needed noticeable
adjustment. On the contrary, the K2 problem seems much harder, each new class bringing important change to the population.

(a) Progressive GP on K1 problem.  (b) Progressive GP on K2 problem.

Fig. 2. Best and mean fitness evolution when using dynamic fitness cases.

5 Conclusion

This is a first report on a on-going study. The easy K1 problem has been solved with good precision, and GP performance outmatches traditional polynomial fits and rivals artificial neural nets. The K2 precision is not as good as the K1, but is still considered very good by physicists. To our knowledge, it is the first study that provides a model solving a range of K2 waters, accompanied with detailed results and estimation errors. Following previous works from the GP community, we tried a multi-stage algorithm. In both K1 and K2 cases, this progressive algorithm, with fitness threshold, provided us with better results, and a faster algorithm than standard GP.

We plan to develop this work into several directions: using bigger fitness cases sets for K2 problem, including yellow substances and noise. More experiments are also needed to assess the power of this progressive dynamic fitness scheme. A challenging task would be to characterize which problems can be turned into “separate problems”, i.e. problems that can benefit from having their set of fitness cases cut into classes? It is quite clear that introducing fitness classes step by step in the GP process can reduce the overall computing time, yet the reason why it improves performance on this application is still to be explained.

Acknowledgments: we thank M. Chami and R. Santer, at the LISE laboratory, for providing us with the physical models for generating fitness cases sets and their many helpful hints. We also thank the anonymous referees for their valuable comments.
References

Genetic Programming Bloat without Semantics

W.B. Langdon\(^1\) and W. Banzhaf\(^2\)

\(^1\) Computer Science, University College, London, Gower Street, London, WC1E 6BT
\(^2\) Department of Computer Science, University of Dortmund, D-44221 Dortmund

Abstract. To investigate the fundamental causes of bloat, six artificial random binary tree search spaces are presented. Fitness is given by program syntax (the genetic programming genotype). GP populations are evolved on both random problems and problems with "building blocks". These are compared to problems with explicit ineffective code (introns, junk code, inviable code). Our results suggest the entropy random walk explanation of bloat remains viable. The hard building block problem might be used in further studies, e.g. of standard subtree crossover.

1 Introduction

It has been suggested that ineffective code (introns, junk code) is essential for bloat found in genetic programming (GP). An alternative interpretation is based on random expansion into the accessible parts of the search space. To investigate the differences between these a series of artificial random binary tree search spaces are presented. These avoid the complications associated with the semantics of the programming language by defining fitness directly from the syntax of each program (the GP genotype). In order to avoid introducing hidden assumptions, the remaining linkage within the search spaces is random.

A total of six parameterisable problems are investigated: random and two variations where fitness is composed of "building blocks" (hard and easy variants). For each of these a second version is obtained by explicit introduction of syntax elements which do not change the fitness of a program containing them.

In the next section we give the background and indicate why studies of bloat remain important and then Sect. 3 summarises theories about bloat. Section 4 describes the six artificial problems. GP is evaluated on them in Sect. 5. Finally Sect. 6 draws some conclusions for automatic programming.

2 Background

Many researchers have investigated the importance of crossover in genetic programming (GP), particularly its effectiveness compared to mutation. Mostly these have been empirical studies [27,10,2,25,7],[14, Chapter 56]. While this may be viewed as a GP issue, there is a crossover-related controversy in bit string genetic algorithms (GAs). Proponents of GAs argue that their effectiveness in real problems comes from their use of genetic crossover in a population
of trial solutions. They argue crossover and selection would allow improved individuals to be created from components of the better samples of the search space that have already been tested. Note the assumption that in soluble real problems better (or ideally the best) solutions can be constructed from parts, and that the parts themselves are good (or at least better than average). Further it is assumed that solutions to the problem have been represented by the GA user in such a way that crossover is capable of doing the assembly at random. This is the well known building-block hypothesis. Even in simple fixed representation GAs this remains disputed, although there is a growing body of theory about building blocks in GAs.

From the empirical studies of GP it has been known for some time that programs within GP populations tend to rapidly increase in size as the population evolves [13, 1, 33, 4, 26, 16, 32, 24]. If unchecked, this consumes excessive machine resources and so is usually addressed either by enforcing a size or depth limit on the programs or by an explicit size component in the GP fitness [13, 12, 34, 29] although other techniques have been proposed [30, 6, 32, 31, 18]. Depth or size limits [9, 20] and simple parsimony pressure [32] may have unexpected and untoward effects, while [11] shows that addition of duplicated code segments (i.e. addition of ineffective code, code that has no impact on the behaviour of the program) can sometimes be helpful. Therefore it is still interesting to explore why such bloat happens, and the factors influencing its speed and its limits.

3 Bloating

Tackett [33, page 112] and Altenberg [1] both suggest the common "intron" explanation for bloat is due to Singleton (however James Rice and Peter Angeline may also have contributed). Briefly this says program size tends to increase over time because programs which are bigger contain more ineffective code ("junk" code, code that has no effect on the program). Since changes to ineffective code have no impact on the execution of the program, a higher proportion of ineffective code means a higher chance programs produced by crossover will act like their parents. Therefore they will also be of high fitness and so themselves have a higher chance of being selected to reproduce. Various experiments have shown this to be essentially correct. However Soule [24] shows that there are at least two mechanisms involved. It needs to be noted that this implicitly assumes that the problem is static, i.e. behaving as your parents (who must have been good to have been selected to have children) will also be good for you. Yet bloat has also been observed in dynamic problems [23]. An alternative suggestion [3] that ineffective code could act as safe storage areas for code which is not needed at present but may be needed in future has received only little experimental support [11].

While the ineffective code mechanism is essentially correct one of us (W.B.L.) has proposed a simpler alternative explanation which is independent of mechanisms and indeed has been applied to non-GP search: After a period GP (or any other stochastic search technique) will find it difficult to improve on the
best trial solution it has found so far and instead most of the trial solutions it 
finds will be of the same or worse performance. Selection will discard those that 
are worse, leaving active only those that are as good as the best-so-far. In the 
absence of bias, the more plentiful programs with the current level of perfor­
mance are more likely to be found [21]. But as [17] proves the distribution of 
these is similar to the distribution of trees, therefore we expect the search to 
evolve in the direction of the most popular tree shape. I.e. trees whose depth lies 
near $2^{\sqrt{\pi}N}$ (internal nodes) (ignoring terms $O(N^{1/4})$ ) [8]. See Flajolet parabola 
curve in (lower right graph in) Figs. 3 and 4. [24] and [17] confirm this in various 
problems when using GP with standard crossover. The simplicity of this expla­
nation makes it easy to apply to non-GP search [15,22], to devise new genetic 
operators [18], to explain the evolution of shape [24] and to produce quantitative 
predictions of the evolution of both size and shape [19]. 
In [5] we consider bloat in abstract representation-less random search spaces 
and highlight the importance of inviable code. In the next section we extend this 
approach to more concrete but still artificial representations.

4 Artificial Binary Tree Search Spaces

To repeatably assign a fitness value to a program based only on its syntax, the 
syntax is first reduced to a single hash value. This is deterministically converted 
into a random fitness value. Hashing techniques are used to ensure similar pro­
grams have different hash values and so different fitnesses. 
Ineffective code is introduced by a special function within the function set 
(opcode 1). When calculating fitness, subtrees starting with opcode 1 are treated 
as if they had been replaced by their first leaf, i.e. the rest of the subtree is ignored 
(see dashed subtree in Fig. 1). This means ineffective code always lies away from 
the root and towards the leaves, as is usually the case in GP [32].
4.1 Random Binary Tree Search Spaces

The programs are composed of four functions, opcodes 0...3, and six terminals (leafs), opcodes 4...9 (cf. Fig. 1). To hash the program tree it is traversed in conventional depth-first order. At each node the opcode is packed into an integer. When the 31 bit integer (the sign bit is not used) is full a new integer is used. (Since $10^9 \leq 2^{31}$ nine syntax elements can be packed into each integer.) Successive 31 bit values are combined using XOR. Thus the whole tree is reduced in a single pass to an integer value. This is converted to a random fitness by adding a large constant and feeding the sum into a pseudo-random number generator [28]. Alternative problems can be generated by changing the constant (As a confidence check, many runs were repeated replacing Park-Miller with Mersenne Twister. No important differences were noted). The fitness is the number of bits set (1...31) in the random number produced.

4.2 Random Binary Sub-Tree Building Block Search Spaces

In the building block (BB) problems we define fitness as the combined fitness of the building blocks within the program. Every subtree within the program is treated as a building block and given a random fitness using the mechanisms described above in Sects. 4 and 4.1. (With a suitable stack, fitness can still be calculated in a single pass through the tree.) To simulate idealised building blocks, the randomised value of each subtree is converted to a bit location (0...31) which is set. The behaviour of the whole tree is the union of these bits and its fitness is the number of bits set.

In the easy problem each bit is equally likely to be set, so finding a tree which sets all 32 is relatively easy. Each subtree's bit is given by the least significant five bits of its randomised value.

The more difficult case is where some bits are much more likely to be set than others. Each subtree's bit is now calculated by counting the number of set bits in the random value of the subtree. This gives a binomial distribution (1...31) centered in the middle of the word. Setting the bits far from the middle is very rare and achieving maximum fitness (31) is very difficult.

5 Experiments

On each search space we ran 10 independent GP runs for each of two or four different ways of creating the initial population. Apart from the search space, the absence of size or depth restrictions and the use of tournament selection the GP runs are essentially the same as [13]. Table 1 gives parameters.

The average evolution of each GP population from initial trees of two different ranges of sizes (r2:6 and r7:8) on the six problems (random, easy and hard building blocks, with and without explicit ineffective code) is plotted in Figs. 2-4 and summarised in Table 2 (numbers in round brackets indicate the standard deviation across ten runs).
Table 1. GP Parameters for Random Search Problems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions set</td>
<td>b0 b1 (intron) b2 b3</td>
</tr>
<tr>
<td>Terminal set</td>
<td>t4 t5 t6 t7 t8 t9</td>
</tr>
<tr>
<td>Fitness</td>
<td>Determined by tree syntax</td>
</tr>
<tr>
<td>Selection</td>
<td>Tournament group size of 7, non-elitist, generational</td>
</tr>
<tr>
<td>Wrapper</td>
<td>none</td>
</tr>
<tr>
<td>Pop Size</td>
<td>500</td>
</tr>
<tr>
<td>Max program</td>
<td>no limit</td>
</tr>
<tr>
<td>Initial pop</td>
<td>Created using “ramped half-and-half” with depths between 2 and 6 (r2:6), 7 and 8 (r7:8), ramped uniform (u7:55) or ramped sparse depths 4 and 28 (i.e. size s7:55) (no uniqueness requirement)</td>
</tr>
<tr>
<td>Parameters</td>
<td>90% one child crossover, no mutation. 90% of crossover points selected at functions, remaining 10% selected uniformly between all nodes.</td>
</tr>
<tr>
<td>Termination</td>
<td>50 generations</td>
</tr>
</tbody>
</table>

As expected, all but the easy building block landscape prove difficult for GP, and programs of the maximum possible fitness are only found in the easy landscapes, Fig. 3. The presence of explicit ineffective code in the search space makes little difference in the best of run fitness but, particularly in the random landscape (Fig. 2, top left), it does reduce disruption by crossover so increasing average fitness (plotted with lines) in the population. Also the size of the initial programs makes little difference to the overall behaviour but the shape of the initial programs is important. In all but the random landscape, bloat occurs.

5.1 Anti-Bloat in Random Binary Tree Landscape

Only in the case of the totally random landscape do we not see bloat. Here runs started with both normal and large random trees converge to tiny trees whose parents are identical. (Similar convergence is reported in [19], [16, page 184]). This is explained by the difficulty of the search, so in all most all generations no improved programs are found. Non-elitist selection means improved solutions are removed from the population at the end of the generation. However they have children but they are in competition with each other as well as the rest of the population. Thus only genetic lines which reproduce themselves fastest continue. Most children are produced by subtree crossover. Therefore programs which crossover is more likely to reproduce exactly (clone) have an advantage. The chance subtree crossover between two identical parents producing a clone falls as the parent trees get bigger. This means smaller programs have an advantage (when crossover cannot find either better programs or programs of the same fitness which are not identical to their parents) [16, pages 197–201]. An equilibrium is reached between the local optima and its unfit offspring. These equilibria are stable for at least 1000 generations.

5.2 Evolution of Depth

With bushy initial trees (r2:6 and r7:8) and except for both the easy landscapes, average program height increases roughly linearly by about one level per gen-
Fig. 2. Evolution of GP populations on random and intron landscapes. Means of ten runs with standard (r2:6) and large (r7:8) initial populations.

Table 2. Mean of 10 runs on each landscape and method of creating the initial population. Power law fit of mean program size in population over last 38 gens. v. gen.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Initialization</th>
<th>Fitness</th>
<th>Final population mean size</th>
<th>max size</th>
<th>Depth mean per gen</th>
<th>Power law Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>R2-6</td>
<td>25 (1)</td>
<td>6 (0.9)</td>
<td>11 (3)</td>
<td>3</td>
<td>1.23 (.09)</td>
</tr>
<tr>
<td></td>
<td>R7-8</td>
<td>25 (1)</td>
<td>10 (12)</td>
<td>57 (130)</td>
<td>3</td>
<td>1.19 (.08)</td>
</tr>
<tr>
<td></td>
<td>intron</td>
<td>R2-6</td>
<td>25 (1)</td>
<td>530 (200)</td>
<td>2100 (1300)</td>
<td>33.0 (0.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R7-8</td>
<td>25 (.7)</td>
<td>950 (210)</td>
<td>2700 (480)</td>
<td>30.5 (0.1)</td>
</tr>
<tr>
<td>BBlock</td>
<td>R2-6</td>
<td>32 (0)</td>
<td>870 (100)</td>
<td>2800 (480)</td>
<td>33.4 (0.1)</td>
<td>1.02 (.12)</td>
</tr>
<tr>
<td>(easy)</td>
<td>R7-8</td>
<td>32 (0)</td>
<td>730 (59)</td>
<td>2100 (490)</td>
<td>23.3 (0.0)</td>
<td>1.10 (.08)</td>
</tr>
<tr>
<td></td>
<td>U7-55</td>
<td>32 (0)</td>
<td>1500 (170)</td>
<td>5900 (2000)</td>
<td>93.1 (0.3)</td>
<td>.96 (.09)</td>
</tr>
<tr>
<td></td>
<td>S7-55</td>
<td>32 (0)</td>
<td>2900 (680)</td>
<td>14000 (4200)</td>
<td>527.8 (6.4)</td>
<td>.98 (.11)</td>
</tr>
<tr>
<td></td>
<td>intron</td>
<td>R2-6</td>
<td>32 (0)</td>
<td>1100 (130)</td>
<td>3900 (880)</td>
<td>38.5 (0.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R7-8</td>
<td>32 (0)</td>
<td>1000 (160)</td>
<td>2800 (470)</td>
<td>26.3 (0.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U7-55</td>
<td>32 (0)</td>
<td>1700 (220)</td>
<td>5600 (920)</td>
<td>97.1 (0.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S7-55</td>
<td>32 (0)</td>
<td>2600 (340)</td>
<td>13000 (2200)</td>
<td>413.7 (1.6)</td>
</tr>
<tr>
<td>BBlock</td>
<td>R2-6</td>
<td>26 (.8)</td>
<td>9000 (4600)</td>
<td>20000 (11000)</td>
<td>132.2 (1.7)</td>
<td>1.34 (.19)</td>
</tr>
<tr>
<td>(hard)</td>
<td>R7-8</td>
<td>25 (.8)</td>
<td>5900 (2800)</td>
<td>13000 (5100)</td>
<td>73.1 (0.4)</td>
<td>1.50 (.64)</td>
</tr>
<tr>
<td></td>
<td>U7-55</td>
<td>26 (.4)</td>
<td>14000 (7100)</td>
<td>33000 (15000)</td>
<td>306.6 (2.6)</td>
<td>1.32 (.20)</td>
</tr>
<tr>
<td></td>
<td>S7-55</td>
<td>28 (.9)</td>
<td>76000 (47000)</td>
<td>200000 (110000)</td>
<td>3145.7 (42.3)</td>
<td>1.60 (.17)</td>
</tr>
<tr>
<td></td>
<td>intron</td>
<td>R2-6</td>
<td>25 (.5)</td>
<td>6300 (2200)</td>
<td>14000 (3700)</td>
<td>93.2 (0.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R7-8</td>
<td>24 (.9)</td>
<td>4500 (2400)</td>
<td>11000 (6600)</td>
<td>69.1 (0.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U7-55</td>
<td>25 (.7)</td>
<td>17000 (9000)</td>
<td>44000 (26000)</td>
<td>359.8 (5.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S7-55</td>
<td>26 (.8)</td>
<td>24000 (17000)</td>
<td>71000 (55000)</td>
<td>1078.256 (19.8)</td>
</tr>
</tbody>
</table>
Fig. 3. Evolution of GP populations on easy correlated landscapes. Means of ten runs with standard (r2:6) and large (r7:8) initial populations. Lower right (evolution of shape) also includes average evolution of runs started with uniform (u7:55) and sparse populations (s7:55). Tick marks at each generation.

Fig. 4. Evolution of GP populations on hard correlated landscapes.
eration (cf. Table 2 col 10). This has been observed in several GP benchmark problems [24, 18].

The non-linear, slower depth increase in the easy search spaces is produced by reduced selection pressure. GP populations converge in the sense that most of the programs within them have identical fitness values. On average in the last ten generations 79–86% of tournaments are between potential parents of identical fitness and so parents are chosen at random.

Unexpectedly populations which initially contain random trees (u7:55) increase their depth faster than those starting with bushy trees (r2:6 and r7:8). However depth increase in sparse trees (s7:55) is even bigger and is non-linear but faster bloat of thin trees can be expected.

5.3 Evolution of Shape

The average shape of trees within the populations evolves away from the bushy nearly full trees created by “ramped half-and-half” in the direction of the most popular part of the search space (denoted by “peak” or “Flajolet” in Figs. 2–4). However the population mean lies to the left of it. There are several possible reasons for this: 1) The ridge itself is quite wide and, except near the boundaries, change in program depth makes comparatively little difference to the number of programs, i.e. the local gradient lies nearly parallel to the size axis. 2) The initial population is to the left. 3) As is usual, our subtree crossover is biased to select functions as crossover points rather than terminals. 4) Interaction with the search space. To test the fourth option, ten runs were conducted on both building block landscapes (with and without explicit introns) starting with a) random sparse trees and b) trees selected uniformly at random between two sizes [18]. As observed in GP benchmark problems [24] populations initialised with sparse trees evolve towards the most populated part of the search space but remain on the sparse (right) side of it. These trees are taller than used in [24] and they remain comparatively sparse, i.e. they move more slowly towards the ridge. While those initialised with common tree shapes remain near this ridge in the search space, see Figs. 3–4 (lower right). Therefore it does not appear that these search spaces promote the evolution of busier trees.

5.4 Sub-Quadratic Bloat

As discussed in [24, 18, 19] if the programs within the population remain close to the ridge in the number of programs versus their shape and they increase their depth at a constant rate this leads to a prediction of sub-quadratic growth in their lengths. For modest size programs we expect size $O(gens^{1.3})$ rising to a limit of quadratic growth for $|\text{program}| \gg 1000$ cf. [8, Table II]. For traditional bushy initial tree and excluding the very hard and the easy landscapes, Table 2, column 10, reports variation between runs but values near 1.3 on average.
6 Conclusions
We have investigated subtree crossover, building blocks and bloating by using random search spaces where fitness is based only on syntax. Thus avoiding consideration of the semantics of the programming language. We find GP behaves like it does on real program spaces (i.e. with semantics).

Only in the case of a totally random landscape does bloat not occur. In this case crossover is unable to find children which are different from their parents (i.e. to explore) and who have a fitness at least as good as their parents. I.e. the extreme nature of the problem prevents bloat. In the other problems there is correlation and bloat occurs, whether the correlation is due to building blocks or the explicit introduction of ineffective code. Thus the entropy explanation of bloat [21,24] remains viable.

The failure of GP to solve the harder problem with building blocks might be due to premature convergence. This may be a common failing in similar genetic search and these artificial search spaces may be useful in future research to investigating this and other aspects of GP and related techniques.

C++ code may be found in ftp://cs.ucl.ac.uk/wblangdon/gp-code

Acknowledgements
We acknowledge funding of the “Deutsche Forschungsgemeinschaft”, under project B2 of “Sonderforschungsbereich SFB 531”.

References


Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery

Alain Ratle and Michèle Sebag

LMS - CNRS UMR 7649, Ecole Polytechnique, 91128 Palaiseau Cedex

Abstract. Application of Genetic Programming to the discovery of empirical laws is often impaired by the huge size of the domains involved. In physical applications, dimensional analysis is a powerful way to trim out the size of these spaces. This paper presents a way of enforcing dimensional constraints through formal grammars in the GP framework. As one major limitation for grammar-guided GP comes from the initialization procedure (how to find admissible and sufficiently diverse trees with a limited depth), an initialization procedure based on dynamic grammar pruning is proposed. The approach is validated on the problem of identification of a materials response to a mechanical test.

1 Introduction

This paper investigates the use of Genetic Programming [Koz92] for Machine Discovery (MD), the automatic discovery of empirical laws. In the classical Machine Learning framework introduced in the seminal work of Langley [LSB83], MD systems are based on inductive heuristics combined with systematic exploration of the search space. This approach suffers from severe limitations with real-world problems, due to ill-conditioned data and huge search spaces.

Such limitations are avoided in Genetic Programming (GP) due to its stochastic search principle. The price to pay is that GP offers no direct way to incorporate expert knowledge, although the knowledge-based issues of Evolutionary Computation are now recognized [Jan93]. In this paper, the emphasis is put on a particular, albeit rather general, expertise: in all application domains, variables most often have physical dimensions that cannot be ignored, for example, mass and length can not be added together. The restriction of the search space to dimensionally admissible laws has been tackled by [WM97] in a Machine Learning framework and by [KB99], using dedicated GP operators. On the other hand, an elegant and promising way to encode domain-knowledge is by formal grammars [RC098,Hör96]. One major difficulty with Grammar-Guided Genetic Programming (G3P) lies in the initialization step, the importance of which cannot be overestimated [Dai99]. Finding admissible trees within a maximum depth might be challenging enough to result in poorly diversified populations.

This paper investigates the use of grammars to restrict the GP search space to dimensionally admissible laws. The next section briefly presents context-free
grammars and Sect. 3 discusses some related works. Sect. 4 describes a class of grammars for dimensionally admissible expressions, Sect. 5 presents its use for generating the population, and finally, Sect. 6 reports on numerical experiments with G3P for the identification of phenomenological laws in materials science.

2 Context-Free Grammars

A Backus-Naur form (BNF) grammar describes the admissible constructs of a language through a 4-tuple \( \{ S, N, T, P \} \), where \( S \) denotes the start symbol, \( N \) the set of non-terminal symbols, \( T \) the set of terminal symbols, and \( P \) the production rules. Any expression is built up from the start symbol. Production rules specify how should the non-terminal symbols, e.g. \(< expr >\), be rewritten into one of their derivations (e.g. \(< oper >< expr >< expr >\) or \(< var >\)) until the expression contains terminal symbols only.

\[
\begin{align*}
N &= \{ < expr >, < oper >, < var > \} \\
T &= \{ x, R, +, *, (, ) \} \\
P &= \{ S := < expr > \\
    < expr > := ( < oper >< expr >< expr > ) | < var > \\
    < oper > := + | * \\
    < var > := x | R \\
\}
\end{align*}
\]

The above grammar describes all polynomials of the variable \( x \) (\( R \) is interpreted as any real-valued constant); hence it is equivalent to the GP search space with the node set \( N = \{ +, * \} \) and terminal set \( T = \{ x, R \} \). One advantage of grammars is to allow fine-grained constraints to be imposed on the search space. Assume for instance that for one particular application, the parent node of an additive node must be a multiplicative node only, and vice versa. This is enforced via grammars by describing two non-terminals, \(< add - expr >\) and \(< mult - expr >\), with the following production rules:

\[
\begin{align*}
< add - expr > &:= ( + < mult - expr >< mult - expr > ) | < var >; \\
< mult - expr > &:= ( * < add - expr >< add - expr > ) | < var >;
\end{align*}
\]

In canonical GP, satisfying this constraint would require either to design a specific initialization procedure and evolution operators, or to filter out any non-complying individual.

3 GP and Grammars: Previous Works

Canonical GP relies on the hypothesis of closure of the search space [Koz92], which assumes that the return value of any subtree is a valid argument for any

\footnote{It should be clear that \textit{non-terminals} and \textit{terminals} have different meanings in canonical GP and in BNF grammars. In the former, terminals are the leaves (variables and constants), and non-terminals are the nodes (operators), while in the latter, terminals comprise both variables and operators, and non-terminals are expressions that must be resolved into terminals.}
function. This ensures that simple crossover and mutation (respectively swapping sub-trees and replacing an arbitrary subtree by a random one) shall produce admissible offsprings. What is gained in procedural overhead is lost in expressiveness: neither syntactic nor semantic restrictions are accounted for, and prior knowledge can dictate nothing but the node set. This implies several limitations:

- The size of the search space is huge, even for problems of moderate difficulty [Whi95]: it is typically exponential with respect to the number of terminals and nodes and to the maximum depth.
- The general shape of the trees is arbitrary.
- No consideration is given to the types (integers, reals, complexes,...).
- Variables are assumed to be dimensionless.

Consequently, the use of canonical GP with typed or dimensioned variables implies the useless generation of a vast majority of irrelevant trees [CY97]. Several authors have addressed this problem using various kinds of bias. A first kind is provided by the expert through domain knowledge. The importance of taking this knowledge into account is now generally admitted [Jan93]. In an MD context, prior knowledge might concern the shape of the solution\(^2\). A significant improvement in the success rate of a GP application can be obtained by biasing the shape of the parse trees toward some shapes that are a priori judged interesting. This can be enforced by syntactic constraints; their beneficial effect have been illustrated by Whigham [Whi95] for the 6-multiplexer problem.

The use of syntactic constraints with genetic programming have been suggested as a potential form of bias by Koza [Koz92] in 1992. More formally, Gruau [Gru96] has shown that syntactic constraints can be used for reducing the size of the search space by allowing only type-consistent parse trees. However, a major limitation of Gruau's approach is that no limitation is put on the depth of the trees. This usually results in a severe growth in tree size.

A second kind of bias consists of constraining the types of the variables manipulated by the tree expression. These constraints might be related to the adequacy of the variables and operators (e.g. don’t take the square root of a negative value), or to the physical dimensionality of the variables. A first step toward dimensionally aware GP was proposed by Keijzer and Babovic [KB99]. The dimensionality of each expression is encoded by a label which consists of a vector of the exponents of the basic units. For example, a variable expressed in Newtons \((kg \times m/s^2)\) is labeled \([1, 1, -2]\). The requirements on the label of a subtree is defined from its parent and sibling nodes. This implies that the initialization procedure may have to construct a subtree with any label (compound unit). A DimTransform function is defined and produces a terminal of the required units. Since no terminal exists for each possible unit, DimTransform might introduce non-physically meaningful constructs, precluding the physical relevance of the final tree. Therefore, an auxiliary fitness measure is introduced in order to favor trees with few calls to DimTransform.

\(^2\) For instance, in the example presented below, the force recorded during an experiment with the parameters \(x\) has the form \(F(x, t) = f_1(x) \times e^{f_2(x)t}\)
Type constraints are closely related to the strongly typed GP (STGP) proposed by Montana [Mon95] and extended by Haynes et al. [HSW96]. In STGP, a type label is associated to every terminal, argument, and return value. The initial population is created by restricting the random choices to terminals or functions having the appropriate type label. Crossover operates by swapping a subtree with another subtree of the same type, and mutation replaces a subtree with a random subtree of the same type. STGP does not address, however, the problem of the dimensional consistency of the expressions.

Formal grammars have been implemented in a GP system by Hörner [Hör96], with crossover and mutation using procedures that are similar to those of the STGP. However, Hörner's system suffered limitations from the difficulty of initializing valid parse trees, as was pointed out by Ryan [RCO98].

4 Dimensionalization through formal grammars

The new concepts presented in this paper are twofold. The first part consists of using grammar rules for incorporating dimensionality constraints into a GP framework. Second, the limitations of Grammar-Guided GP are broken down by a new initialization procedure based on a dynamic pruning of the grammar, in order to generate only feasible trees of prescribed derivation depth.

This approach is illustrated by a problem of mechanical behavior law identification. The elementary units involved are mass, length and time. The characterization of any compound unit as an \( n \)-tuple giving its exponent with respect to the elementary units is borrowed from [KB99]. The allowed compound units are specified by the user. The present study is restricted to integer powers of the basic units in the range \( \{-2\ldots2\} \). This excludes operators that return fractional units (e.g., the square root). The domain of allowed units therefore contains \( 5^3 = 125 \) possible combinations. A non-terminal symbol is defined for each allowed compound units, together with the corresponding derivation rules to express all the admissible ways of resolving this symbol. Such a large number of combinations makes necessary the use of an automatic grammar generator. It might be objected that the size of this grammar makes it unpractical for real-world applications. Indeed, its memory complexity is exponential with respect to the number of elementary units, but no extra housekeeping is devoted to the GP kernel for units management. Therefore, the computational cost of this approach is no larger than other grammar-guided GP systems, and the use of a standard GP engine is allowed with no internal modifications. For instance, the results presented in this paper use Hörner's GP kernel as a basic engine [Hör96].

The grammar generator builds up each production rule with all the dimensionally coherent derivations. For example, a non-terminal with units \( [i, j, k] \) can be replaced by the multiplication of two non-terminals with units \( [a, b, c] \) and \( [d, e, f] \) if and only if \( [a, b, c] \times [d, e, f] = [i, j, k] \). A similar rule applies to division, and addition/subtraction require both arguments to be of units \( [i, j, k] \). Experts have to provide the derivation rule associated to the start symbol \( S \), thereby encoding its expected units, and possibly the shape of the sought solu-
tion. The set of available variables and their units should also be provided. The procedure is described as follows:

**Begin Grammar Generation**

For each combination of units \([i,j,k]\) do
- Create the production rule \(<NT_{ijk}> := expr_{ijk}\)
  - with \(expr_{ijk} = + <NT_{ijk}><NT_{ijk}> | - <NT_{ijk}><NT_{ijk}>\)
  - \(expr_{ijk} = expr_{ijk} | <NT_{ijk} \times exp <NT_{000}>\)
- For each variable/constant terminal \(T_{i}\) with units \([i,j,k]\)
  - \(expr_{ijk} = expr_{ijk} \mid T_{i}\)
- For each pair of combinations of units \([a,b,c], [d,e,f]\) do
  - If \([a,b,c] + [d,e,f] = [i,j,k]\)
    - \(expr_{ijk} = expr_{ijk} \times <NT_{abc}><NT_{def}>\)
  - If \([a,b,c] - [d,e,f] = [i,j,k]\)
    - \(expr_{ijk} = expr_{ijk} \div <NT_{abc}><NT_{def}>\)
- End for each pair \([a,b,c], [d,e,f]\)
- End for each \([i,j,k]\)
- End procedure

5 Initialization of Bounded Depth Trees

The initialization procedure has to build up trees based on the provided grammar. A major difficulty arises with the dimensioned grammar since most derivation rules can not be resolved directly into a terminal. The fraction of terminal derivations can be so small that there is almost no chance for a random process to select a terminal symbol. This implies, as noted by Ryan [RC098], that the trees tend to be very deep. On the other hand, if the user specifies a maximum tree depth, the initialization proceeds by massively rejecting oversized trees. The problem is similar to what occurs in constrained optimization whenever the feasible region is very small.

Some mechanisms for controlling the derivation depth must therefore be incorporated in the initialization procedure. The proposed approach is intended to bound the initialization operator to the domain of dimensionally-feasible trees of depth equal or inferior to a prescribed value \(D_{max}\). During grammar generation, to each non-terminal symbol \(<NT>\) is associated an integer \(d(<NT>)\), giving the depth of the smallest tree needed to rewrite \(<NT>\) into terminal symbols. The depth associated to each terminal symbol (operators, variables and constants) is set to 1. The depth of each \(<NT>\), initially set to infinity, is recursively computed according to the following relations:

\[
d(<OP><NT_{a}><NT_{b}>) = 1 + \max(d(<NT_{a}>) , d(<NT_{b}>) ) \]
\[
d(<NT_{i}> ) = \min_{j} \{d(deriv_{j})\} \text{ for } <NT_{i}> = deriv_{1}|deriv_{2}|\ldots|deriv_{n};
\]
During the tree-generation phase, depth labels are employed in order to enforce the bound on tree size. Given a non-terminal node at a depth $D$ in a tree, and assuming a maximum tree depth of $D_{\text{max}}$, the remaining allowed depth $D_{\text{max}} - D$ is computed. The chosen derivation is randomly drawn among the subset of the derivations for which $d(<\text{NT}>)_r \leq D_{\text{max}} - D$. This way, it is impossible for the algorithm to engage into a path that has no fully terminal solution in less than $D_{\text{max}} - D$ steps, and by the way, all the generated trees are feasible.

6 Numerical Experiments

The test-case presented herein is a simplified real-world application where an algebraic law is expected to be found for modeling experimental data corresponding to the constitutive law of a material during an indentation test. Figure 1 presents a schematic view of the experimental setup. A hard indenter of a prescribed shape (usually conical or tetrahedral) is pressed against the surface of the material to be tested out. The experimenter records the reaction force $F$ along time $t$ and displacement $u$.

For simple constitutive laws, the analytical relations between force, displacement, and materials properties are well known [Joh87]. For complex constitutive laws, finite elements models allow one to simulate the material reaction force. However, this simulation is rather expensive (3 hours on an HP350 workstation). For ill-known materials, only experimental data are available. This pinpoints the need for a simple analytical model in the two latter cases.

![Image](image1.png)

**Fig. 1.** Experimental setup of the indentation tests

![Image](image2.png)

**Fig. 2.** Typical force vs time relations of numerical simulation

Examples have been generated according to random values of the material properties, and the material behavior has been computed with the finite element model. The examples are, by the way, noisy, due to the limitations of the numerical method: roundoff errors and modeling approximations. Typical results of simulated force–time relation are presented on Figure 2. From prior knowledge, this relation is expected, during the loading phase, to be of the form:

$$F(t) = Au^2e^{(Pt)}$$

(2)
where \( A \) and \( P \) are unknown functions of the materials properties. The available physical quantities and their associated units are presented on Table 1. Due to the noisy nature of the examples, it is not expected that GP, nor any other machine discovery algorithm, will find out a solution that exactly fits the data.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>mass</th>
<th>length</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E ) (Young's modulus)</td>
<td>+1</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>( K ) (viscous modulus)</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>( n ) (plasticity factor)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( S_y ) (yield strength)</td>
<td>+1</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>( u ) (displacement)</td>
<td>0</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td>( t ) (time)</td>
<td>0</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>( F ) (Indentation Force)</td>
<td>+1</td>
<td>+1</td>
<td>-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>4000</td>
</tr>
<tr>
<td>Max. number of generations</td>
<td>1000</td>
</tr>
<tr>
<td>Probability of Crossover</td>
<td>0.8</td>
</tr>
<tr>
<td>Probability of Tree Mutation</td>
<td>0.2</td>
</tr>
<tr>
<td>Probability of Point Mutation</td>
<td>0.8</td>
</tr>
<tr>
<td>Number of training examples</td>
<td>20</td>
</tr>
<tr>
<td>Number of independent runs</td>
<td>20</td>
</tr>
</tbody>
</table>

Machine Discovery experiments have been conducted with the GP parameters given on Table 2. The crossover consists of swapping two arbitrary subtrees from two parents, with a choice restricted to subtrees having a root node of the same type. Tree mutation consists of crossing over one individual with a random admissible tree. The point mutation replaces one terminal node by another terminal of the same type. This operator is analogous to a local improvement operator, and has been observed, for the present problem, to be less destructive than the tree mutation. Six grammars were devised and are described as follows:

1. **universal-non-dim**: The most general case, with no a priori knowledge or dimensional constraints. This grammar is equivalent to canonical GP:

   \[
   S := <NT> ;
   <NT> := <OP> <TER> <TER> | <OP> <NT> <NT> | <OP> <TER> <NT> | <OP> <NT> <TER> ;
   <OP> := + | - | * | ÷ | \exp ;
   <TER> := E | K | N | SY | u | t | 1 | 2 | 3 | 4; (\(=^N\))
\]

2. **[\(\text{Ae}^\text{Pt}\)]-non-dim**: A partial constraint on the shape of the tree is introduced by this second grammar:

   \[
   S := \exp <NT> <NT1> ;
   <NT1> := * <NT> t ;
   <NT> := <OP> <TER> <TER> | <OP> <NT> <NT> | <OP> <TER> <NT> | <OP> <NT> <TER> ;
   <OP> := + | - | * | ÷ ;
   <TER> := E | K | N | SY | u | t | 1.0 | 2.0 | 3.0 | 4.0 ;
\]

This grammar enforces two constraints on the search space: the highest-level (root) operator is necessarily an \( \exp \) operator, and this \( \exp \) operator is multiplied by an arbitrary expression (first argument) but exponentiates an expression multiplied by the time \( t \).
3. \([\text{Au}^2\text{e}^\text{Pt}]-\text{non-dim}\): The complete shape constraint \([\text{Au}^2 \exp(\text{Pt})]\) is now enforced in a way similar to the previous case.

4. \(\text{universal-dim}\): Dimensional constraints but no shape constraint. The solution is expressed in Newtons, so the start symbol is defined a priori as:

\[
S := <\text{NT}+1+1-2>;
\]

5. \([\text{Ae}^\text{Pt}]-\text{dim}\): Dimensional constraints plus the partial shape constraint of the second grammar.

6. \([\text{Au}^2\text{e}^\text{Pt}]-\text{dim}\): Dimensional constraints plus the complete shape constraint as in the third grammar.

Figure 3 presents the size of the search space computed as a function of the allowed derivation depth, with the universal grammar (case 1), and the dimensionally-constrained grammar (case 4). These curves show that in both cases, the number of solutions grows exponentially, but the search space can be reduced by several order of magnitude with the use of dimensional constraints.

Average best fitness value over 20 independent runs, and standard deviation are presented on Table 3, while the evolution of the average best fitness with respect to the number of evaluations is plotted on Fig. 4 for the non-dimensional grammars and on Fig. 5 for the dimensional grammars. Comparisons based on the number of evaluations are fair benchmarks since no significant variation in total computation time have been noticed between the grammars.

Figures 4 and 5 ask for two comments. First of all, giving the expected shape of the equation does not necessarily improve the results\(^3\). It partially does so in the case of non-dimensional grammars. But this might be due to the fact that the shape constraint prevents the search from being trapped in the same local

\(^3\) Note that the problem at hand is based on real data where the solution is actually unknown.
optimum the universal grammar always falls in, which causes the null standard deviation observed for this case. This local optimum corresponds to the function $F = t^2e^{2e^u}$. For the dimensional grammars, shape constraints are detrimental to the quality of the results in both cases. Second, the dimensional constraints appear to be clearly beneficial since the results obtained with dimensional grammars always supersede those obtained with untyped grammars, by an average of 6 standard deviations.

Table 3. Results

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Average fitness</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal-untyped</td>
<td>6.2236E+4</td>
<td>0.0E+0</td>
</tr>
<tr>
<td>[A exp(Pt)]-untyped</td>
<td>6.5762E+4</td>
<td>2.2E+3</td>
</tr>
<tr>
<td>[Au^2 exp(Pt)]-untyped</td>
<td>5.1194E+4</td>
<td>1.9E+3</td>
</tr>
<tr>
<td>universal-dim</td>
<td>3.1009E+4</td>
<td>5.8E+3</td>
</tr>
<tr>
<td>[A exp(Pt)]-dim</td>
<td>4.0089E+4</td>
<td>2.7E+3</td>
</tr>
<tr>
<td>[Au^2 exp(Pt)]-dim</td>
<td>3.6357E+4</td>
<td>3.4E+3</td>
</tr>
</tbody>
</table>

Fig. 4. Average best fitness for the three non-dimensional grammars

Fig. 5. Average best fitness for the three dimensional grammars

7 Conclusion

The innovations presented in this paper are twofold. First, a novel approach for the management of dimensionality constraints by the means of an automatic grammar has been presented. Second, the point of designing an admissible and still sufficiently diversified initial population has been addressed through dynamic pruning of the grammar, depending on the maximum tree depth allowed, and the current position in the tree. So far, the initialization step was a major limitation to the use of formal grammar for constraining a GP search space.
The main limitation of the presented approach is its dependence over a limited range of allowed units. Using fractional units can be made possible by the use of rational instead of integer numbers. This would allow the use of a broader range of operators (square root, powers,...), but would be equivalent to having twice as many basic units. Further research will be devoted to the simultaneous evolution of the grammar and the GP trees, in order to evolve grammars that facilitate the discovery of fitter individuals.

Acknowledgments

The authors acknowledge Helmut Hörner who developed an efficient basis for G3P research. Finite elements simulations data were provided by Nicolas Tardieu.

References


Polymorphy and Hybridization in Genetically Programmed Networks

Arlindo Silva¹, Ana Neves¹, Ernesto Costa²
Centro de Informática e Sistemas da Universidade de Coimbra

¹ Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco,
Av. do Empresário, 6000 Castelo Branco – Portugal
{arlinho, dorian}@dei.uc.pt
² Departamento de Engenharia Informática, Universidade de Coimbra, Polo II – Pinhal de Marrocos, 3030 Coimbra – Portugal
ernesto@dei.uc.pt

Abstract. In this paper we discuss the polymorphic abilities of a new distributed representation for genetic programming, called Genetically Programmed Networks. These are inspired in a common structure in natural complex adaptive systems, where system functionality frequently emerges from the combined functionality of simple computational entities, densely interconnected for information exchange. A Genetically Programmed Network can be evolved into a distributed program, a rule based system or a neural network with simple adjustments to the evolutionary algorithm. The space of possible network topologies can also be easily controlled. This allows the fast exploration of various search spaces thus increasing the possibility of finding a (or a better) solution. Experimental results are presented to support our claims.

1. Introduction

As artificial agents grow in intelligence and autonomy it becomes harder to design by hand the controllers to guide those agents. Evolutionary approaches to the synthesis of autonomous intelligent agents have been proposed mainly in the evolutionary robotics and artificial life communities. These approaches try to reduce the intervention of the agent designer by evolving a solution from a population of individuals using the principles of natural selection instead of more complex engineering techniques.

Evolutionary approaches to agent synthesis can be divided in three main groups according with the representation used for the individuals:

- Neural networks [4, 8, 10].
- Rule based systems [6, 7, 9].
- Computer programs [1, 2].

The choice of the most appropriate controller architecture for autonomous agents is the center of an ongoing discussion [2, 15] which will probably never end. But from it we can conclude that the use of very specific representations has obvious disadvan-
tages in the fact that when the approach fails to find a solution to a given problem, a lot of effort is needed to start over, using a new representation and corresponding approach. Even if a solution is found, it would sometimes be useful to have an easy way of trying other representation, to see if better solutions could be found in the corresponding search space.

In this paper we address the way the polymorphy in Genetically Programmed Networks (GPN), i.e. the ability to be evolved into individuals with dissimilar structures and topologies, can be used as a tool for the fast exploration of different search spaces.

Genetically Programmed Networks were initially presented as a new evolutionary approach to the synthesis of controllers for autonomous agents and showed promising results in benchmark problems [16,17,18]. They are the result of our search for an evolutionary algorithm centered on a polymorphic representation. This search has been, from the beginning, conditioned by three self-imposed constraints:

- First, the approach should be as independent from the problem as possible to reduce the designer intervention, thus incorporating the other evolutionary approaches main advantage.
- Second, it should provide support for the emergence of memory mechanisms, which we believe are essential for the agent’s autonomy and intelligence. This aspect of our approach has been discussed in [17].
- Third, the level of polymorphy in the representation should be enough to allow the evolution of controllers with the most common architectures in previous approaches, namely neural networks, rule based systems and computer programs.

These constraints have been fulfilled in GPN while maintaining the flexibility and simplicity characteristic of evolutionary approaches.

Our representation was inspired in natural complex adaptive systems, which, as noted in [11], are frequently composed of simple computational entities densely interconnected, with the connections allowing the flow of information between entities. This structure is recurrent in nature, being seen in our brain and immune system as well as in cities or ecosystems. The same model is also used in several computational systems, e.g. artificial neural networks and cellular automata.

It seemed to us that the flexibility of the underlying structure could provide us with a representation capable of sustaining the desired polymorphy. The way memory mechanisms are implemented in systems with this structure, by allowing delays in the connections used to transmit information, e.g. recurrent connections in neural networks, also suggested an elegant way to of fulfilling our second constraint.

Genetically Programmed Networks are presented in more detail in the second section of this paper. The evolutionary algorithm and the control of polymorphy are discussed in section 3. Section 4 is devoted to the presentation of experimental results to illustrate the polymorphic abilities of GPN. Finally, in section 5, we summarize our work and draw some conclusions.
2. Genetically Programmed Networks

We can follow the natural model that served us as inspiration, to describe the representation used for Genetically Programmed Networks. We will start by a description of the GPN genotype, followed by a discussion of its phenotype and the relation between them.

2.1. GPN Genotype

In GPN entities are represented by programs that completely codify their behavior, which is simply a set of computations. Entities are usually called nodes in a GPN.

Connections are represented by variables in the program that codifies an entity's behavior. When a program accesses one of these variables we say that a connection is being established between the entity associated with the program and the entity corresponding to the variable. This means we don't need an explicit representation for connections, thus avoiding the usual problems with variable length representations when there is the need to insert or remove a given connection.

We now need a representation for the programs. Instead of using the most usual representation in evolutionary algorithms, more specifically in classical genetic programming (GP) [13], which is an explicit syntactic tree-based representation, we chose to use a linear representation as suggested in [12]. This allowed us some savings in computational resources, i.e. memory and processor time, at the cost of a somewhat more complex implementation. Each program is simply a vector of integer indexes each one referencing a function, variable or constant in corresponding tables. As in GP, functions, variables and constants can be chosen from a priori defined sets. The differences between these sets and the ones used in GP will be discussed later.

We still have a syntactic tree but is no more an explicit one, being instead implicitly constructed when the indexes in the chromosome are recursively interpreted as functions, variables and constants.

The genome of a GPN individual is then nothing more than a sequence of programs codified as integer vectors. An individual will have \( n \) chromosomes, corresponding to \( n \) programs, each associated with one of \( n \) entities or nodes.

2.2. GPN Phenotype

In terms of phenotype, a GPN individual is a network of interconnected entities or nodes. Besides nodes and connections a GPN has two other components:

1. Inputs, which allow the GPN to access information about the environment. The agent, after gathering information with its sensors, is responsible for presenting the GPN inputs with the relevant information, preprocessed or not.

2. Outputs. The final computations of a GPN are copied to a set of outputs so that the agent can use them to choose an action to perform. The entities whose final computations are copied to the outputs are called external nodes, while the other ones are called internal nodes.
As already mentioned, the connections in a GPN are created when a variable cor­responding to the output of an entity is accessed by a program associated with other entity. Following the same principle, connections between inputs and entities are established when a variable corresponding to an input is used in the entity’s program. Two types of connection can be created:

1. **Forward connections** are established from an entity \( q \) to an entity \( p \) when the program associated with \( p \) uses a variable \( f_q \) which contains the output of the entity \( q \) in the same iteration. In the same way, a forward connection between an input \( m \) and an entity \( p \) is established when a variable \( i_m \) with the value of input \( m \) is used by the program associated with \( p \). Forward connections between entities correspond to functional relations, since the destination entity will use the result computed by the origin entity as a partial result in its own computations.

2. **Recurrent connections** are established from an entity \( q \) to an entity \( p \) when the program associated with \( p \) uses a variable \( r_q \) which contains the value computed by \( q \) in the previous iteration. Recurrent connections result in the delayed transmission of information, thus permitting the emergence of memory mechanisms in a GPN, i.e. they allow the formation of temporal relations between entities. This type of connection is not permitted between an input and an entity.

A special type of forward connections is assumed to always exist from external nodes to the GPN outputs. Its function is only to allow the instantaneous flow of the values computed by the programs associated with external nodes to the correspondent GPN outputs.

### 3. The Evolutionary Algorithm

Since a GPN is merely a sequence of programs it seems obvious that a form of genetic programming should constitute the ideal evolutionary algorithm to evolve a population of GPN individuals. In fact, the algorithm centered on Genetic Programmed Networks is a new flavor of distributed GP, with several differences from classical GP to deal with the particularities of our representation.

The most important differences deal with the choice of function and terminal sets for the programs and the inclusion of a new level of operators.

#### 3.1. Controlling Polymorphism

In opposition to GP the most important aspect in the choice of functions and terminals is not if they are the most suitable for the problem being solved, but if they allow the evolution of the type of controller we want.
To better control the polymorphy of our representation the function set is divided in two other sets:

1. The **root set** $R$, which has the functions that are allowed to be chosen as the root of the program's syntactic tree.

2. The **function set** $F$, which has all the functions that can be included in the program except at its root.

Internal and external nodes have different root, function and terminal sets, respectively $R_i, F_i, T_i$ and $R_e, F_e, T_e$.

Manipulating the terminal sets we can control the topology of the individuals, i.e. the space of allowed connection patterns that can be explored. Table 1 presents the terminal sets for three different topologies.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Terminal Sets</th>
</tr>
</thead>
</table>
| **Complete Topology** | $T_i = \{ i_1, i_2, ..., i_l, r_1, r_2, ..., r_m, r_{m+1}, r_{m+2}, ..., r_{m+n} \}$  
|                  | $T_e = \{ i_1, i_2, ..., i_l, f_1, f_2, ..., f_m, r_1, r_2, ..., r_m, r_{m+1}, r_{m+2}, ..., r_{m+n} \}$ |
| **One Layer Topology** | $T_i = \{ i_1, i_2, ..., i_l, r_1, r_2, ..., r_m, r_{m+1}, r_{m+2}, ..., r_{m+n} \}$  
|                  | $T_e = \{ i_1, i_2, ..., i_l, r_1, r_2, ..., r_m, r_{m+1}, r_{m+2}, ..., r_{m+n} \}$ |
| **Memory Topology**  | $T_i = \{ r_1, r_2, ..., r_m, r_{m+1}, r_{m+2}, ..., r_{m+n} \}$  
|                  | $T_e = \{ i_1, i_2, ..., i_l, f_1, f_2, ..., f_m \}$ |

Table 1. Terminal sets for three different topologies, assuming a network with $l$ inputs, $m$ internal nodes and $n$ external nodes.

In a complete topology every possible connection is allowed. This is the most flexible topology, but corresponds to the larger search space, since is the one that includes more terminals. In a one layer topology, forward connections are not allowed from internal nodes to external nodes. This topology is less flexible than a complete one, but produces the GPN with fastest execution, since all programs can be ran concurrently. Finally, a memory topology allows recurrent connections only to internal nodes, which act as a memory layer. This is the topology with the smallest search space, but is also the least flexible.

Manipulating the function and root sets, we can control the architecture of the GPN being evolved. When no particular restrictions are made, all function and root sets are equal and we say that distributed programs are being evolved. To evolve rule based systems an *if-than-else* function is the only member of the root sets, and its not allowed in the function sets. This way all evolved programs will be *if-than-else* rules. To evolve neural networks more significant changes are needed. Every terminal gains an associated randomly generated weight, the root sets can only have a transfer or activation function and the function sets only members are an *add* and a *subtract* function. Each evolved program is no more than the sum of weighted terminals passed through the activation function.

These examples maintained equal root and function sets for internal and external nodes. The networks evolved when this is true are called homogeneous networks. If
the sets are chosen differently for internal and external nodes, we can evolve heterogeneous GPN, e.g. neural networks with different activation functions in the internal and external nodes, and even hybrid controllers, e.g. neurons in the internal nodes and rules in the external nodes.

<table>
<thead>
<tr>
<th>Distributed Program</th>
<th>Rule Based System</th>
<th>Neural Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_i=F_e=R_i=R_e={\ldots}$</td>
<td>$R_i=R_e={\text{if-then-else}}$</td>
<td>$R_i=R_e={\text{transf}}$</td>
</tr>
<tr>
<td>$F_i=F_e={\ldots}$</td>
<td>$F_i=F_e={\ldots}$</td>
<td>$F_i=F_e={\text{add, subtract}}$</td>
</tr>
</tbody>
</table>

Table 2. Function and root sets for three different architectures, assuming a network with $l$ inputs, $m$ internal nodes and $n$ external nodes.

We used these examples since they correspond to the most common architectures in other evolutionary approaches, but it is obvious that with further manipulations other architectures (and topologies) could be used.

The functions and terminals that are not architecture or topology related are chosen to allow the computations that take place in the nodes and do not directly depend on the problem. Functions or terminals with side effects, common in GP, are not allowed in our approach, since they are highly problem dependent.

3.2. Operators

At program level we maintained the use of the three most important operators in GP, namely fitness proportional reproduction, tree crossover and tree mutation. A new level of GPN operators had to be introduced to control the program operators’ application strategies. In the experiments whose results we present here, three of these operators are used:

1. GPN reproduction: when a GPN is selected for fitness proportional reproduction, all the programs of the new individual are copied from the corresponding node in its parent.

2. GPN mutation: when a GPN is selected for mutation, the resulting offspring will be obtained by applying program mutation to every program in the parent individual.

3. GPN crossover: when two GPN are selected for crossover, the two resulting offspring will be obtained by applying program crossover to every pair of corresponding programs in the parents.

Other strategies are obviously possible, e.g. instead of always applying program crossover to each pair of programs, it could be applied with some chosen probability.
3.3. Evolving Genetically Programmed Networks

The evolutionary algorithm is an obvious instance of the generic evolutionary algorithm and similar to GP. After the creation of an initial population by randomly generating $m \times n$ programs, where $m$ is the population size and $n$ is the number of nodes in an individual, the main evolutionary cycle begins:

- All individuals are evaluated by being allowed to control the agent in the given task.
- A temporary population is constituted by selecting individuals with replacement from the population using a tournament selection strategy.
- GPN crossover is applied to 85% of the temporary population, GPN reproduction to 10% and GPN mutation to the remaining 5%, obtaining a new population of offspring.
- The population is completely substituted by the new population and a new cycle begins.

The cycle ends when some stopping condition is satisfied, usually when a solution is found or a limit generation is reached.

4. Experimental Results

To test the polymorphic abilities of Genetically Programmed Networks, we chose a benchmark problem with a fitness landscape considered hard for traditional GP. This is the well known "Ant in the Santa Fé Trail" problem, which had its first variant "Ant in the John Muir Trail" proposed by Collins [5]. In the version most used in GP an artificial ant has to follow a discontinuous trail of food pellets in a toroidal cellular world. The ant has only one sensor that informs it of the presence or not of food in the next cell and can choose from three actions: move forward, turn left while remaining in the same cell, turn right while remaining in the same cell. The ant can use a maximum of 400 actions to complete the trail.

Since Koza [13] applied GP to this problem for the first time, that it has been considered a GP hard problem, not because it is difficult to find a solution, but because GP compares poorly to other search procedures in the effort it needs to find one. This effort is the measure we use here for comparison and equals the number of individuals that must be evaluated so that a solution is found with a probability of 99% [13].

Langdon [14] discusses the difficulties GP finds in this problem in terms of its fitness landscape and presents some results on the effort needed by several GP and non-GP approaches to find a solution. We summarize these results in Table 3 where it can be seen that all GP based approaches need significantly more effort than simple hill climbing. The best results obtained by an evolutionary approach, also shown in Table 3, are the ones presented in [3] and use a form of evolutionary programming. But even these are only marginally better than the ones obtained with hill climbing.
Table 3. Effort needed by several GP, non GP and GPN based approaches to solve the ant problem.

We hoped that by applying GPN to this problem we could find architecture / topology pairs corresponding to search spaces more friendly to genetic search. To verify this, we made 18 groups of 200 experiences corresponding to 18 architecture / topology pairs, including some hybrid forms. The experiments and results in terms of effort are summarized in Table 3. All the evolved GPN had two inputs, with values 10 when food was ahead and 01 otherwise. We used 6 internal nodes and 3 external nodes connected to 3 outputs. The output with the largest value decided which action to perform. The program size was limited to 64 so that the maximum of 9×64 instructions would allow a fair comparison with the other approaches, which usually have a 500 instructions limit. The evolution of distributed programs used root and function sets with the functions and, or, not, equal, greater and if-than-else. To evolve rule systems the if-than-else function was removed from the function sets and made the only member of the root sets. Neural networks were evolved as described before.

From Table 3 we can see that our best approach, a neural network / memory combination, needed only to evaluate 39,000 individuals to find a solution with 99% probability, which is roughly one third of the best non GPN approach and less than
one fourth of the effort needed by hill climbing. All other combinations found solutions with less effort than the needed by hill climbing and only one pair performed worst that the best non GPN approach. This shows the aptitude of our representation and methodology to solve this particular problem. It also shows that our claims towards the polymorphy of GPN can hold and that that polymorph can really be a useful tool in finding more friendly fitness landscapes to search, when dealing with hard problems.

While most combinations obtained good results it can be seen that the neural network architecture and the memory topology are usually associated with the better ones. While this was unexpected for the architecture, since we didn’t had any clue on which should perform better, we cannot say the same for the topology since it is the one that uses less terminals while still allowing the emergence of the memory mechanisms needed to solve the problem. But it must be noted that other problems could possible need the flexibility of the complete topology or the faster execution of one layer GPN.

5. Conclusions

The usefulness of a polymorphic, problem independent representation for the evolutionary synthesis of controllers for autonomous agents was discussed. We proposed such a representation, called Genetically Programmed Networks, inspired in the structure of natural complex adaptive systems. We claimed that GPN could be used to evolve neural networks, distributed programs and rule-based systems with several topologies, and even hybrids of the listed architectures.

To support our claims, we presented the results of evolving GPN with 18 different pairs architecture / topology to solve a GP hard problem. The results show that GPN could be used to find instances of the general GPN representation corresponding to search spaces where our algorithm needed much less effort to find a solution than previous approaches.

We hope these results can be extended to other problems, allowing for the quick exploration of many promising search spaces, so that new or better solutions can be found, or at least that known solutions can be found faster.

6. Acknowledgements

This work was partially funded by the Portuguese Ministry of Science and Technology under the Program PRAXIS XXI.

References

1. W. Banzhaf, P. Nordin, and M. Olmer, “Generating Adaptive Behavior for a Real Robot using Function Regression within Genetic Programming”, Genetic Program-


Abstract. Committee machines are known to improve the performance of individual learners. Evolutionary algorithms generate multiple individuals that can be combined to build committee machines. However, it is not easy to decide how big the committee should be and what members constitute the best committee. In this paper, we present a probabilistic search method for determining the size and members of the committees of individuals that are evolved by a standard GP engine. Applied to a suite of benchmark learning tasks, the GP committees achieved significant improvement in prediction accuracy.

1 Introduction

Committee machines are learning methods that make decisions by combining the results of several individual learners. The learners may be neural networks, decision trees, or any other algorithms. Recently, several authors have presented methods for combining models learned by means of evolutionary algorithms [1–4]. The basic idea behind this approach is that evolutionary algorithms are population-based distributed search methods that produce a variety of individuals. The diversity of the individuals generated during the evolution provide a rich resource for building committee machines.

Most research on evolutionary methods for committee machines has focused on two main issues: finding combination weights and creating diverse individuals. Examples of combining methods include weighted averaging [5] and majority voting [6]. The diversity of structures of committee members is essential in building effective committee machines. However, little research has been done to determine the committee size and to choose the members of the committee.

In this paper, we present a probabilistic method that finds an optimal committee through evolution of individuals that represent committee members. We assume Poisson distribution on the number of committee members whose mean is adapted during the search. An advantage of this approach is that the search can be fast since it focuses on the promising regions of the best committees. The method is applied to genetic programming (GP) to solve classification problems.
The results of GP committees are compared with those obtained by the standard GP and decision tree algorithms. The committee selection method achieved significant improvement in predictive accuracy of the standard GP engine.

The paper is organized as follows. In Section 2, we review previous work on evolutionary methods for building committee machines. Section 3 presents the probabilistic method for building the best committee from individuals evolved by a standard GP engine. Section 4 reports experimental results on the problems taken from the UCI-repository data sets. Section 5 contains conclusions.

2 Previous Work on Building Committee Machines

The basic idea behind the committee machine approach is to fuse knowledge acquired by individual experts to arrive at an overall decision that is supposedly superior to that attainable by any one of them acting alone [7]. Committee machines can be built in two different ways. One is to use a static structure. This is known generally as an ensemble method. Here, the input is not involved in combining committee members. Examples include ensemble averaging [8] and boosting [9]. The other method for building committees is to use a dynamic structure. This includes combining local experts such as mixtures of experts [10]. Here input is directly involved in the combining mechanism that uses an integrating unit such as a gating network adjusting the weights of committee members according to input. Most studies on committee machines have been based on neural networks [8,11], decision trees [12], and statistical methods [13].

Some authors have used evolutionary algorithms for committee machines. Evolutionary algorithms generate a number of individuals during evolution. Most methods select the single best solution, discarding all remaining individuals generated during the evolution. However, the individuals evolved can be better utilized if they are combined to build committees. Opitz and Shavlik [1] presented the ADDEMUP method that uses genetic algorithms to search for a correct and diverse population of neural networks to be used in the ensemble. It has an objective function that measures both the accuracy of the network and the disagreement of that network with respect to the other members of the set. Yao and Liu [3] experimented with a variety of combination methods to integrate multilayer perceptrons that were evolved by evolutionary programming and a modified back-propagation algorithm. They also try to find a committee of variable size using a genetic algorithm. However, only the individuals in the final generation were used as candidates for the committee members. Neri and Giordana [14] introduced universal suffrage selection that chooses a suitable concept description to evolve partial concept descriptions as a whole. Here the concept description can be viewed as a committee member and universal suffrage selection may be regarded as a method for selecting committee members.

Zhang and Joung [2] presented the mixing GP (MGP) method that uses weighted majority voting to combine individuals evolved by genetic programming. It regards genetic programs as cooperating experts and makes the final decision on the basis of a pool of the best n experts from the population. Experi-
mental results show that MGP improves the robustness of genetic programming. It also has the effect of accelerating the evolution speed since MGP requires smaller number of generations than a standard GP to obtain a specified level of performance. MGP has recently been extended to CENTs [4], the committees of evolutionary neural trees, to improve predictive accuracy of individual neural trees evolved. Both in MGP and CENTs, the size of committees was fixed. However, the committee size influences the performance of committee machines as well as the members of the committee. In the next section, we present an evolutionary search method that determines the appropriate size and members of committees.

3 Building Committees by Probabilistic Evolution

We consider an evolutionary method that builds committee machines in two separate stages. In stage one, given an evolutionary engine, a pool of diverse and fit individuals are evolved and kept. In stage two, these individuals evolved are used to find the best size and members of the committees.

3.1 Evolving Individuals

The goal of stage 1 is to generate good candidates for building committees. The candidates should be as diverse as possible. Let $A_i$ denote an individual. We use symbol $A(g)$ to denote the population at generation $g$:

$$A(g) = \{A_i(g)\}_{i=1}^M,$$  

where $M$ is the size of the population. The populations generated during the maximum number $G$ of generations constitute the space of candidate individuals for committees:

$$A = A(0) \cup ... \cup A(G).$$  

Starting from a random population, individuals are evolved by using genetic operators such as crossover and mutation. New generations of populations are produced until some termination criterion is met.

To measure the fitness of individuals at each generation, a set $D$ of $N$ data items are given:

$$D = \{(x_c, y_c)\}_{c=1}^N,$$  

where $x_c$ is the input vector and $y_c$ is the desired output. $N$ is the total number of data items. The fitness of the $i$th individual is measured as the sum of errors and complexity of individual $A_i$:

$$F(A_i) = E(A_i) + \alpha C(A_i)$$  

$E(A_i)$ is usually measured by the misclassification rate or the sum of squared errors on the data set $D$. The parameter $\alpha$ is the Occam factor that controls the accuracy and complexity of individuals [15]. $C(A_i)$ is based on the number of nodes and depth.
3.2 Evolving Committees

The goal of stage 2 is to find an optimal committee, optimal in the sense of the size and members. We use the symbol \( V_k(g) \) to denote the \( k \)th committee at generation \( g \). The population \( V(g) \) at generation \( g \) of the evolutionary algorithm for evolving committees consists of

\[
V(g) = \{V_k(g)\}_{k=1}^K,
\]

where committee size \( m_k \) is variable for each committee \( V_k(g) \). In this paper, we limit the maximum size \( K \) to 100 since the best individuals for each generation are considered as candidates and the typical number of maximum generation is 100. The fitness of committee \( V_k \) is defined similar to the fitness of individuals as follows:

\[
R(V_k) = E(V_k) + \beta C(V_k),
\]

where \( \beta \) controls the tradeoff between accuracy and complexity of committees. The error \( E(V_k) \) of committee \( k \) is the total error on \( D \) made by the weighted average of committee members.

\[
E(V_k) = \frac{1}{N} \sum_{j=1}^{m_k} \sum_{c=1}^{N} (v_{kj}f_j(x_c) - y_c)^2.
\]

Here \( v_{kj} \) is the \( j \)th component of the combining weight vector \( v_k \) of the \( k \)th committee and satisfies the condition \( \sum v_{kj} = 1 \). \( f_j(x_c) \) denotes the output value of the committee machine for input vector \( x_c \). If the type of the problem is classification, an indicator function can be used to count the number of misclassifications. In our approach, we use the generalized ensemble method (GEM) as the combining method [16]. The complexity of committee \( k \) is defined as the committee size \( m_k \) divided by training set size \( N \):

\[
C(V_k) = \frac{m_k}{N}.
\]

If all the individuals over the whole generations are considered as candidate members for committees, the number of combinations for the committee is very large. We consider only the best individual \( A_{\text{best}}(g) \) at each generation as the candidates. This is reasonable since a good fitness is a minimum requirement for good committee members. To maintain the diversity of individuals, we do not use elitist selection. Let \( B \) be the set of the best individuals:

\[
B = \{B_g\}_{g=1}^S, \quad B_g = A_{\text{best}}(g).
\]

Here \( S \) is the maximum size of the pool. Each committee member \( B_i \) is selected with the probability

\[
P(B_i) = \frac{\exp \{F(B_i)/T\}}{\sum_{j=1}^S \exp \{F(B_j)/T\}},
\]

where \( F(B_i) \) is the fitness of the committee \( B_i \).
where \( T \) is the constant temperature that determines the randomness of the selection. Equation (10) says that the candidate with a high fitness is selected with a higher probability. In fact, the useful individuals for building committees may be the individuals found in early generations. This selection scheme does not exclude the possibility of selecting the candidates with low fitness.

### 3.3 Probabilistic Aspects of Committee Selection

The size of the search space for the optimal committee \( V^* \) is \( (2^S - 1) \) for pool size \( S \). This is a large number. The search time can be reduced by using a strategy that concentrates on the search space of high performance. Equation (6) expresses that the complexity term \( C(V_k) \) prevents the committee size from growing unnecessarily.

The search for optimal committees can be formulated as a Bayesian inference problem. Here, we show how the committee selection can be guided by probabilistic models. Let \( P(V_k) \) denote the prior probability of committee \( V_k \). Once we observe the data \( D \), the likelihood \( P(D|V_k) \) of the committee can be computed. Bayes rule provides a method for combining the prior and likelihood to obtain the posterior probability \( P(V_k|D) \) of the committees:

\[
P(V_k|D) = \frac{P(D|V_k)P(V_k)}{P(D)}. \tag{11}
\]

According to the Bayesian framework for evolutionary computation [17], this problem can be solved by an evolutionary algorithm. The objective here is to find the best committee \( V^* \) that maximizes \( P(V_k|D) \). Note that maximizing (11) is equivalent to maximizing its numerator since \( P(D) \) does not depend on \( V_k \). Note also that a committee machine can be parameterized as \( V_k = (v, m) \), where \( m \) is the number of committee members and \( v \) is the values for mixing parameters for the members. Thus, we have

\[
P(V_k|D) \propto P(D|V_k)P(V_k) \tag{12}
\]

\[
= P(D|v, m)P(v, m) \tag{13}
\]

\[
= P(D|v, m)P(v|m)P(m). \tag{14}
\]

In the experiments, we assume the normality of distributions for \( P(D|v, m) \) and \( P(v|m) \). Under these assumptions, it can be shown that minimizing the sum of squared errors (7) maximizes the posterior probability. We also assume that committee size \( m \) is distributed according to the following Poisson distribution

\[
P(m - 2) = \frac{\lambda^{m-2} \exp(-\lambda)}{(m-2)!}, \quad m = 2, 3, 4, \ldots, \tag{15}
\]

where \( \lambda \) is the average size of committees for the generation. The Poisson distribution turned out to be useful since it tends to sample smaller sizes of committees. The mean value \( \lambda \) was chosen for each generation with respect to the
fitness for the best committee with size $m$:

$$P(\lambda) = \frac{R(V_{\text{best}}^\lambda(g-1))}{\sum_{m=2}^{K} R(V_{\text{best}}^m(g-1))},$$

(16)

where $V_{\text{best}}^\lambda(g-1)$ is the best committee with size $\lambda$ at generation $g - 1$. Through this adaptation, the search for an optimal size focuses more on the regions where the fitter committees were frequently generated. Here $R(V)$ is the fitness of committee $V$ measured on the validation-set which is distinct from the training set used to determine the weight of each member.

4 Empirical Results

4.1 Experimental Setup

The performance of the committee selection method was evaluated on the UCI data [18]. Four different problems were chosen, i.e. breast cancer, Australian credit card, heart disease, Pima-indians diabetes. These have different characteristics in several aspects, including the number of attributes, the number of examples, the distribution of positive and negative examples, and mixed numeric/discrete attributes. We used genetic programming to evolve solutions to these problems. The performance was measured by a 10-fold cross-validation for each data set.

The parameters for genetic programming were: maximum number of generations = 100, population size = 200, selection method = ranking selection, reproduction rate = 0.01, and mutation rate = 0.01. In this experiment, we did not use elitism. If it is used, the same best individuals tend to appear every generation, reducing the diversity of the genetic pool for building committee members. We used the GP function set consisting of arithmetic operators, such as $+,-,\times,\div$, and $\geq$. These functions have also been used in [19] for solving classification problems.

The parameter values for building committees in the second stage were: maximum number of generations = 50, population size = 25, pool size = 100, temperature for selection probability $T = 0.01$, initial mean of Poisson distribution $\lambda = 10$, maximum number of committees = 20, Occam factor for fitness of committees $\beta = 0.1$. Here the pool size is the same as the number of generations in genetic programming.

4.2 Experimental Results

Figures 1 to 3 show the results for 100 runs on the diabetes data. Figure 1 shows the evolution of value $\lambda$ of the Poisson distribution as a function of generation. Figure 1(a) shows that $\lambda$ changes adaptively with respect to the distribution of the size of best committees. Here, the committee $V_{\text{best}}^m$, the best committee with size $m$ over the generations, does not mean the best committee in the final generation. The best committee appeared mostly in the middle of the evolution.
Fig. 1. Evolution of the best committee size (used as the mean value $\lambda$ of the Poisson distribution) as a function of generation: (a) result for one run, (b) result averaged over 100 runs.

Fig. 2. Accuracy and frequency as a function of committee size: (a) Accuracy of the best committee for a fixed size. (b) Frequency of committees when the best committee is selected by the proposed committee selection method.

In this example, $V_{best}^m$ appeared in generation 31 and $m$ was 5. In Figure 1(b), the curve shows a convergence of $\lambda$ to 5.

Figure 2 compares the performance of the presented method with that for not using committee selection. The left figure shows the performance by hill-climbing search with a fixed size. In the experiment, the number of iterations of hill-climbing was the same as the total number of generations multiplied by the size of the population in the step for evolving committees. In this particular experiment, the result shown in Figure 2(b) indicates that committee sizes of 2 to 7 have a higher accuracy than others.

Figure 3 plots the best fitness and the frequency for each size of the committee as generation goes on. The left figure shows the general tendency that the accuracy of each size gets higher as generation goes on. The right figure shows that in this problem the best committees concentrate around the small sizes for all generations. Figure 4 shows the average size of the best committee. Larger
Fig. 3. Evolution of committees: (a) Accuracy of the best committee vs. generation vs. committee size. (b) Frequency vs. generation vs. committee size.

Fig. 4. Average size of the best committee for different problems.

Table 1. Comparison of misclassification rates for C5.0, C5.0 Boosting, the standard GP, and the GP committees on the four UCI data sets. The values are averages of ten standard 10-fold cross-validations.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Data size</th>
<th>C5.0</th>
<th>Standard GP</th>
<th>C5.0Boosting</th>
<th>GP Committee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>699 10-fold</td>
<td>5.42± 4.1</td>
<td>4.05± 2.8</td>
<td>3.14± 3.2</td>
<td>3.55± 2.1</td>
</tr>
<tr>
<td>Credit</td>
<td>690 10-fold</td>
<td>13.63 ± 4.4</td>
<td>14.63± 4.7</td>
<td>13.34± 3.3</td>
<td>12.02± 4.0</td>
</tr>
<tr>
<td>Heart</td>
<td>270 10-fold</td>
<td>22.96± 8.7</td>
<td>19.25± 7.7</td>
<td>19.63± 9.3</td>
<td>17.30± 7.2</td>
</tr>
<tr>
<td>Diabetes</td>
<td>768 10-fold</td>
<td>23.69± 6.5</td>
<td>25.50± 5.1</td>
<td>24.73± 5.5</td>
<td>24.90± 4.7</td>
</tr>
</tbody>
</table>
committees were more effective for the heart problem, while smaller committees were effective for the cancer and diabetes problems. The size of the best committee was less than 7 on average. Table 1 compares the average generalization error for the decision tree learner C5.0, C5.0Boosting, standard genetic programming (GP), and the GP committee using the presented committee selection method. Here the baseline results by C5.0 are the values reported in [20].

As in [20], we evaluated the results by using ten standard 10-fold cross-validations. The standard GP achieved better performances for C5.0 in the two problems (breast cancer and heart disease) out of four. The GP committee improved the performance of the standard GP and achieved better results than C5.0 in three out of four. GP committee outperformed even C5.0Boosting in two cases out of four.

5 Conclusions

This paper describes a new approach to searching an optimal committee. This approach includes the characteristic that searches the optimal committee size probabilistically. It also has a mechanism for penalizing large committees to promote compact committees. Therefore, redundant members tend to be removed from the committee. Experiments have been performed on a suite of four problems from the UCI machine learning database. Our experimental results show significant improvement in generalization accuracy by selecting committee size and members. Compared to simple genetic algorithms or other evolutionary algorithms, a distinguishing feature of the probabilistic evolutionary search is the sequential sampling of the size and the members of the committee, which can be naturally implemented in the Bayesian framework for evolutionary computation. The scaling properties of the presented method for committee selection for even larger committee sizes remain to be studied.

Acknowledgments

This research was supported in part by the Korea Ministry of Science and Technology through KISTEP under grant BR-2-1-G-06, by KOSEF Grant 981-0920-107-2 and by BK21 Program.

References


Scheduling
Distributed Simulated Annealing for Job Shop Scheduling*

ANDREAS ALBRECHT\textsuperscript{1,2}, UWE DER\textsuperscript{3}, KATHLEEN STEINHÖFEL\textsuperscript{3}, AND CHAK-KUEN WONG\textsuperscript{2**}

\textsuperscript{1} Univ. of Hertfordshire, Dept. of Computer Science, Hatfield, Herts AL10 9AB, UK.
\textsuperscript{2} CUHK, Dept. of Computer Science and Engineering, Shatin, N.T., Hong Kong.
\textsuperscript{3} GMD - National Research Center for IT, Kekuléstr. 7, 12489 Berlin, Germany.

\textbf{Abstract.} In the paper, we investigate theoretical and practical aspects of distributed computing for simulated annealing algorithms applied to the problem of scheduling $l$ jobs on $m$ machines. Given $n = l \cdot m$, the total number of tasks, $O(n^2)$ processors and an upper bound $\Lambda = \Lambda(l, m)$ of the objective function, the expected run-times of parallelized versions of our heuristics [14] are $O(n \cdot \log n \cdot \log \Lambda)$ for the exponential cooling schedule and $O(n^2 \cdot \log^{3/2} n \cdot m^{1/2} \cdot \log \Lambda)$ for the hyperbolic one. For Markov chains of constant length, the results imply a polylogarithmic run-time $O(\log n \cdot \log(l + m))$ for the exponential schedule, where we employ $\Lambda \leq O(l + m)$, see Leighton et al. [10]. We implemented a distributed version of our sequential heuristics and first computational experiments on benchmark instances are presented.

\section{Introduction}

In the job shop scheduling problem, $l$ jobs have to be processed on $m$ different machines. Each job $J_i$ consists of a sequence of tasks $T_{i,1}, \ldots, T_{i,m}$ that have to be completed during an uninterrupted time period of length $p_{i,j} \in \mathbb{N}$ on a given machine $M(T_{i,j}) \in \{1, \ldots, m\}$. A schedule is an allocation of tasks to time intervals on machines and the objective is to minimize the overall completion time which is called makespan. This scheduling problem is NP-hard [5] and even hard to approximate [21]. The problem remains NP-hard when each job is processed on each machine only once. The complexity of parallel algorithms for special cases that are solvable in polynomial time are analyzed in [7, 8, 19].

Implementations of parallel algorithms for the job shop scheduling problem are discussed in [20, 12, 17]. In [12], results of two parallel branch-and-bound algorithms are presented. The implementations are running on a 16-processor...
system. The algorithms perform a parallel search for solutions, i.e., the branching operation of sequential algorithms is parallelized. For the FT10 problem, they obtain a speed-up of about 14.67 is obtained. However, branch-and-bound methods tend to an exponential increase of branching steps.

In the present paper, the underlying algorithmic method is simulated annealing and sequential versions of these algorithms have been studied earlier in [14]. The parallelized versions in [15] have been further analyzed and we obtained first computational results using a distributed implementation.

The calculation of longest paths in disjunctive graphs is the critical computational task of our heuristics [14]. Therefore, we are particularly interested in parallel algorithms for longest path computations in disjunctive graphs. Given \( O(n^3) \) processors and an upper bound \( \Lambda = \Lambda(l, m) \) for the length of longest paths, the expected run-times of parallelized versions are \( O(n \cdot \log n \cdot \log \Lambda) \) for an exponential cooling schedule and \( O(n^2 \cdot \log^{3/2} n \cdot m^{1/2} \cdot \log \Lambda) \) for a hyperbolic cooling schedule, where \( n = l \cdot m \) is the number of tasks.

For problems where each job is processed by a machine at most once and all \( p_{i,j} = 1 \), Leighton et al. [10] have shown that there is always a schedule of length \( O(M_{\text{max}} + J_{\text{max}}) \). Here, \( M_{\text{max}} = \max M_k \sum_{M(T,j)=k} p_{i,j} \) is the maximum machine load and \( J_{\text{max}} = \max_j \sum_j p_{i,j} \) the maximum job length. In our case, the result implies \( \Lambda \leq O(l + m) \) for the completion time of job shop scheduling problems with \( l \) jobs on \( m \) machines.

Since \( O(n^3) \) is an extremely large number of processors, the parallel run-time for this number is merely of theoretical interest. Therefore, we implemented our heuristics in a distributed computing environment by using the software package PROMISE. This tool consists of multiple levels, including the language extension PROMOTER [3, 4]. In the present scheme, the sequential versions of our heuristics were running on 12 processors in parallel with independently chosen initial schedules. Our first computational experiments on FT10 and LA36 - LA40 show that stable results equal or close to optimum solutions are calculated by the distributed implementation in a relatively short time.

2 Two Simulated Annealing-Based Heuristics

Simulated annealing algorithms are acting within a configuration space in accordance with a certain neighborhood relation, where the particular transitions between adjacent elements of the configuration space are governed by an objective function \( Z \). More information about simulated annealing and its application in diverse areas can be found in [1, 2]. Throughout the paper, we follow the notations from [2].

2.1 Configuration Space and Neighborhood Function

The configuration space, i.e., the set of feasible solutions of a given problem instance, is denoted by \( \mathcal{F} \). Minimizing the makespan \( \lambda(S) \) in a job shop scheduling problem with no recirculation can be represented by a disjunctive graph [13].
Feasible solutions correspond to acyclic directed graphs that are admitted by a disjunctive graph. We define \( G = (V, A, E, \mu) \) for the set of tasks \( V \) with an additional source (1) and sink (O). All vertices in \( V \) are weighted. The weight of a vertex \( \mu(v) \) is given by the processing time \( p(v) \), \( \mu(v) := p(v) \), \( \mu(I) = \mu(O) = 0 \).

The arcs in \( A \) represent the given precedences between the tasks. The edges in \( E \) represent the machine capacity constraints, i.e., \( \{v, w\} \in E \) with \( v, w \in T \) and \( M(v) = M(w) \) denotes the disjunctive constraint, and the two ways to settle the disjunction correspond to the two possible orientations of \( \{v, w\} \).

An orientation on \( E \) is a function \( \delta : E \rightarrow T \times T \) such that \( \delta(\{v, w\}) \in \{(v, w), (w, v)\} \) for each \( \{v, w\} \in E \). A feasible schedule corresponds to an orientation \( \delta \) on \( E \) \( (\delta(E) = \{\delta(e) \mid e \in E\}) \) for which the resulting directed graph (called digraph) \( D := G' = (V, A, E, \mu, \delta(E)) \) is acyclic. The makespan of a feasible schedule is determined by the length of a longest path from \( I \) to \( O \) in the digraph \( D \). The problem of minimizing the makespan therefore can be reduced to finding an orientation \( \delta \) on \( E \) that minimizes the length of \( \lambda(P_{\text{max}}) \).

The neighborhood of a solution \( S \in \mathcal{F} \) is defined by \( \eta : \mathcal{F} \rightarrow \varphi(\mathcal{F}) \). Van Laarhoven et al. [18] proposed a neighborhood function \( \eta_L \) that is based on interchanging two adjacent operations which are processed on the same machine and do belong to a longest path. The transition rule guarantees that the resulting schedule is feasible, i.e., that the corresponding digraph is acyclic:

**Theorem 1** [18] For each schedule \( S \notin \mathcal{F}_{\text{min}} \) it is possible to construct a finite sequence of transitions leading from \( S \) to an element of \( \mathcal{F}_{\text{min}} \).

We proposed in [14] an extension (called \( \eta \)) of the neighborhood \( \eta_L \) such that changing the orientation of a larger number of arcs is allowed within a single machine path. Our neighborhood includes \( \eta_L \). Therefore, Theorem 1 applies also to \( \eta \). Moreover, we defined in [14] a non-uniform generation probability of neighbors that depends on the number of longest paths only.

The acceptance probabilities \( A[S, S'] \), \( S' \in \eta(S) \subseteq \mathcal{F} \) are derived from the underlying analogy to thermodynamic systems is the following:

\[
A[S, S'] := \begin{cases} 1, & \text{if } Z(S') - Z(S) \leq 0, \\ e^{-\frac{ \Delta Z_{\text{max}} }{c(0)}} & \text{otherwise.} \end{cases}
\]

### 2.2 Sequential Run-Time Estimation

We employ two cooling schedules that are comprehensively analyzed in [14]. In a brief description we want to recall their main features. The starting “temperature” \( c(0) \) is defined by

\[
c(0) = \frac{ - \Delta Z_{\text{max}} }{ \ln(1 - p_1) }.
\]

where \( p_1 \) is a small positive value (see also [2]).

The decremental rule of the first cooling schedule is given by the simple relation \( c(t + 1) := (1 - p_2) \cdot c(t) \), where \( p_2 \) is a small value larger than zero.
The stopping criterion is related to expected number \( R_c(S) \) of trials that are necessary for leaving a given configuration \( S \). By using a suitable upper bound for \( R_c(S) \) one obtains
\[
L < e^{\Delta Z_{\text{max}}/c(t_{\text{fin}})}, \quad c(t_{\text{fin}}) < \frac{\Delta Z_{\text{max}}}{\ln L}.
\]

Let \( \chi(c) \) denote the expected ratio of the total number of processed trials and the length \( L_c \) at temperature \( c \). The number \( t_{\text{fin}} \) denotes the number of cooling steps, and we define the average acceptance rate by setting \( \bar{\chi} := 1/t_{\text{fin}} \cdot \sum_c \chi_c \). The number of steps \( t_{\text{fin}} \) reducing the parameter \( c(t) \) can be calculated from
\[
(1 - p_2)^{t_{\text{fin}}} \cdot c(0) = c(t_{\text{fin}}),
\]
i.e., the temperature decreases exponentially. The equation leads to
\[
t_{\text{fin}} \leq \left\lfloor \frac{1}{\ln(1 - p_2)} \cdot \ln \left( -\frac{\ln(1 - p_1)}{\ln L} \right) \right\rfloor.
\]
Thus, the number of cooling steps does not depend on the objective function.

Given the number of transition steps \( L \), the algorithm has to perform \( L \cdot t_{\text{fin}} \) accepted moves before the algorithm halts because of (3).

Theorem 2 For the first cooling schedule, the expected run-time is bounded by
\[
T_I \leq \frac{L}{\ln(1 - p_2)} \cdot \ln \left( -\frac{\ln(1 - p_1)}{\ln L} \right) \cdot T \cdot \bar{\chi},
\]
where \( T \) denotes a time bound for updating the objective function.

For the second cooling schedule, the control parameter \( c(t) \) is decreased in accordance with the following hyperbolic function:
\[
c(t + 1) := \frac{c(t)}{1 + \varphi(p_3) \cdot c(t)} = \frac{c(0)}{1 + (t + 1) \cdot \varphi(p_3) \cdot c(0)},
\]
where \( \varphi(p_3) \) is defined by \( \varphi(p_3) := \ln(1 + p_3)/(Z_{\text{max}} - Z_{\text{min}}) < 1 \).

Our stopping criterion is derived from the condition (cf. [2]):
\[
c(t) \cdot \frac{\partial \tilde{Z}_c}{\partial c} \bigg|_{c=c(t)} \leq \varepsilon \cdot \tilde{Z}_{\text{max}},
\]
i.e., the changes of the objective function are very small compared to the expected initial value of \( Z \) at \( c(0) \). In our specific case the condition leads after a series of transformations to the following inequality (cf. [14]):
\[
\frac{\ln |\mathcal{F}|}{c(0)} \cdot c^2 + c < \varepsilon \cdot Z_{\text{max}}.
\]
If we assume integer values for the processing times \( p(t) \), the minimum improvement of the objective function is lower bounded by 1. Hence, we can take
\( \varepsilon := 1 / Z_{\text{max}} \) as a lower bound for \( \varepsilon \). From (7), one can derive the following upper bound of cooling steps:

\[
 t_{\text{fin}} < \sqrt{-\ln(1 - p_1) \cdot \ln |F| \cdot \frac{Z_{\text{max}} - Z_{\text{min}}}{\ln(1 + p_3)}},
\]

The upper bound is related to the approximation (7). Finally, we obtain:

**Theorem 3** For a number of transition steps \( L, \varepsilon := 1 / Z_{\text{max}}, \) and the second cooling schedule, the expected run-time can be upper bounded by

\[
 T_{\text{I}L} \leq L \cdot \sqrt{-\ln(1 - p_1) \cdot \ln |F| \cdot \frac{Z_{\text{max}} - Z_{\text{min}}}{\Delta Z_{\text{max}}} \cdot \frac{T \cdot \bar{x}}{1 + p_3}}.
\]

The run-time is larger compared to the bound given in Theorem 2, but one has a better control of the final outcome because the objective function is explicitly used in this cooling schedule. We can take \( \Delta Z_{\text{max}} = O(n/m) \) and \( Z_{\text{max}} - Z_{\text{min}} = O(n) \). Furthermore, we have \( \ln |F| < n \cdot \ln n \).

### 3 Parallelized Optimization Heuristics

Since local search algorithms are inherent sequential by its iterative nature, we parallelize the most time-consuming part of our heuristics, namely the calculation of the longest paths.

Our basic parallel algorithm is an adaptation of the shortest path algorithm described in [6]. As an input we take the weighted adjacency matrix \( A_S \) of the directed graph that represents the solution \( S \) where \( A_S(u, v) = p(u) \) if \( (u, v) \in \delta_S(E) \), and \( A_S(u, v) = 0 \) otherwise. At the beginning, we set the following values in parallel on \( n^2 \) processors:

\[
 \text{length}(u, v) := A_S(u, v).
\]

We use the PRAM model, where a constant time for data access and arithmetic operations is assumed. Hence, we obtain for the pair \( [P; T] \) of processor number and time requirement

\[
 [P; T] = [n^2; O(1)].
\]

For the value of the makespan we assume a uniform upper bound \( \Lambda \) such that \( \lambda(S) \leq \Lambda(l, m) = \Lambda \). Let \( p_{\text{min}} \) denote the minimum of the processing times of all tasks, then we have a maximum number of vertices which participate on the makespan of \( \Lambda/p_{\text{min}} \). Therefore, to cover all nodes on the makespan we have to repeat only \( \log(\Lambda/p_{\text{min}}) \) recursive steps. Potential upper bounds \( \Lambda \) are discussed in Section 5.

We repeat \( \lfloor \log \Lambda \rfloor \) times in parallel for triples \((u, w, v)\) the two partial steps:

\[
\text{trans}(u, w, v) := \text{length}(u, w) + \text{length}(w, v);
\]

\[
\text{max}_{\text{dist}}(u, v) := \max\{\text{length}(u, v), \text{trans}(u, w_1, v), \ldots , \text{trans}(u, w_{\ell}, v)\}.
\]
The first operation computes the transitive step from \( u \) to \( v \) via the node \( w \). To each pair of nodes a particular processor is assigned. According to our computation model, the time costs are constant.

To maintain the information about the values of \( \text{trans}(u, w_i, v) \), we do need \( n^3 \) processors. The maximum of the \( n - 1 \) values is computed in \( \lceil \log n \rceil \) steps. For both steps together we obtain:

\[
[P; T] = [n^2 + n^3; \lceil \log n \rceil + O(1)].
\]

The values \( \text{length}(u, v) \) have to be computed at each step of the \( \lceil \log A \rceil \) iterations, i.e., the overall complexity is given by

\[
[P; T] = [O(n^3); O(\log n \cdot \log A)]
\]

We note that our procedure provides the information about the nodes belonging to a longest path: At each step of the \( \lceil \log A \rceil \) iterations we maintain the set of \( \text{length}(u, v) \) where the maximum is achieved. We obtain

**Lemma 1** For the PRAM model, the computation of a complete sequence of nodes belonging to the makespan can be performed in \( O(\log n \cdot \log A) \) time by using \( O(n^3) \) processors.

For the calculation of the number of longest paths, we proceed as in [14]. The complexity does not exceed the order of magnitude of the complexity stated in Lemma 1 for the longest path computation.

### 4 Complexity of the Parallelized Heuristics

For the first cooling schedule, the number of steps \( L \) at fixed temperatures is chosen in accordance with \( |\eta(S)| \leq O(n) \). The complexity of local computations is assumed to be constant, in accordance with the PRAM model. By using again the parameter \( \overline{\chi} \) from the average acceptance rate, we obtain

**Theorem 4** When \( O(n^2) \) processors are available, the parallel run-time of the exponential cooling schedule can be upper bounded by

\[
T^\text{par}_L \leq O(\overline{\chi} \cdot n \cdot \log n \cdot \log A)
\]

When \( L \) is taken as a constant value, the upper bound is polylogarithmic:

**Corollary 1** For \( L = O(1) \) the upper bound \( T^\text{par}_L \leq O(\log n \cdot \log A) \) is valid.

For the second, the hyperbolic cooling schedule the number of cooling steps \( t_{\text{fin}} \) is upper bounded by \( t_{\text{fin}} < O(n^2 \cdot \sqrt{\ln n}) \).

As for the first heuristic, the number \( L \) of transitions at the same temperature is taken equal to \( O(n) \). Hence, the number of steps where the parallel algorithm for longest paths has to be applied is upper bounded by \( O(n^2 \cdot \sqrt{m} \cdot \sqrt{\ln n}) \), when \( \Delta Z^\text{max} = O(n/m) \) is assumed. It remains to take into account the number \( O(\log n \cdot \log A) \) of parallel steps. Thus, we have for the second heuristic
Theorem 5 When $O(n^3)$ processors are available, the parallel run-time of the hyperbolic cooling schedule can be upper bounded by

$$T_{parr} \leq O(\sqrt{\Lambda} \cdot n^2 \cdot \sqrt{m} \cdot \log^{3/2} n \cdot \log \Lambda).$$

The complexity is still very high due to the large number of cooling steps which depends on $\ln |\mathcal{F}| < O(n \cdot \ln n)$. Again, for a constant number $L$ of steps at fixed temperatures we have

Corollary 2 If $L = O(1)$, then $T_{parr} \leq O(\sqrt{\Lambda} \cdot n \cdot \sqrt{m} \cdot \log^{3/2} n \cdot \log \Lambda)$.

5 Upper Bounds of the Makespan

As seen in the previous section, the length of the makespan affects significantly the run-time bound of our parallelized algorithms. From Leighton et al. [10] we know that there is always a schedule of length $O(l + m)$.

<table>
<thead>
<tr>
<th>SWV</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>M</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>LB</td>
<td>2983</td>
<td>2972</td>
<td>3104</td>
<td>2968</td>
<td>2885</td>
<td>2924</td>
<td>2794</td>
<td>2852</td>
<td>2843</td>
<td>2823</td>
</tr>
<tr>
<td>UB</td>
<td>3005</td>
<td>3012</td>
<td>3104</td>
<td>2968</td>
<td>2917</td>
<td>2924</td>
<td>2794</td>
<td>2852</td>
<td>2843</td>
<td>2823</td>
</tr>
<tr>
<td>$\lambda_{\max}$</td>
<td>25399</td>
<td>25526</td>
<td>25860</td>
<td>25017</td>
<td>24989</td>
<td>25333</td>
<td>24459</td>
<td>24671</td>
<td>25595</td>
<td>24617</td>
</tr>
<tr>
<td>$p(v)$</td>
<td>50.8</td>
<td>51.1</td>
<td>51.7</td>
<td>50.1</td>
<td>50.0</td>
<td>50.7</td>
<td>48.9</td>
<td>49.3</td>
<td>51.2</td>
<td>49.2</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>3048</td>
<td>3064</td>
<td>3104</td>
<td>3002</td>
<td>2999</td>
<td>3040</td>
<td>2936</td>
<td>2961</td>
<td>3072</td>
<td>2954</td>
</tr>
<tr>
<td>$\Delta$ in %</td>
<td>43</td>
<td>52</td>
<td>1.7</td>
<td>0.34</td>
<td>82</td>
<td>116</td>
<td>142</td>
<td>109</td>
<td>229</td>
<td>131</td>
</tr>
</tbody>
</table>

Table 1. The bound $\Lambda$ and its distance to the upper bound for some SWV instances.

In case that the task lengths are randomly drawn over a certain time interval as for most benchmark instances, we conjecture a tighter upper bound

$$(9) \quad \Lambda := \frac{\overline{p(v)} \cdot (l + m)}{2},$$

where $\overline{p(v)}$ denotes the average task length.

We verified the conjecture on famous benchmark problems which were introduced in [11] by Fisher and Thompson (FT), in [9] by Lawrence (LA), in [22] by Yamada and Nakano (YN), and in [16] by Storer et al. (SWV). For all instances, the conjectured upper bound was correct. The size of the instances varies from six jobs on six machines to fifty jobs on ten machines. For some of the benchmark problems, we provide in Tab. 1 the value of $\Lambda$ and its distance to the actual optimum value of the makespan or current upper bounds for unsolved problems.
respectively. By $\lambda_{\text{max}}$ we denote the trivial upper bound of the makespan, i.e., the summation over all task processing times. In order to compare our $\mathcal{A}$-Bound, see equation (9), with the best known solution for the corresponding problem, we indicate the difference $\Delta$ and the percentage of the difference.

6 Distributed Computing Implementation

The number of processors $O(n^3)$ is extremely large for real world applications. Therefore, we first implemented a version that employs the sequential heuristics in a distributed computing environment by using the software package PROMISE [3, 4].

PROMISE is a high-level, massively parallel programming environment for data-parallel operations/programs. It comprises an extension of the programming languages C/C++ for SPMD programs. The processes created by such a program perform parallel computations locally. PROMISE allows the user to formulate a parallel program in terms of application-specific concepts for an abstract machine rather than for a particular architecture. It handles low-level issues such as optimal data distribution, communication and co-ordination of the parallel threads.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FT10</th>
<th>LA36</th>
<th>LA37</th>
<th>LA38</th>
<th>LA39</th>
<th>LA40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Machines</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Optimum</td>
<td>930</td>
<td>1268</td>
<td>1397</td>
<td>1196</td>
<td>1233</td>
<td>1222</td>
</tr>
<tr>
<td>sequential</td>
<td>930</td>
<td>199</td>
<td>1268</td>
<td>8964</td>
<td>1397</td>
<td>14442</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on 12 processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>930</td>
<td>234</td>
<td>1268</td>
<td>4083</td>
<td>1397</td>
<td>5613</td>
</tr>
<tr>
<td>75</td>
<td>930</td>
<td>232</td>
<td>1268</td>
<td>6118</td>
<td>1397</td>
<td>5337</td>
</tr>
<tr>
<td>100</td>
<td>930</td>
<td>234</td>
<td>1268</td>
<td>5976</td>
<td>1397</td>
<td>4616</td>
</tr>
</tbody>
</table>

Table 2. Sequential runs compared to runs with 12 processors.

In our global distribution of our scheduling heuristics (written in C++), each processor starts its own simulated annealing process with an independent initial schedule. The initial schedules are calculated in accordance with conjectured upper bound 9. Since the annealing processes are individual for each processor, we can use the (sequentially) optimized code without any changes except the communication strategy. It determines when and how the processes exchange the best solution found so far. We present first computational experiments, were the communication strategy was chosen as follows. Each individual calculations was interrupted when the temperatures was $E$ times lowered by the simulated annealing process. Tab. 2 shows our results for $E = 50, 75, 100$. The computations were performed on a Linux PC-Cluster with 12 Athlons of 550 MHz, 128 MB RAM, and by using Myrinet.
We tested our implementation on six benchmarks. The LA36 - LA40 were introduced by S. Lawrence [9]. Especially LA40 is believed to be one of the most challenging instance among the known benchmarks. The our test set of benchmarks includes also the smaller, but notorious instance FT10 due to [11]. Moreover, we restricted our experiments to runs employing the hyperbolic cooling schedule (CS2). The parameter settings are taken from [14].

In Tab. 2 we compare our results computed by 12 processors with results we obtained by using the same parameter setting but on a single processors of the cluster. In all runs we obtained near or equal to optimum solutions. The emphasized numbers are the run-time in seconds. The distributed computations produced the same solutions in less time or improved the result compared to a sequential run. A speed-up could be achieved for all runs where an optimum solution was found. Therefore, we conclude that, in general, the speed-up does not depend on our parameter $E$.

7 Concluding Remarks

We investigated theoretical and practical aspects of distributed computing for job shop scheduling problem. We designed parallel algorithms for the calculation of longest paths in disjunctive graphs which is the critical computational task in the sequential version of our heuristics. We employ an upper bound $A = A(l, m)$ for the length of longest paths, where $l$ is the number of jobs and $m$ the number of machines. When Markov chains of constant length are assumed, our complexity analysis implies a run-time $O(\log n \cdot \log A)$ for an exponential cooling schedule on $n^3$ processors. The analysis of benchmark problems of size $10 \times 10$ to $50 \times 10$ led us to the conjecture $A \leq \bar{p} \cdot (l + m)$, where $\bar{p}$ is the average task length. An implementation of the parallel heuristics is impractical since it requires $O(n^3)$ processors. Therefore, we implemented the sequential heuristic that employs a hyperbolic cooling schedule in a distributed computing environment of 12 processors. The conjectured $A$-bound was used to produce initial schedules. Our computational experiments produced results equal or close to optimum solutions for the benchmark problems FT10 and LA36 - LA40 in a relatively short time. Compared to sequential runs, we obtained either speed-up in finding an optimum solution or better makespans. In forthcoming research, we will increase the number of processors and apply the implementation to large scale benchmark problems.

References

252


Anticipation in Dynamic Optimization: The Scheduling Case

Jürgen Branke\(^1\) and Dirk C. Mattfeld\(^2\)

\(^1\) Univ. of Karlsruhe, Institute AIFB, 76128 Karlsruhe, Germany
\(jbr@aifb.uni-karlsruhe.de\)

\(^2\) Univ. of Bremen, Dept. of Economics, 28334 Bremen, Germany
\(dirk@logistik.uni-bremen.de\)

Abstract. This contribution addresses the role of anticipation in evolutionary algorithms for dynamic optimization problems. Recent approaches have mainly focused on maintaining the population diversity as a warrant for the ability of tracking the optimum. In our paper, we show that it is also useful to anticipate changes of the environment by explicitly searching for solutions which maintain flexibility. Although this is a valid approach to all dynamic optimization problems, it seems particularly important for optimization problems where a part of the solution is fixed at each step. For the example of job shop scheduling, we suggest a measure of flexibility and show that much better solutions can be obtained when this measure is incorporated into the fitness-function.

Keywords: anticipation, dynamic stochastic job shop scheduling, dynamic optimization problems, evolutionary algorithm

1 Introduction

Many real world optimization problems are dynamic in the sense that they change over time. For such problems, the goal no longer is to find a single optimum, but rather to continuously adapt the solution to the changing environment. So far, the vast majority of research on applying EAs to dynamic optimization problems has focused on maintaining diversity in the population, in order to keep the Evolutionary Algorithms flexible enough to adapt to a changing environment. For a survey on research in that area, see [5]. In our paper, we stress the role of anticipation in dynamic environments.

Dynamic stochastic job shop scheduling problems with new jobs arriving over time are a special case of dynamic optimization problems [10]. Basically, following a suggestion by Raman and Talbot [9], this kind of problem may be solved by using a rolling time horizon and decomposing the problem into a sequence of deterministic scheduling problems: Whenever a new job arrives, a new scheduling problem is created, consisting of the backlog of operations not yet processed and the operations from the new job. That way, each created "sub-problem" can be solved independently by a "standard" EA as developed for the time-invariant case. In this paper, we argue that it is worthwhile to anticipate
the forthcoming arrival of new jobs. Since we know that only the front part of the evolved schedule will actually be implemented, while the remainder will have to be rescheduled after the arrival of a new job, we suggest to explicitly search for schedules that are flexible enough to allow easy adaptation after a new job has arrived. For that we propose a measure of flexibility, and show that much better solutions can be obtained when the fitness for all subproblems is modified to take the flexibility of solutions into account.

The general idea of guiding the search towards solutions that are "well prepared" for changes in the environment is similar to the idea underlying the search for robust solutions, i.e. solutions that show a good average performance under all possible future scenarios (c.f. [3, 4, 12, 14]). Usually, this is achieved by evaluating each individual under a sample of all possible future scenarios and taking the average performance as fitness. In [13], it was observed for the case of rescheduling after a machine breakdown, that robust solutions in terms of insensitivity to swapping neighbored operations on a machine allow better rescheduling than other solutions of similar quality. In our paper, we know that we are going to reschedule after the arrival of a new job. Thus, future flexibility rather than robustness is important.

For that purpose, we here suggest the explicit incorporation of a criterion into the fitness-function which focuses on the early utilization of machine capacity in order to warrant flexibility for future decisions.

The paper is structured as follows: Section 2 provides a brief introduction to dynamic stochastic job shop scheduling and also discusses the role of machine idle-times. Then, in Section 3, the idea of incorporating anticipation in the fitness function is explained in more detail. In Section 4, this approach is empirically evaluated. The paper concludes with a summary and some ideas for future work.

2 Dynamic Job Shop Scheduling

2.1 Decomposing Dynamic JSSPs

Job shop scheduling problems (JSSPs) can be divided into static problems, where all jobs could start immediately, and dynamic problems, where each job has an earliest start time\(^1\). If all jobs are known in advance, the problem is called static, while it is called stochastic, if new additional jobs become known over time. We will be looking here at dynamic stochastic JSSPs.

As noted earlier, the dynamism of the problem is usually treated following the approach of a rolling time horizon (cf. Raman and Talbot [9]). First, a scheduling problem consisting of all known jobs is solved. When a new job arrives at time \(t\), the part of the solution consisting of operations already started before \(t\) is fixed and a new problem is constructed, consisting of the backlog to be starting

\(^1\) Unfortunately, the terminology in scheduling and the EA community is incompatible. What the EA community would consider a dynamic, i.e. time-varying problem is called stochastic in the scheduling community. We hope that the reader will not be confused by us using both terms in this paper, depending on the context.
after time $t$ plus all the operations from the newly arrived job. The dynamic problem is thus decomposed into a series of static sub-problems. For solving each of the sub-problems a "standard" EA as developed for the time-invariant case can be used\(^2\). For that purpose, many specialized EA approaches have been developed differing in representation, genetic operators, and decoding strategies. For a survey, the interested reader is referred to e.g. [6]

Note that in the case of a dynamic rolling time horizon, two consecutive sub-problems are closely related. As has been shown in [2, 7], it is therefore helpful to transfer adapted successful individuals from one sub-problem to the next, thereby implicitly transferring some information on promising areas of the search space. That way, an EA may gain speed and sometimes even solution quality. Since this approach has proven to be successful, it will be used as basis for our experiments.

2.2 The role of schedule builders

To assess the quality of a schedule, usually regular measures are used, i.e. measures that decrease monotonously with decreasing completion times of jobs. Prominent examples are the minimization of the maximal job completion time (makespan), the minimization of the mean flow time, and the minimization of the mean tardiness. Without loss of generality this research is confined to the latter objective.

For regular measures of performance, shifting an operation to an earlier start time without delaying any other operation already scheduled will never decrease a schedule's quality. Therefore, it suggests itself to use a schedule builder which constructs the schedule by iteratively inserting one operation after the other at the earliest possible time. Following that idea, most EA approaches for JSSPs operate on a set of priorities and then use a schedule builder to generate a valid schedule. The sequence in which the operations are inserted into the schedule by the schedule builder is in that case based on the precedence constraints and the evolved priorities for all operations.

There are two basic forms of schedule builders: those that never leave a machine idle when there is an operation that could be processed (the resulting schedules are called non-delay schedules), and those that allow a machine to wait for an urgent operation, although there is another operation that could be processed immediately. It has been shown that the latter case may indeed be advantageous. For instance, waiting for an almost tardy job (which is about to complete processing at the preceding machine) can prevent this job from becoming tardy by introducing additional machine idle-time. The maximal reasonable time span to wait for is given by the smallest processing time among those operations already queued. If one would wait even longer, this least time consuming

\(^2\) Note however, that in order to model dynamic scheduling problems, release times of jobs and periods of non-availability for machines have to be considered. These modifications can be easily integrated into an EA, but prevent many other optimization methods proposed from being applied.
operation could be processed in the meantime, which would obviously be a waste of machine capacity. The resulting schedules are called active, i.e. no operation can be left-shifted (leap-frogging included) without deteriorating the objective function value.

More specifically, a schedule builder operates by iteratively considering machine $M'$ with the earliest possible starting time $t'$ of an operation. For non-delay schedules one of the operations queued in front of $M'$ is picked for dispatching which can start at $t = t'$, i.e. a machine is never kept idle when there is an operation that might be started. To produce active schedules, an operation is determined on $M'$ with a minimal possible completion time $c''$. Here, an operation is dispatched on $M'$, which can start in the interval $t' < t < c''$. Furthermore we can think of hybrid schedules, simply by considering the interval $t' < t < (c'' - t')\delta$ with $\delta \in [0, 1]$ defining a bound on the time span a machine is allowed to remain idle (cf. Figure 1, see also [11]).

From the above considerations it is obvious that the set of non-delay schedules is smaller than the set of active schedules and that we can scale the size of the set of hybrid schedules by means of $\delta$. It is known from literature, that there is at least one optimal schedule in the set of active ones whereas not necessarily an optimal non-delay schedule exists. Since active schedules allow for additional machine idle-time, for regular measures of performance there is evidence that non-delay schedules perform much better on average. To summarize, compared to active schedules the set of non-delay schedules is smaller and shows a better mean performance. However, it does not necessarily include the optimal solution.

In a recent paper [2], it has been shown that EA approaches using hybrid scheduling are remarkably successful. In general, it has been shown that the more complex the problem, the smaller should be the value of $\delta$. With a smaller value for $\delta$, an EA profits from searching a smaller search space of a superior mean solution quality. If, on the other hand, the EA potentials are not exhausted, improvements can be gained by extending the search space slightly.

3 Anticipation of Scheduling Events

By decomposing a JSSP into deterministic subproblems which are solved successively, the problem's dynamics are basically ignored. Transferring individuals from a previous sub-problem to the current sub-problem tries to learn from the past, but is still ignoring future events. In this paper, we propose to integrate an explicit form of anticipation into the algorithm, by maintaining a schedule's flexibility and suitability for rescheduling.
Fig. 2. Assuming the tardiness of both schedules being equal, our approach would favor the schedule depicted in Gantt-chart (b) because there, idle-time is preserved longer.

There are two underlying ideas of our approach:

1. The flexibility to adapt a schedule in order to integrate new jobs is largely defined by machine idle-times. Basically, idle-time is considered an asset that might be used later for scheduling newly arrived jobs, and that should not be wasted easily.
2. When a new job arrives, the front part of the schedule will be fixed permanently, while the backlog may be rescheduled according to future needs. Any idle-time occurring in the fixed schedule is lost permanently.

As a consequence, we suggest to explicitly penalize early idle-times, in addition to the original fitness, i.e. the mean tardiness of the schedule.

Consider the two schedules depicted in Figure 2. Both may have the same total tardiness (e.g. when the due date for both jobs is 9), but they differ in their distribution of idle-times. If a new job arrives at time step 5, the idle-time for schedule (a) has been wasted, while it might be possible to utilize the idle-time of schedule (b) now.

The time interval during which idle-time should be penalized depends on the inter-arrival time of jobs. The longer the inter-arrival time, the longer the period that will be fixed when the next job will eventually arrive, and the larger the danger to waste idle-time permanently. For our implementation, we decided to use twice the inter-arrival time as the maximum time span considered. Luckily, the approach does not seem to be very sensitive to changes in that time frame, such that a rough estimate of the inter-arrival time would be sufficient. Any idle-time during that period is weighted with a linearly decreasing function, because the earlier the idle-time, the larger the probability to lose it irrevocably.

In most practical cases anyway, rescheduling will be done in fixed periods, with all new jobs arrived since the last rescheduling being added at once. In that case, the inter-arrival time is known exactly.
As fitness for the individual $i$, we then use a combination of tardiness $t_i$ and idle-time penalty $p_i$, both values being normalized to the interval $[0..1]$, i.e.

$$f_i = \frac{t_i - \min_j \{t_j\}}{\max_j \{t_j\} - \min_j \{t_j\}} + \alpha \frac{p_i - \min_j \{p_j\}}{\max_j \{p_j\} - \min_j \{p_j\}}$$

with the parameter $\alpha$ being the weighting factor.

Note that the approaches of restricting the search space to non-delay schedules and of penalizing early idle-time are not entirely independent. Generating non-delay schedules will in most cases implicitly also avoid early idle-times, simply by never leaving a machine idle when it could be used. However, this is done on the basis of purely local, greedy decisions made by the schedule builder. Nevertheless, this observation might explain why for dynamic scheduling problems non-delay schedules seem to have an advantage over active schedules, while for static problems the opposite is true [8]. On the other hand, when penalizing early idle time, active schedules will be avoided implicitly (at least in the period during which idle-time is actually penalized) unless they are really better than alternative non-delay schedules. As we will show, the explicit benefits of both approaches can be combined by using a joined approach: some degree of non-delayed schedules to reduce the search space for the EA, plus penalizing early idle-times in order to maintain flexibility.

4 Computational Investigation

The simulation environment described in the following has been already widely used for simulating manufacturing systems, e.g. [15]. The inter-arrival times of jobs in the manufacturing system affect its workload, i.e. the number of operations in the system which await processing. The mean inter-arrival time $\lambda$ can be prescribed by dividing the mean processing time of jobs $\bar{P}$ by the number of machines $m$ and a desired utilization rate $U$, i.e. $\lambda = \bar{P} / (mU)$. We simulate a simplified manufacturing system by using the following attributes:

- The manufacturing system consists of 6 machines.
- Each job passes 4 to 6 machines resulting in 5 operations on average.
- The machine order of operations within a job is generated from a uniform probability distribution.
- The processing times of operations are uniformly distributed in the range of $[1,19]$ which leads to a mean processing time of $\bar{P} = 5 \cdot 10$.
- We generate exponentially distributed inter-arrival times with mean $\lambda$ for various utilization rates $U$.

Utilization rates of $U = 0.7$ represent a relaxed situation of the manufacturing system. A moderate load is produced by $U = 0.8$ whereas utilization rates of $U = 0.9$ produce an excessive workload.

Modeling the inter-arrival times by a Poisson process can lead to extreme deviations of the workload in different phases of the simulation run. Therefore
30 different simulations are performed consisting of 500 jobs each. We discard job 1 to 100 as well as job 401 to 500 from being evaluated in order to circumvent distortion effects [2]. Consequently, the following results are calculated as the mean tardiness of job 101 to job 400 averaged over 30 different simulations.

Since the arrival of each job constitutes a new dynamic problem instance, 500 EA runs are performed within one simulation. A schedule of such an instance is encoded by a permutation of all operations involved in the problem. For decoding, the schedule builder interprets the permutation as array of priorities. Whenever more than one operation could be dispatched, the one occurring left-most in the permutation is given priority, cf. [2]. The crossover operator PPX [1] is used, which aims at preserving precedence relations among operations. The mutation operator picks an operation and inserts it at an arbitrary position in the permutation. PPX is applied with 0.6 probability while mutation is performed with 0.1 probability. We use a population size of 100 individuals, fitness proportional selection, and generational reproduction with an elite of 1. The number of generations an EA is allowed to run is calculated as half of the number of operations involved in the problem. This way, larger problem instances are given a longer time to converge.

The goal of this investigation is to shed light on the improvements which can be expected from penalizing early idle-times. For each $U \in \{0.7, 0.8, 0.9\}$ experiments are performed by varying both $\alpha$ and $\delta$ from 0 and 1. We consider different workloads in order to investigate whether the incorporation of early idle-times into the fitness is load dependent. Different schedule builder ranging from non-delay with $\delta = 0.0$ to active with $\delta = 1.0$ are engaged

- because it is already known, that hybrid decoding can improve the quality of dynamic scheduling significantly. Hence, different $\delta$ values provide a basis of comparison for valuating a modification of $\alpha$.
- because interdependencies between $\alpha$ and $\delta$ may exist. Both, a small $\delta$ (by means of decoding) as well as a large $\alpha$ (by means of fitness penalty) will avoid early idle-times. Their interdependencies are not yet clear.

Table 1 lists the improvements in percent achieved against EA runs performed with active decoding ($\delta = 1.0$) and just tardiness considered as fitness ($\alpha = 0.0$). It can be clearly seen (for arbitrary $U$ and $\delta$) that any consideration of early idle-times of machines ($\alpha > 0$) yields significant improvements on the minimization of mean job tardiness. The improvements gained for active scheduling by taking different values of $\alpha$ into account are equal, if not superior, to those obtained by varying solely $\delta$ towards 0.0. This preliminary result impressively points to the competitiveness of our approach.

Appropriate combinations of $\alpha$ and $\delta$ yield further significant improvements up to 33% for $U = 0.7$, 26% for $U = 0.8$ and 20% for $U = 0.9$. Since the interdependencies of $\alpha$ and $\delta$ are seemingly rather small we can conclude that the impact of non-delay decoding on the efficacy of the fitness penalty term is negligible. A small $\delta$ narrows the search space by excluding solutions which make use of machine idle-times. Although this will affect also the existence of early idle-times within a schedule, obviously the effect of $\alpha$ remains undeterred.
Table 1. Improvement in percent against active scheduling for different settings of $\alpha$ and $\delta$. Three different machine utilization scenarios are considered.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\delta$ for $U = 0.7$</th>
<th>$\delta$ for $U = 0.8$</th>
<th>$\delta$ for $U = 0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.00 0.25 0.50 0.75 1.00</td>
<td>0.00 0.25 0.50 0.75 1.00</td>
<td>0.00 0.25 0.50 0.75 1.00</td>
</tr>
<tr>
<td>0.2</td>
<td>10 17 20 8 0</td>
<td>15 16 14 7 0</td>
<td>10 11 10 6 0</td>
</tr>
<tr>
<td>0.4</td>
<td>23 31 33 27 26</td>
<td>21 26 24 21 13</td>
<td>15 17 16 12 8</td>
</tr>
<tr>
<td>0.6</td>
<td>22 27 29 22 20</td>
<td>22 22 26 19 16</td>
<td>18 18 16 15 7</td>
</tr>
<tr>
<td>0.8</td>
<td>20 22 21 20 19</td>
<td>22 22 20 19 16</td>
<td>20 19 15 14 9</td>
</tr>
<tr>
<td>1.0</td>
<td>15 21 20 20 17</td>
<td>20 22 19 17 16</td>
<td>20 20 16 14 8</td>
</tr>
</tbody>
</table>

In order to determine whether a suitable $\alpha$ depends on the $U$ considered, we proceed as follows. For each $U$ we identify the $\alpha$ value for which the best result is achieved for each $\delta$ column separately. In case of tie, we take the smaller $\alpha$. As a mean of these measures we compute 0.20 for $U = 0.7$, 0.33 for $U = 0.8$ and 0.56 for $U = 0.9$. We recognize that $\alpha$ should be increased with increasing $U$. The higher the workload gets, the more the avoidance of early idle-times is desirable. The importance of the original measure of performance, i.e. the mean tardiness, is reduced by the need to occupy the machine capacity.

At the extreme ($U = 0.9$) the highest improvements are achieved at $\delta = 0.0$ and equal weighting of a) the penalty of early idle-times and b) the tardiness of jobs as the original measure of performance ($\alpha = 1.0$). In more relaxed conditions a higher relative weight of the tardiness term will be more appropriate. At the same time also $\delta$ might be increased in order to benefit from additional idle-times allowed. This consideration explains the best overall improvement of 33% for $U = 0.7$ with $\alpha = 0.2$ and $\delta = 0.5$. Fortunately, the results are quite insensitive to minor changes of the parameters $\alpha$ and $\delta$.

The percentage of improvement against active scheduling shows a seemingly decreasing efficacy with increasing $U$. Note, that the absolute improvements gained are even larger for $U = 0.9$ than for $U = 0.7$, i.e. the tardiness units gained are 33% $\approx$ 1000 for $U = 0.7$, 26% $\approx$ 2000 for $U = 0.8$ and 20% $\approx$ 4000 for $U = 0.9$. Summarizing, the results presented impressively show the validity of our approach, particularly because significant improvements can be achieved at negligible computational burden.

5 Summary and Conclusion

We have stressed the role of anticipation for optimization in dynamic environments and have shown that it is important to explicitly search for solutions that are flexible enough to be easily adapted to changes in the environment.

In particular, for the example of job shop scheduling, we have shown that flexibility can be gained by avoiding early idle-times and that an Evolutionary Algorithm can take that into account by simply penalizing early idle-times.
Our empirical tests yielded excellent results and clearly demonstrated the
effectiveness of our approach. Furthermore, we have examined and discussed
the interdependencies between penalizing early idle-times and restricting the
search space to non-delay schedules. As can be seen from our experiments, these
two approaches are largely independent and best results can be obtained by an
appropriate combination of both.

Altogether, we have been able to reduce tardiness by up to 33 % compared
to active scheduling without idle-time penalty. This is even more impressive
considering that the additional computational cost for our approach is negligible.

Note that the general idea of anticipation is not restricted to scheduling
problems, although it is particularly useful for problems where a part of the
solution is fixed, because these decisions can not be revised later.

There remain numerous avenues for future research. First of all, the effect of
the weighting of idle-times and the time interval considered should be examined
more closely. Then, the degree of anticipation could be extended, e.g. by incor­
porating predictions on the type of jobs to be expected. But most importantly,
the idea of incorporating anticipation into EAs for dynamic optimization prob­
lems should transferred to other classes of dynamic optimization problems, which
would require the development and testing of appropriate flexibility-measures.

References

for Scheduling Problems. In Voigt et al. (eds.): Proc. of PPSN IV, Springer, Berlin,
pp. 310-318
rithm. In Eiben et al. (eds.): Proc. of PPSN V, Springer, Berlin, pp. 119-128
Survey. Technical Report No. 387, Insitute AIFB, University of Karlsruhe, Germany
namic Job Shop Scheduling Problems. In Bäck T. (ed.): Proc. of the 7th Int. Conf.
on Genetic Alg., Morgan Kaufmann, San Mateo CA, pp. 481-489
Approach. EJOR, 69, pp. 187-199
with Application to Job Shop Scheduling. Man. Sci., 38, pp. 1495-1509
Abstract. Over the past three decades extensive search have been done on pure m-machine flow shop problems. Many researchers faced the Flow Shop Scheduling Problem (FSSP) by means of well-known heuristics which, are successfully used for certain instances of the problem providing a single acceptable solution. Current trends involve distinct evolutionary computation approaches.

This work shows an implementation of diverse evolutionary approaches on a set of flow shop scheduling instances, including latest approaches using a multirecombination feature, Multiple Crossovers per Couple (MCPC), and partial replacement of the population when possible stagnation is detected. A discussion on implementation details, analysis and a comparison of evolutionary and conventional approaches to the problem are shown.

1 Introduction

In general, the task of scheduling [9] is the allocation of jobs over time when limited resources are available, where a number of objectives should be optimized, and several constraints must be satisfied. In particular, in a simple flow shop each job is processed by a series of machines in exactly the same order.

Starting with the Johnson's exact algorithm for the two-machine makespan problem [8], over the past decades extensive search have been done on pure m-machine flow shop problems. Many researchers faced the Flow Shop Scheduling Problem (FSSP) by means of well-known heuristics which, are successfully used for certain instances of the problem providing a single acceptable solution. Current trends involve distinct evolutionary computation approaches providing multiple near-optimal solutions.

We assume a FSSP model, where each job is processed on all machines in the same order, each machine processes a job at a time, and each job is processed
in a single machine at a time. The operations are not preemptable and set-up
times are included in the processing times. Well-known algorithms implement­
ing conventional heuristics were selected to be contrasted against evolutionary
approaches. Palmer [11] proposed to sequence the jobs according to a slope
order index, giving higher priority to jobs with processing times tending to in­
crease from machine to machine. Gupta [7] defined another slope index by using
some properties of Johnson's rule for the 3-machine problem. Campbell et al.
[1] develop a heuristic based on the Johnson's rule applying it to m-1 artificial
2-machine problems creating several schedules from which a best one can be
chosen. Nawaz et al. [10] proposed an algorithm which builds the final sequence
by adding a new job to a partial schedule. At each step the best partial schedule
is found. Recent improvements in evolutionary algorithms include a multiplicity
feature, which allows multiple recombination on a pair of parents (MCPC). As
the flow shop problem is essentially a permutation schedule it is natural that
chromosomes be encoded as permutations. In this case adequate genetic oper­
ators, such as ordered (OX), cyclic (CX) or partial mapped (PMX) crossovers
should be used [16].

This proposal includes eight evolutionary approaches for a set of flow shop
scheduling instances with the makespan as objective. Latest variants consider
an implementation of the MCPC approach on a population of permutations,
where the best or a random selected offspring couple is inserted in the next
generation. Also, the inclusion of one or more 'good' individuals in the initial
population, generated by conventional heuristics, as proposed by Reeves [12], is
considered. When possible stagnation at a local optimum is suspected a partial
replacement of the population, acting as a macro-mutation operator, is applied
to provide genetic diversity. Next sections show a comparison of evolutionary
and conventional approaches to the problem.

2 Conventional Heuristics Considered

Now the conventional heuristics contrasted against the Evolutionary Algorithms
(EAs) are briefly described. In what follows we consider a flow shop with n jobs,
m machines and we denote with $p_{ij}$ the processing time of job $i$ on machine $j$.

**Palmer's Algorithm.** Here a slope order index is proposed to sequence the
jobs, based on the job's processing time. In this way priority is given to jobs
with processing times that tend to increase from machine to machine. The slope
index for job $i$ is given by:

$$ S_i = \sum_{j=1}^{m} (2j - m - 1) p_{ij}, \quad i = 1, 2, ..., n $$

and a permutation schedule is built by sequencing jobs in non-increasing order
of $S_i$.

**Gupta's Algorithm.** It is similar to Palmer's algorithm but the slope index
is defined as follows:
\[ s_i = \frac{a_i}{\min_{1 \leq k \leq m-1} \{p_{i,k} + p_{i,k+1}\}} \]  
where,

\[ a_i = \begin{cases} 
1, & \text{if } p_{i1} < p_{im} \\
-1, & \text{if } p_{i1} \geq p_{im} 
\end{cases} \]

**CDS Algorithm.** Campbell, Dudek and Smith proposed a heuristic, which is an extension of the Johnson’s algorithm. This algorithm first generates a set of \( m-1 \) artificial two-machine problems by combining the \( m \) machines into two groups (see Table 1). Then Johnson’s rule is applied to find \( m-1 \) schedules (one at each stage) and finally the best one among them is selected.

**Table 1. The \( m - 1 \) artificial two-machine problem**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Artificial Two Machines Problem</th>
<th>Combined Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group 1</td>
<td>Group 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( p_{i1} )</td>
</tr>
<tr>
<td>2</td>
<td>1,2</td>
<td>( m, m-1 )</td>
</tr>
<tr>
<td></td>
<td>( p_{i1}, p_{i2} )</td>
<td>( p_{im} + p_{i,m-1} )</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>( m, m-1, m-2 )</td>
</tr>
<tr>
<td></td>
<td>( p_{i1}, p_{i2}, p_{i3} )</td>
<td>( p_{im} + p_{i,m-1} + p_{i,m-2} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m-1</td>
<td>1,2,\ldots,m-1</td>
<td>( m, m-1, \ldots, 2 )</td>
</tr>
<tr>
<td></td>
<td>( p_{i1}, p_{i2}, \ldots, p_{i,m-1} ) ( p_{im} + p_{i,m-1} + \ldots + p_{i,2} )</td>
<td></td>
</tr>
</tbody>
</table>

At stage \( k \) the algorithm considers the artificial machine 1 as composed by the set \( \{1, 2, \ldots, k\} \) and the artificial machine 2 as composed by the set \( \{m, \ldots, m-k+1\} \). The processing times of each artificial machine are computed as the sum of the partial processing times.

**NEH Algorithm.** Nawaz, Enscore and Ham proposed a heuristic, which gives higher priority to jobs with higher processing time on all machines. The final sequence is built by adding a new job at each step and then constructs the best partial schedule until finding the best total one. The NEH algorithm can be sketched in the following stages:

1. Rank the jobs by decreasing order of the sums of processing times on all machines.
2. Take the first two jobs and schedule them minimizing the makespan.
3. For \( k = 3 \) to \( n \) do: insert the \( k^{th} \) job at a place, which minimizes the partial makespan.

**3 The Evolutionary Approach**

Different evolutionary approaches, using genetic algorithms were proposed to solve the FSSP [2], [12], [13], [16]. Most of them have the same representation
as a chromosome: a permutation of integers where each allele stands for a job identifier. Distinct variants are addressed to the selection mechanism, genetic operators, and the decision to include or not in the initial population an individual generated by some conventional heuristic (Reeves).

New trends in Evolutionary Computation also are addressed to multirecombination. In earlier works [4], [5], we devised a different approach: to allow multiple offspring per couple, as often happens in nature. In order to deeply explore the recombination possibilities of previously found solutions, we decided to conduct several experiments in which a number $n_1$ of crossover operation for each mating pair was allowed. The number of children per couple was fixed or granted as a maximum number and the process of producing offspring was controlled, for each mating pair, in order not to exceed the population size. The idea of multiple children per couple was tested on a set of well-known testing functions (De Jong functions $F_1$, $F_2$ and $F_3$ [3], Schaffer $F_6$ [14] and other functions). Best quality results were obtained allowing between 2 and 4 crossovers per couple, providing from 4 to 8 children. These effects were a consequence of a greater exploitation of the recombination of good, previously found solutions. But on the other hand, because better parents produce a greater number of offspring, those experiments also showed that in some cases, the method increased the risk of premature convergence due to a loss of genetic diversity. To overcome this problem, each time a couple generated a number of children only at most two of them, best or random selected, were inserted in the next population.

Another attempt to avoid premature convergence implies the replacement of a percentage of the worst individuals by new randomly generated individuals. This is done during the evolutionary process when possible search improvements are not perceived through the measure of the mean population fitness. The decision criterion establish to apply partial replacement each time the mean population fitness remains unchanged within a threshold $\epsilon$, along a predefined number $n_2$ of consecutive generations. The evolutionary process unconditionally finish when the maximum number of generations is achieved.

3.1 Experiments

Returning to FSSP, to contrast the behaviour of the evolutionary approaches against the conventional ones, we devised eight different EAs. All of them with a randomized initial population of 100 individuals, a maximum number of generations fixed at 2000 and probabilities of crossover and mutation set to 0.65 and 0.1, respectively. These parameter-settings were established after a long series of runs. Different genetic operators and selection methods were used. Partially Mapped Crossover (PMX), One Cut Point Crossover (OCPX), Random Exchange Mutation (RXM), Shift Mutation (SM), Proportional Selection (PS) and Ranking Selection (RS) were chosen for this set of trials. The corresponding EAs versions are described below:

- $EA_1$: Uses PS, PMX and RXM.
- $EA_2$: Uses RS, OCPX and SM.
- **EA3**: Uses RS, OCPX and SM. Additionally the initial population includes the insertion of a random number (from 1 to 4) of seeds, ‘good’ individuals generated by the different above-mentioned conventional heuristics.

- **EA4**: Uses PS, PMX and RXM. Additionally the initial population includes the insertion of a random number (from 1 to 4) of seeds.

- **EA5**: Uses PS, PMX and RXM. Includes multirecombination of each selected couple using MCPC, allowing a random number \( n_1 \) of crossovers (1 to 4) and random selection of two offspring to be inserted in the next generation.

- **EA6**: Uses PS, PMX and RXM. Includes multirecombination of each selected couple using MCPC, allowing a random number \( n_1 \) of crossovers (1 to 4) and selection of the best two offspring as those to be inserted in the next generation.

- **EA7**: Uses RS, OCPX and SM. Additionally the initial population includes the insertion of a random number (from 1 to 4) of seeds. Includes the partial replacement of the worst 90% individuals by new randomly generated individuals when after a number \( n_2 = 10 \) consecutive generations the mean population fitness remains unchanged within a threshold \( \epsilon = 0.01 \).

- **EA8**: Uses PS, PMX and RXM. Includes multirecombination of each selected couple using MCPC, allowing a random number \( n_1 \) of crossovers (2 to 4) and selection of the best two offspring as those to be inserted in the next generation. The partial replacement of the worst 90% individuals by new randomly generated individuals is done as in EA7.

All the conventional and evolutionary algorithms were tested for five Taillard's benchmarks [15] for the flow shop problem. Given \( n \) jobs and \( m \) machines we run the experiments for the ten instances of each of the following \((n \times m)\) problem sizes: 20x5, 20x10, 20x20, 50x10, and 100x5. For each instance a series of ten runs was performed.

To compare the algorithms, a measure of the error was considered as follows:

\[
E_{\text{best}} = \frac{\text{opt-val} - \text{best value}}{\text{opt-val}} \times 100
\]

It is the percentile error of the best found individual when compared with the known, or estimated, optimum value \( \text{opt-val} \). It gives us a measure of how far are we from that \( \text{opt-val} \). The same percentile error measure was used to determine the quality of the single solution provided by the conventional heuristics.

### 4 Results

Tables 2 and 3 show detailed results for the 20x5-problem size. Table 2 refers to mean \( E_{\text{best}} \) values obtained from the ten runs for each instance while table 3 refers to minimum \( E_{\text{best}} \) values.

Average mean and average minimum values of the performance variables through all instances are indicated in the last row of the corresponding table. By observing table 2 we can remark that for this problem size, average behaviour of all evolutionary approaches is better than that of the conventional approaches. Only for the first instance NEH is the best one. Moreover, table 3 shows that
evolutionary approaches reach optimal values for many instances in some runs (italic zero $E_{best}$ values). This goal is not attained by any conventional approach. Values in italic in the last row of tables 2 and 3, indicate that EA8 and EA1 are the best average performers, while EA8 and EA4 are the best in finding near optimal solutions though all instances.

Table 2. Mean $E_{best}$ values for 20x5 instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>EA1</th>
<th>EA2</th>
<th>EA3</th>
<th>EA4</th>
<th>EA5</th>
<th>EA6</th>
<th>EA7</th>
<th>EA8</th>
<th>Palmer</th>
<th>Gupta</th>
<th>CDS</th>
<th>NEH</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst-01</td>
<td>1.29</td>
<td>1.08</td>
<td>1.35</td>
<td>1.31</td>
<td>1.34</td>
<td>1.34</td>
<td>1.39</td>
<td>1.19</td>
<td>8.28</td>
<td>10.25</td>
<td>11.27</td>
<td>0.63</td>
</tr>
<tr>
<td>inst-02</td>
<td>0.34</td>
<td>1.02</td>
<td>0.66</td>
<td>0.46</td>
<td>0.31</td>
<td>0.47</td>
<td>1.57</td>
<td>0.19</td>
<td>5.89</td>
<td>1.55</td>
<td>4.78</td>
<td>0.44</td>
</tr>
<tr>
<td>inst-03</td>
<td>1.23</td>
<td>2.21</td>
<td>2.05</td>
<td>1.22</td>
<td>1.50</td>
<td>1.74</td>
<td>1.69</td>
<td>1.20</td>
<td>7.49</td>
<td>15.36</td>
<td>15.54</td>
<td>5.46</td>
</tr>
<tr>
<td>inst-04</td>
<td>1.00</td>
<td>1.66</td>
<td>1.14</td>
<td>0.97</td>
<td>1.04</td>
<td>1.31</td>
<td>1.37</td>
<td>0.97</td>
<td>14.31</td>
<td>20.19</td>
<td>9.67</td>
<td>3.63</td>
</tr>
<tr>
<td>inst-05</td>
<td>0.93</td>
<td>1.33</td>
<td>1.05</td>
<td>0.99</td>
<td>1.03</td>
<td>1.09</td>
<td>1.69</td>
<td>0.57</td>
<td>10.03</td>
<td>10.84</td>
<td>7.04</td>
<td>5.58</td>
</tr>
<tr>
<td>inst-06</td>
<td>1.10</td>
<td>1.82</td>
<td>1.54</td>
<td>1.13</td>
<td>1.39</td>
<td>1.19</td>
<td>1.67</td>
<td>0.75</td>
<td>12.47</td>
<td>11.55</td>
<td>9.79</td>
<td>2.76</td>
</tr>
<tr>
<td>inst-07</td>
<td>0.97</td>
<td>1.01</td>
<td>0.98</td>
<td>0.96</td>
<td>0.96</td>
<td>1.77</td>
<td>0.95</td>
<td>12.99</td>
<td>12.19</td>
<td>12.43</td>
<td>3.15</td>
<td></td>
</tr>
<tr>
<td>inst-08</td>
<td>0.35</td>
<td>1.44</td>
<td>0.98</td>
<td>0.50</td>
<td>0.41</td>
<td>0.49</td>
<td>1.33</td>
<td>0.31</td>
<td>6.97</td>
<td>16.92</td>
<td>11.53</td>
<td>2.40</td>
</tr>
<tr>
<td>inst-09</td>
<td>1.48</td>
<td>2.97</td>
<td>2.91</td>
<td>1.75</td>
<td>1.87</td>
<td>1.93</td>
<td>1.75</td>
<td>0.93</td>
<td>15.93</td>
<td>17.40</td>
<td>10.57</td>
<td>4.96</td>
</tr>
<tr>
<td>inst-10</td>
<td>1.29</td>
<td>1.44</td>
<td>1.19</td>
<td>1.21</td>
<td>0.79</td>
<td>1.16</td>
<td>1.47</td>
<td>0.42</td>
<td>10.92</td>
<td>9.66</td>
<td>5.05</td>
<td>3.88</td>
</tr>
<tr>
<td>Average</td>
<td>0.99</td>
<td>1.60</td>
<td>1.38</td>
<td>1.05</td>
<td>1.06</td>
<td>1.17</td>
<td>1.57</td>
<td>0.74</td>
<td>10.53</td>
<td>12.59</td>
<td>9.77</td>
<td>3.29</td>
</tr>
</tbody>
</table>

Table 3. Minimum $E_{best}$ values for 20x5 instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>EA1</th>
<th>EA2</th>
<th>EA3</th>
<th>EA4</th>
<th>EA5</th>
<th>EA6</th>
<th>EA7</th>
<th>EA8</th>
<th>Palmer</th>
<th>Gupta</th>
<th>CDS</th>
<th>NEH</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst-01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.62</td>
<td>0.62</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
<td>8.28</td>
<td>10.25</td>
<td>11.27</td>
<td>0.63</td>
</tr>
<tr>
<td>inst-02</td>
<td>0.00</td>
<td>0.51</td>
<td>0.44</td>
<td>0.07</td>
<td>0.07</td>
<td>1.09</td>
<td>0.00</td>
<td>5.89</td>
<td>1.55</td>
<td>4.78</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td>inst-03</td>
<td>0.00</td>
<td>0.64</td>
<td>1.57</td>
<td>0.00</td>
<td>0.64</td>
<td>0.65</td>
<td>0.64</td>
<td>0.64</td>
<td>7.49</td>
<td>15.36</td>
<td>15.54</td>
<td>5.46</td>
</tr>
<tr>
<td>inst-04</td>
<td>0.30</td>
<td>0.61</td>
<td>0.46</td>
<td>0.54</td>
<td>0.54</td>
<td>0.54</td>
<td>0.26</td>
<td>0.26</td>
<td>14.31</td>
<td>20.19</td>
<td>9.67</td>
<td>3.63</td>
</tr>
<tr>
<td>inst-05</td>
<td>0.56</td>
<td>0.64</td>
<td>0.64</td>
<td>0.24</td>
<td>0.08</td>
<td>0.08</td>
<td>0.82</td>
<td>0.08</td>
<td>10.03</td>
<td>10.84</td>
<td>7.04</td>
<td>5.58</td>
</tr>
<tr>
<td>inst-06</td>
<td>0.50</td>
<td>1.25</td>
<td>0.66</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.53</td>
<td>0.00</td>
<td>12.47</td>
<td>11.55</td>
<td>9.79</td>
<td>2.76</td>
</tr>
<tr>
<td>inst-07</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>12.99</td>
<td>12.19</td>
<td>12.43</td>
<td>3.15</td>
</tr>
<tr>
<td>inst-08</td>
<td>0.00</td>
<td>0.66</td>
<td>0.66</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.95</td>
<td>0.00</td>
<td>6.97</td>
<td>16.92</td>
<td>11.53</td>
<td>2.40</td>
</tr>
<tr>
<td>inst-09</td>
<td>0.24</td>
<td>1.86</td>
<td>2.03</td>
<td>0.00</td>
<td>0.56</td>
<td>1.13</td>
<td>0.4</td>
<td>0.00</td>
<td>15.93</td>
<td>17.40</td>
<td>10.57</td>
<td>4.96</td>
</tr>
<tr>
<td>inst-10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.27</td>
<td>0.36</td>
<td>0.36</td>
<td>10.92</td>
<td>9.66</td>
<td>5.05</td>
<td>3.88</td>
</tr>
<tr>
<td>Average</td>
<td>0.27</td>
<td>0.71</td>
<td>0.80</td>
<td>0.24</td>
<td>0.28</td>
<td>0.37</td>
<td>0.62</td>
<td>0.18</td>
<td>10.53</td>
<td>12.59</td>
<td>9.77</td>
<td>3.29</td>
</tr>
</tbody>
</table>

The following figures show average behaviour for all the FSSP problem sizes under either evolutionary or conventional approaches. Figure 1 indicates the average of the mean $E_{best}$ values over all instances for each problem size. For example, in figure 1 [a], for the 20x5-problem size all the points plotted in the
same indicated vertical position correspond to the last row of table 2. By observing figure 1 [a] we can remark that for any problem size, average behaviour of all evolutionary approaches is better than that of the conventional approaches. Conventional and evolutionary heuristics have a similar response to the problem set: instances of the 100x5-problem size became the easiest, while those of the 20x10 and 50x10 were the hardest (except for Palmer’s algorithm for which those of the 20x20-problem-size became the hardest).

Regarding conventional heuristic the best performing one was NEH with percentile error values ranging from 0.35% (in 100x5-problem size) to 5.00% (in 50x10-problem size), while the worst one was Gupta’s algorithm with mean percentile error values ranging from 5.81% (in 100x5-problem size) to 24.20% (in 20x10-problem size).

![Fig. 1. [a] Average of Mean Ebest values over all instances for each problem size under conventional and evolutionary approaches. [b] Average of Mean Ebest values over all instances for each problem size under evolutionary approaches](image)

Concerning evolutionary algorithms, (see fig. 1 [b]) best performers were EA7 and EA8. EA7 showed mean Ebest values ranging from 0.32% (in 100x5-problem size) to 2.81% (in 50x10-problem size) and EA8 obtained mean Ebest values ranging from 0.51% (in 100x5-problem size) to 3.30% (in 50x10-problem size). The worst one was EA2 with mean Ebest values ranging from 0.80% (in 100x5-problem size) to 4.77% (in 50x10-problem size).

Figure 2 indicates the average of the minimum Ebest values over all instances for each problem size. They are similar to figures 1 [a] and [b] differing mainly in the range of values and showing the same general behaviour. As conventional heuristics provide a single solution to the problem they are not discussed
Fig. 2. [a] Average of Minimum Ebest values over all instances for each problem size under conventional and evolutionary approaches. [b] Average of Minimum Ebest values over all instances for each problem size under evolutionary approaches.

here. Regarding evolutionary algorithms, best performers were again EA7 and EA8. EA7 showed minimum Ebest values ranging from 0.11% (in 100x5-problem size) to 1.71% (in 50x10-problem size) and EA8 obtained minimum Ebest values ranging from 0.18% (in 100x5-problem size) to 2.40% (in 50x10-problem size). The worst performer was EA2 with mean Ebest values ranging from 0.34% (in 100x5-problem size) to 3.29% (in 50x10-problem size).

Figure 3 shows the general average behaviour of the algorithms contrasted in this work through all problem-sizes. As it can be seen, best minimum and mean percentile errors (0.8% and 1.6% respectively) are found, in average, under multirecombination and partial replacement of the population when possible stagnation is detected.

5 Conclusions

The Flow Shop Scheduling Problem was faced by means of well known heuristics which, are successfully used for certain instances of the problem providing one acceptable solution. Current trends involve distinct evolutionary computation approaches. The present contribution contrasted the behaviour of conventional and evolutionary approaches. The later included newest recombinative methods and partial replacement of the population when needed. Results obtained on the selected test suite deserve the following observations:

- Conventional heuristics provides a single near-optimal solution to the problem while evolutionary heuristics provide a population of alternative opti-
Fig. 3. [a] Average of Mean and Minimum Ebest values over all problem sizes under evolutionary approaches. [b] Average of percentile error values over all problem sizes under conventional approaches.

- The best off the above considered conventional heuristics, NEH, could contend against the evolutionary approaches only in the easiest 100x5-problem size regarding average error, but fails in finding the best individual. The rest of them remain far of obtaining results of comparable quality in any problem size.
- Evolutionary algorithms applied to the FSSP showed improved performance when multiple recombination is applied joined to partial replacement to escape from local optima.

At the light of these results research will be oriented to continue searching for further improvements of evolutionary approaches including more advanced features of multirecombination [6] considering multiplicity of parents and crossovers.

Acknowledgements

We acknowledge the cooperation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis and the ANPCYT from which we receive continuous support.
References

GA Based on the UV-Structure Hypothesis and Its Application to JSP

Kokolo Ikeda¹ and Shigenobu Kobayashi¹

Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology { psyche@fe.dis.titech.ac.jp , kobayasi@dis.titech.ac.jp }

Abstract. Genetic Algorithms (GAs) are effective approximation algorithms which focus on "hopeful area" in searching process. However, in harder problems, it is often very difficult to maintain a favorable trade-off between exploitation and exploration. All individuals leave the big-valley including the global optimum, and concentrate on another big-valley including a local optimum often. In this paper, we define such a situation on conventional GAs as the "UV-phenomenon", and suggest UV-structures as hard landscape structures that will cause the UV-phenomenon. We propose Innately Split Model (ISM) as a new GA model which can avoid the UV-phenomenon. We apply ISM to Job-shop Scheduling Problem (JSP), which is considered as one of globally multimodal and UV-structural problems. It is shown that ISM surpasses all famous approximation algorithms applied to JSP.

1 Introduction

GA is an approximation algorithm which can find high quality solutions efficiently, by searching hopeful area intensively in the search space. However, if the population converges so early in searching process, GA may fail to find the global optimum ("the optimum" we call) and may converge to a local optimum. Big-valley structure is the well-known landscape structure proposed in [Boese 1994, 1995]. When several big-valleys exist in the search space, and each valley has its own eminent local optimum, we call these landscape globally multimodal.

It is very difficult for conventional GAs to find the optimum in globally multimodal problems, especially when local optimums are scarcely inferior to the optimum and are located far from it in the search space. We call these local optimums "far-locals" briefly. As the global multimodality is a characteristic probably common to harder optimization problems, it is desirable to design a new GA which considers the global multimodality.

In section 2, we consider why and how GAs fail to find the optimum by the global multimodality, and propose the UV-structure hypothesis which explains why and how. In section 3 and 4, we verify our suggestion briefly on a function optimization problem and JSP. In section 5, we propose Innately Split Model (ISM), which splits individuals to several groups, initializes each group locally, and restricts the crossover within intra-group. In section 6, we apply a GA based on ISM to JSP, and confirm its superiority to conventional approximation algorithms.
2 The UV-structure hypothesis

In this section, we propose the hypothesis which qualitatively explains phenomena that GAs fail to find the optimum by the global multimodality. When the landscape has 2 big-valleys, a U-valley which is broad and shallow, and a V-valley which is narrow and deep, containing the optimum, we call this landscape the "typical UV-structure". Applying conventional GAs to such a landscape, the following phenomenon is expected. As individuals of the U-valley evolve rapidly compared with ones of the V-valley, individuals of the V-valley will be weeded out gradually through the alternation process, even if enough individuals exist on the both valleys in early generations. In consequence, GAs search among the U-valley intensively, and fail to find the optimum.

In globally multimodal problems, it is often observed that GAs search intensively among a valley which doesn't include the optimum and fail to find the optimum as above. We define the phenomenon the UV-phenomenon. When the quality of individuals searching among the valley which includes the optimum ("opt-valley" we call) is inferior to the quality of individuals searching among valleys which have one of far-locals ("local-valleys" we call), the UV-phenomenon is caused by the alternation. Here we call the landscape which cause the UV-phenomenon the UV-structure, and propose 3 hypotheses of UV-structures.

1. Structures which affect the apparent hopefulness in early generations
   The average quality of individuals searching among the opt-valley, so to say the apparent hopefulness of the opt-valley, is worse than of local-valleys in early generations. In this case, GAs may abandon the opt-valley before the real hopefulness of the opt-valley is known.

2. Structures which affect the number of individuals in valleys
   When the opt-valley is located on an outland in the search space or is very narrow, fewer offsprings will be generated by crossover or mutation. If individuals searching among the opt-valley are relatively few, the probability of mating within the opt-valley will be small, so the speed of evolution in the opt-valley will be relatively slow. Consequently GAs may fail to find the optimum.

3. Structures which affect the speed of evolution
   The complexity of a valley will make much effect on the speed of evolution
of individuals searching among it. When the opt-valley is more complex to evolve than one of local-valleys, the quality of individuals searching among the local-valley will be superior to the opt-valley. This may cause GAs to fail to find the optimum.

3 The UV-phenomenon on function optimizations

We introduce $f_9$ function minimization problems here. These functions have simple structures, and we can control the shape of these functions easily by operating parameters. The landscape of an example is shown in Fig. 2.

$$f_9(x_1, x_2) := \sum_{i=1}^{2} \{4(x_i - [x_i] - 0.5)^2 + bB(x_i)\} + A_{x_1x_2}; \quad x_1, x_2 \in (0, 3)$$

$$B(x_i) := 1 - \left\{ \frac{1 + \cos(2\pi C_{x_1x_2}x_i)}{2} \right\}^{D_{x_1x_2}}$$

$$A_{x_1x_2}, C_{x_1x_2}, D_{x_1x_2} := \begin{cases} (0, c_{opt}, d_{opt}), & \text{if } [x_1] = r_1, [x_2] = r_2 \\ (a, c_{loc}, d_{loc}), & \text{otherwise} \end{cases}$$

Fig. 2. an example of $f_9 : a=1.5, b=0.1, c=6, d_{opt}=d_{loc}=4, r_1=1, r_2=0$

These functions have 9 big-valleys. One of them lies down relatively to others, and has the optimum with the value of 0 , and has 4 quasi optimaums, i.e. the closest local optimums to the optimum with the value of about $4/c^2$. And other 8 valleys have far-locals with $a$. In following experiments, we use as default parameters $a=0.005, b=1, c_{opt}=c_{loc}=16, d_{opt}=d_{loc}=4$. $f_{9c}$ is the function of $r_1=1$ and $r_2=0$, this has its opt-valley in the center. $f_{9s}$ is the function of $r_1=1$ and $r_2=0$, its opt-valley is in the side. $f_{9e}$ is the function of $r_1=0$ and $r_2=0$, its opt-valley is in the edge. And, $f_{9e+}$ is the function of $r_1=1, r_2=0$, and $d_{opt}=8$. In $f_{9e+}$, the opt-valley has narrower bays, so its apparent hopefulness in early generations is worse than others. $f_{9es}$ is with $c_{loc}=6$, far-locals are easily detected.

It is expected that we may confirm the hypothesis 2, the location of the opt-valley effects the difficulty, by comparing among $f_{9c}, f_{9s}, \text{and } f_{9e}$. Also we may confirm hypothesis 1, wrong apparent hopefulness of the opt-valley will cause
UV-phenomenon, by comparing $f_{9s}$ with $f_{9s+}$. In following experiments, we employ UNDX[Ono 97](offsprings 100) and DDA[Takahashi 99] which is known as a very effective GA for function optimization problems.

Fig3 shows a typical behavior of population with size 500, about a successful case at $f_{9c}$ (left) and an unsuccessful case at $f_{9e}$ (right). If the evolution goes well, individuals closer to the optimum will increase. So, these graphs show the variation per generation of the number of individuals which is closer than 0.03, 0.06, ..., 0.9 to the optimum. In the successful case, closer individuals increase favorably. In the unsuccessful case, closer individuals increase instantaneously till about generation 5, and decrease gradually. Though a few individuals try to evolve at about generation 30, finally all individuals in the opt-valley disappear. It is important to see that population converges not on closest local optimums but on one of far-locals, i.e. a local-optimum located far from the optimum.

![Fig. 3. variation of the number of individuals in some neighborhoods of the optimum: a successful case at $f_{9c}$ (left), an unsuccessful case at $f_{9e}$ (right)](image)

Table1 shows the result of following experiments. To judge difficulties of introduced functions, we tried various population sizes for each function. We call a trial 'success' if a solution better than $10^{-7}$ is found within 1000 generations, and 60 trials were done in each condition. We can see by the result, $f_{9c}$ is the easiest and $f_{9e}$ is the hardest among $f_{9c}, f_{9s}, f_{9e}$, and $f_{9c}$, this supports our hypothesis 2. And, $f_{9s+}$ is much harder than $f_{9s}$, this supports hypothesis 1. $f_{9s+}$ is harder than $f_{9s}$, this supports hypothesis 3. A characteristic of the crossover operator of GAs yields these results that the location of the opt-valley effects the difficulty. Although UNDX capably inherits statistics of parents such as the mean vector and the covariance matrix of the population[Kita 99], UNDX can't keep the higher distribution manner. Actually, though all 9 valleys contain 11% individuals when the initial population is uniformly generated, its offsprings fairly slant, center valley will have 20% individuals, one side valley 12%, one edge valley 8%. It is natural that, the more offsprings are generated in a valley, the more

<table>
<thead>
<tr>
<th>population size</th>
<th>$f_{9c}$</th>
<th>$f_{9s}$</th>
<th>$f_{9e}$</th>
<th>$f_{9s+}$</th>
<th>$f_{9s+}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>46/60</td>
<td>11/60</td>
<td>4/60</td>
<td>2/60</td>
<td>5/60</td>
</tr>
<tr>
<td>100</td>
<td>60/60</td>
<td>34/60</td>
<td>8/60</td>
<td>13/60</td>
<td>16/60</td>
</tr>
<tr>
<td>500</td>
<td>60/60</td>
<td>60/60</td>
<td>27/60</td>
<td>28/60</td>
<td>54/60</td>
</tr>
</tbody>
</table>
individuals can search among the valley. In these results, \( f_{9c} \) is harder than \( f_{9e} \).
By these experiments, it is shown that UV-structures can cause the UV-phenomenon. 12(or 13,14...) -dimensional Fletcher-Powell function is extremely UV-structured we detected, they will be shown in following journal paper.

4 The UV-phenomenon on JSP

JSP is known as one of the hardest benchmarks, and would have the global multimodality. Fig4 shows the relationship between the distance to a solution from the optimum and its makespan difference from the optimum about famous instance ft10 (from Fisher, Thompson ). Here the distance is defined as the disagree rate of orders of operations(left) and \( L_2 \)distance [Sakuma 2000] (right). Fig4 suggests that some eminent local-optimums exist and big-valleys are formed around them respectively. Actually, as the optimum makespan of ft10 is 930, ft10 also has an eminent local-optimum 938, which stays away from the optimum with 74\% disagree rate, 170 \( L_2 \) distance. These are enough big distances compared with the average distance between randomly generated two individuals, 83\% disagree rate and 231 \( L_2 \) distance. To conclude, JSP is considered as a globally multimodal problem.

![Fig. 4. relation between the makespan and the disagree rate(left) to the optimum, \( L_2 \) distance(right) in ft10](image)

Next we examine how the population converges in terms of the same expression as section 3. Fig5(left) shows the variation per generation of the number of individuals which is closer than 12,24,...,120 to the optimum by \( L_2 \) distance in a successful case of ft10. Closer individuals increased favorably as Fig3(left), and the optimum was found at the same time nearest individuals increased. In contrast, Fig5(right) shows the variation in an unsuccessful case of abz5, a harder instance than ft10. As closer individuals increased till generation 10, decreased gradually, and finally converged to far-locals. The search space and the landscape structure of JSP is very complicated to analyze. At the first, we try to know information about where the optimum and far-locals are located in the search space. We generated random solutions, and grouped them by \( L_2 \) distances to the optimum and to one of known far-locals. In the easier instance ft10, 52\% of them are closer to the optimum 930 than to a far-local 938. This means, the optimum 930 and a far-local 938 are on an equal footing. On the other hand
Fig. 5. variation of the number of individuals in some neighborhoods of the optimum: successful case at ftl0(left), unsuccessful case at abz5(right)

in the harder instance abz5, only 26% of them are closer to the optimum 1234, than to a far-local 1238. Average $I_2$ distance of randomly generated solutions to the optimum 1234 is 145, and 129 to the 1238. We can say by these data, the optimum 1234 is located on an outland in the search space relatively to the 1238.

Next, we tried to investigate the behavior of evolution of the opt-valley and a local-valley in abz5. Although ideally we want to gather "individuals in the opt(or local)-valley", it's very difficult to define or judge which individuals are in the opt(or local)-valley. Here we define "the opt(or local)-valley set" as "individuals which are closer to the optimum 1234 (or the far-local 1238) than a far-local(or the optimum) by $d$ in $I_2$ distance". We sampled 30 randomly generated individuals of the opt(or local)-valley set in each trial, and tried to find the optimum 1234 (or 1238). It is expected that the opt-valley set find 1234, and the local-valley set find 1238. We employed CCM[Ono 98] as alternation model, 20 offsprings in every crossover, and tried till 200 generations. Table 2 shows the probability randomly generated solution is grouped to the opt(local)-valley set, average makespans of the opt(local)-valley set, the number of times the opt-valley set find the optimum (and average CPU time to find), the number of times the local-valley set find the far-locals 1238 (and average time), and the number of times merged set of the both sets (population 60) finds the optimum.

<table>
<thead>
<tr>
<th>$d$</th>
<th>Generated rate: opt/local</th>
<th>Average makespan</th>
<th>on Opt</th>
<th>on Local</th>
<th>Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>10(abz5)</td>
<td>19.1%/59.8%</td>
<td>1529/1502</td>
<td>4/20, 28s</td>
<td>18/20, 14s</td>
<td>0/20</td>
</tr>
<tr>
<td>50(abz5)</td>
<td>9.0%/37.7%</td>
<td>1535/1497</td>
<td>10/20, 17s</td>
<td>17/20, 11s</td>
<td>2/20</td>
</tr>
<tr>
<td>50(abx5)</td>
<td>2.9%/17.4%</td>
<td>1531/1478</td>
<td>16/20, 6s</td>
<td>19/20, 7s</td>
<td>6/20</td>
</tr>
<tr>
<td>30(ftl0)</td>
<td>32.0%/18.4%</td>
<td>1210/1204</td>
<td>11/20, 20s</td>
<td>14/20, 13s</td>
<td>9/20</td>
</tr>
</tbody>
</table>

The difference of generated rates shows that individuals closer to the optimum are rarer than to the local 1238 (this supports hypothesis 2). As opt(or local)-valley sets can find frequently the optimum(or 1238), our definition of the set of the opt(or local)-valley is enough adequate. The fact that the average quality of the opt-valley set is fairly worse than of the local-valley set, supports hypothesis 1. We can observe the UV-phenomenon, because merged sets seldom find the optimum though opt-valley sets can find frequently. In other words,
this instance is hard not because of the difficulty in finding the optimum in the opt-valley, but because of the UV-phenomenon that far-locals pull individuals in the opt-valley, i.e. because of the crossover between individuals in the opt-valley and individuals in a local-valley. Incidentally, we couldn’t observe some significant differences of evolution speed between the both valleys. ft10 is not(or less) UV-structured, and experiments with merged sets show that UV-phenomenon is seldomly observed in ft10.

By these results, we can say that the UV-phenomenon surely occurs in JSP, and hypothesis 2 and/or 1 is influential at abz5.

5 Innately Split Model

All results suggest that UV-structures will cause UV-phenomenon, and it is UV-phenomenon that makes problems difficult. In this paper, to avoid the UV-phenomenon, we focus on countermeasures the phenomenon that individuals in local-valleys pull individuals in the opt-valley by the crossover operator. Here, we alter the framework of conventional GAs, ”all individuals search among the whole search space together, and crossover wherever they are”, and introduce Innately Split Model in which several groups search among small areas independently.

// Innately Split Model //

1. Individuals are separated into G groups. Each group consists of N individuals.
2. Each group is initialized within a small area respectively.
3. In the crossover phase, parents are selected from the same group. The alternation model in each group is not restricted to be fixed.
4. When two groups seem to search among the same area, kill one group and generate a new group.
5. When a group stand still long with bad quality, kill it and generate a new group.

(N.B. In 2, for example, a group is formed with solutions once mutated from one randomly generated solution. Of course, desirable extent depends on problems. In 3, we employ CCM. 4 and 5 are not essential but optional items to accelerate the evolution. )

We are convinced that, it is imperfect to search only among the most hopeful area in harder problems, it is necessary to search among several hopeful areas simultaneously. By ISM, it is expected that each group can search among close region to the initialized area, without being deceived by the apparent hopefulness of local-valleys located far from the area now they are searching. As another countermeasure against the UV-phenomenon, clustering of population seems to be a natural idea to employ. However, we consider it is difficult to avoid the UV-phenomenon especially caused by hypothesis 2 in early generations only with clustering. Moreover, clustering technique is very awkward in general.
6 Apply ISM to JSP / Experiments

In this section, we apply ISM to JSP which is a very hard benchmark. There enumerated all configurations of our experiments. Previous experiments on JSP were done with the same condition except for the alternation model.

- GTb method (see Appendix) is also used as the enforce method in addition to GT method [Gifler 60]. GTb is a stronger enforcement for the quality of individuals.
- LR-method [Yamada 96,97] is used. By this method, we can expect for better enforcement, and the absorption of bad biases caused by the enforcement.
- Couple Weeding Mutation (CWM) is introduced. This is a new mutation system that, if and only when parents which have the same quality are selected, the mutation is used instead of the crossover. The reason why we introduce CWM bases on the idea and experiences on JSP, if parents which have the same quality are selected, they are probably ordinary, maybe are local-optimums, so go for nothing for the benefit of the search.
- Job-order based mutation [Ono 98] is used as the mutation operator.
- The crossover operator is JOX[Ono 96]. Though more effective crossovers MSXF [Yamada 96,97] and EDX[Sakuma 2000] were proposed, they contain Simulated Annealing(SA) algorithm with CB neighborhood unique to JSP. As it is not the crossover operator but the alternation model that this paper deals with, we employed JOX, a very simple crossover.

We applied our proposal method to the well-known large-size instances of JSP, ten tough problems (abz is from Adams,Balas,Zawack; la is from Lawrence). We set G=40, N=30, and stopped the algorithm if the GA found the optimum or 50 hours past. As ISM contains many inner trials structurally, it is better to continue one trial with long time. YN96(SA) [Yamada 95,96], the best algorithm proposed before now, set the limit time 3 hours, and did 20 trials. So from the viewpoint of total experiment time, our limit time 50 hours is not too long. All experiments are done on Windows98(Celeron 400MHz), coded by Delphi4 (Object Pascal).

Table 3 shows, name of instances, the optimum confirmed by Branch And Bound method or Upper Bound(UB), best solutions of ISM, time at ISM detected them, and best solutions of CCM with even configuration. ISM found the optimum/UB in 6 instances. And, 4 of them were found comparatively in a matter of hours. We can insist that ISM is perfectly superior to CCM.

Some GA/GLS [Ulder 90] methods were applied to JSP. We compare our results to other GA/GLS approaches (EDX00[Sakuma 2000], YN97[Yamada 96,97], Ono96[Ono 96], Matt[Mattfeld 94]). Comparison results are also summarized on Table 3, we can see the proposal method is superior or equal to all methods in all instances. We also compare our results with famous approximation algorithms. Nowi is the taboo search algorithm proposed in [Nowicki 93], Aarts is the SA method proposed in [Aarts 94], Appl is a shuffle algorithm method proposed in [Applegate 91]. And YN96 is the improved SA method proposed in [Yamada 95,96], which used to be the best approximation approach applied to JSP. Comparison results are also summarized on Table 3. We can see the proposal method is almost superior to all methods in all instances.

What is more, ISM found UB solutions in abz9(679),la29(1153) at additional experiments. Furthermore, we picked up swv1, swv6, yn4 as more difficult or large-size instances, and applied ISM with 50 hours. ISM found 1424
Table 3. Results of experiments in 10 tough problem using ISM, comparison with CCM and other approaches

<table>
<thead>
<tr>
<th>instance</th>
<th>Result</th>
<th>time</th>
<th>CCM</th>
<th>EDX</th>
<th>YN97</th>
<th>Ono</th>
<th>Nowi</th>
<th>Aarts</th>
<th>Appl</th>
<th>YN96</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7(*656)</td>
<td>664</td>
<td>19h27m</td>
<td>671</td>
<td>670</td>
<td>678</td>
<td>680</td>
<td>672</td>
<td>-</td>
<td>668</td>
<td>668</td>
</tr>
<tr>
<td>abz8(669)</td>
<td>669</td>
<td>43h52m</td>
<td>671</td>
<td>683</td>
<td>686</td>
<td>685</td>
<td>683</td>
<td>-</td>
<td>670</td>
<td>687</td>
</tr>
<tr>
<td>abz9(679)</td>
<td>683</td>
<td>37h8m</td>
<td>688</td>
<td>686</td>
<td>697</td>
<td>702</td>
<td>703</td>
<td>-</td>
<td>691</td>
<td>707</td>
</tr>
<tr>
<td>la21(*1046)</td>
<td>*1046</td>
<td>3h19m</td>
<td>1052</td>
<td>*1046</td>
<td>*1046</td>
<td>1050</td>
<td>1053</td>
<td>1047</td>
<td>1053</td>
<td>1053</td>
</tr>
<tr>
<td>la24(*935)</td>
<td>*935</td>
<td>1h31m</td>
<td>938</td>
<td>*935</td>
<td>*935</td>
<td>944</td>
<td>938</td>
<td>939</td>
<td>*935</td>
<td>*935</td>
</tr>
<tr>
<td>la38(*1196)</td>
<td>*977</td>
<td>4h46m</td>
<td>984</td>
<td>*977</td>
<td>*977</td>
<td>984</td>
<td>*977</td>
<td>*977</td>
<td>983</td>
<td>*977</td>
</tr>
<tr>
<td>la29(1153)</td>
<td>1157</td>
<td>36h10m</td>
<td>1167</td>
<td>1167</td>
<td>1166</td>
<td>1189</td>
<td>1184</td>
<td>1160</td>
<td>1185</td>
<td>1195</td>
</tr>
<tr>
<td>la38(*1196)</td>
<td><strong>1196</strong></td>
<td>47m</td>
<td>1208</td>
<td>*1196</td>
<td>*1196</td>
<td>1202</td>
<td>1201</td>
<td>*1196</td>
<td>1208</td>
<td>1209</td>
</tr>
<tr>
<td>la40(*1222)</td>
<td>1224</td>
<td>11h25m</td>
<td>1228</td>
<td>1224</td>
<td>1224</td>
<td>1235</td>
<td>1228</td>
<td>1229</td>
<td>1225</td>
<td><strong>1222</strong></td>
</tr>
</tbody>
</table>

in swv1(UB1418, from Storer, Wu, Vaccari), 1702 in swv6(UB1696), 971 in yn4(UB972, from Yamada). These results suggest the robustness of ISM for harder instances. We can expect stronger robustness by the force of non-GA methods, e.g. shuffle algorithm, or EDX+MSXF.

7 Conclusion and Future Work

Conventional GAs can be well applied to problems that are not globally multimodal. However, it is often observed that they fail to find the optimum and converge to a local optimum located far from it. In this paper, we defined this situation the "UV-phenomenon", proposed the UV-structure hypothesis that would cause the UV-phenomenon, and verified them by function optimization problems and JSP. Next, we proposed ISM that could avoid the UV-phenomenon, and applied ISM to JSP. Excellent performance of ISM on JSP suggests that ISM is superior to conventional GAs (e.g. to CCM, or obviously to simple GAs) in herder problems. Consequently, we are convinced that hypothesized UV-structures surely cause the UV-phenomenon.

We showed the performance of ISM on JSP, then we would like to apply ISM to other hard benchmarks, e.g. QAP (new instances are on http://www.fe.dis.titech.ac.jp/~psyche/qap/top.htm), or function optimization problems. As a future work, formulations of UV-structures and further analysis of the UV-phenomenon are desirable. They would show a new orientation of GAs.

References


Appendix

GTb method successively makes a schedule better. In each phase of remaking,
Let C be the set of operations that can be executed (C ≠ ∅). Let mc be the machine at which a operation c ∈ C is executed. Let g_m be the operation which is scheduled to be executed at first in machine m (not necessarily g_m ∈ C). Let St(c) be the fastest time to start the operation c ∈ C. Let Et(c) be the fastest time to finish the operation c ∈ C. Let t_m be the time machine m finish last operation.

In these notations,
Phase1. execute any(if exist) g_m ∈ C s.t. t_m=St(g_m) ←Phase 1.
Phase2. C_2:=\{c∈C|Et(c)≤St(g_{m_c})\}, if C_2 = ∅ →Phase 3. else execute any c_0 s.t. St(c_0)=\min_{c∈C_2}St(c) ←Phase 1.
Phase3. C_3:=\{g_m ∈ C \}, if C_3 = ∅ →Phase 4. else execute any c_0 s.t. Et(c_0)=\min_{c∈C_3}Et(c) ← Phase 1.
Phase4. execute any c_0 s.t. Et(c_0)=\min_{c∈C}Et(c), ←Phase 1.

Calculated amount of GTb is about 1.3 times bigger than GT method experimentally. And diversity of offsprings are tunable by loosening the attribute of selection of executing operation in phase 2,3,4.
Neighbourhood Based Robustness Applied to Tardiness and Total Flowtime Job Shops

Mikkel T. Jensen

Department of Computer Science, University of Aarhus, Denmark.
mjensen@daimi.au.dk

Abstract. The traditional focus of scheduling research is finding schedules with a low implementation cost. However, in many real world scheduling applications finding a robust schedule (a quality schedule expected to still be acceptable if something unforeseen happens) or a flexible schedule (a quality schedule expected to be easy to change) is just as important. In this paper the robustness and flexibility of schedules produced by minimizing neighbourhood based robustness measures are investigated. The basic idea is to minimize not only the implementation cost of a single schedule, but the implementation costs of a set of schedules located around a centre schedule. The problems used in the experiments are worst tardiness, summed tardiness and total flow time job shop problems. It is found that the robustness measures increase robustness and to some degree flexibility for worst tardiness and loose summed tardiness problems, while they do not perform well for tight summed tardiness problems and total flow time problems. It is conjectured that neighbourhood based robustness can be expected to work well on problems with few critical points and not well on problems with many critical points.

1 Introduction

When solving a scheduling problem the focus traditionally is on minimizing a measure of the cost of implementing the schedule. However, most real world scheduling systems operate in dynamic environments in which unforeseen and unplanned events can happen at short notice. Such events include the breakdown of machines, employees getting sick, new jobs appearing etc. The problem encountered when an unforeseen event such as a breakdown occurs and a schedule has to be changed is usually called a rescheduling problem. When a rescheduling problem is solved a new schedule incorporating the changes in the environment and the part of the preschedule (the schedule prior to the breakdown) already implemented is sought. This schedule should ideally have as low an implementation cost as possible. Often the scheduler will also attempt to make the new schedule as similar to the preschedule as possible, since deviations from the preschedule may be costly. When the unforeseen event is a breakdown (the temporary unavailability of a resource), the simplest way to solve a rescheduling problem is often to keep the preschedule, but delay processing when necessary. In the following this kind of rescheduling is called simple rescheduling or right-shifting. Right-shifting is the simplest and fastest kind of rescheduling, but in
order to improve performance often more complex methods searching some set of schedules is used. In the following, this is called rescheduling using search.

The difficulty of the rescheduling problem depends on the nature of the breakdown as well as the preschedule. Some preschedules will generally lead to rescheduling problems with lower implementation costs than others. A preschedule which tends to perform well after a breakdown and right-shifting is usually termed robust, while a schedule which tends to perform well after a breakdown and rescheduling using search is termed flexible.

It is the objective of this paper to investigate if the neighbourhood based robustness measure technique used in [12] on makespan problems can be used on other performance criteria for the job shop problem. The criteria investigated are maximal tardiness, summed tardiness and total flow time. This is done by following the approach used in [12], in which a robustness measure for the makespan job shop scheduling problem is defined. It is then shown experimentally that schedules found by optimizing the robustness measure on average perform better after a simulated breakdown and rescheduling than schedules found by optimizing the objective function.

The outline of the paper is as follows. In section 2 the job shop scheduling problem and notation are defined. In section 3 previous work on robust scheduling is briefly covered. Section 4 introduces the robustness measures for the worst, summed tardiness and total flow time job shop problems. In section 5 the genetic algorithm used to perform the scheduling is described, while section 6 describes how breakdowns are simulated and how rescheduling is performed in the experiments. Section 7 describes the experiments and reports the results. The paper is concluded in section 8 in which some conclusions are drawn.

2 Notation

A \( N \times M \) job shop scheduling problem consists of \( N \) jobs and \( M \) machines. A job \( j \) consists of a sequence of operations \( O_j = (o_{j1}, o_{j2}, ..., o_{jk_j}) \). Each operation \( o_{jl} \) is to be processed on a specific machine and has a specific processing time \( \tau_{jl} \). Each job has at most one operation on each machine. The processing order of the operations in job \( j \) must be the order specified in the sequence \( O_j \). During processing each machine can process at most one operation at a time, and no preemption can take place; once processing of an operation has been started it must run until it has completed. In the following \( C_j \) will denote the end of processing time of the last operation of job \( j \) in a given schedule.

Some problems include a due date \( d_j \) for each job, times by which the processing of the job is supposed to be finished, a release time \( r_j \) for each job, prior to which no processing of the job can be done, or a setup time \( s_m \) for each machine, prior to which no processing can be done on the machine.

A number of different objective functions exist for job shop problems. The most well researched is the makespan \( C_{max} = \max_{j \in \{1...N\}}(C_j) \), the time elapsed from the beginning of processing until the last operation has completed. The makespan objective is not realistic, since it is not well-suited for scheduling on
a rolling time horizon-basis (jobs arriving continuously over time), and since it does not include due dates. More realistic objectives include total flowtime
\[ F = \sum_{j=1}^{N} C_j - r_j, \]
summed tardiness
\[ T_\Sigma = \sum_{j=1}^{N} \max(C_j - d_j, 0) \]
and worst tardiness
\[ T_{\max} = \max_{j \in \{1..N\}} \left( \max(C_j - d_j, 0) \right), \]
all of which will be treated here.

All of the performance measures \( T_\Sigma, T_{\max} \) and \( F \) are regular measures, meaning that a schedule can never be depreciated by starting an operation earlier. This also means that in the set of optimal schedules there will always be an active schedule, a schedule in which no operation can be started earlier without delaying the start of another operation. Since an active schedule can be unambiguously described by the operation processing order, a natural representation of schedules for this kind of problem is the processing order.

3 Previous work

Key references on robust and flexible scheduling include the following:

Uncertain operation processing times have been treated by a number of authors. In [8] the worst case performance under a number of different scenarios of one- and two machine problems is considered. Based on a theoretical analysis of the problems a branch and bound algorithm and some scheduling heuristics are constructed. Using experiments these algorithms are shown to work well. In [14] a branch and bound algorithm is used to decompose a weighted tardiness job shop problem into a series of subproblems, each of which are solved during execution of the schedule. The method is demonstrated to produce more flexible schedules than two other scheduling methods. In [1] job shop scheduling based on pessimistic estimates of the processing times is shown to be superior to scheduling based on processing time averages.

In [7] an artificial immune system (AIS) for solving job shop problems is evolved. The schedules produced by the AIS are demonstrated to be more similar to each other than schedules produced by a standard GA approach. Based on this observation the AIS is conjectured to produce robust schedules.

Breakdown of machines has been treated in [10], in which a robustness measure based on slack for makespan job shop problems is defined, and experiments verify that schedules found optimizing the robustness measure perform better after a series of breakdowns than ordinary schedules.

In [12] a robustness measure is defined for the makespan job shop scheduling problem. By experiment it is shown that on average the schedules found by optimizing the robustness measure perform better after a breakdown and four different kinds of rescheduling rescheduling than ordinary schedules.

4 Robustness measures

The robustness measures are based on an idea introduced for continuous function optimization problems in [13, 4]: robust solutions are located on broad peaks in
Fig. 1. The idea in neighbourhood based robustness. If for some reason the solution will have to be changed, the broad peak solution may do much better than the narrow peak solution, since the solutions close to the broad peak solution are still reasonable solutions (compare $f(x')$ to $f(y')$). The searchspace of job shop scheduling problems is very different from this figure, but the basic idea is the same.

Fig. 2. Two $N_1$-neighbours. From A to B exactly one pair of consecutive operations on the same machine machine 3, jobs 2 and 3) have been exchanged.

The objective function landscape, while brittle solutions are located on narrow peaks (see Fig 1). In some cases a tradeoff must be made between the broadness and height of the solution peak. In scheduling the idea behind this is that when a breakdown occurs, maybe a schedule close to the preschedule can work (partly) around the breakdown. If this schedule has a low implementation cost in the original problem, there is a good chance that it will also have a low implementation cost in the rescheduling problem. For this reason, schedules produced using the robustness measures can be expected to be flexible. Since the schedules are created to be good despite small changes, they are also expected to be robust.

The makespan robustness measure introduced in [12] has the form

$$R_{C_{\text{max}}} (s) = \sum_{s' \in N_1(s)} \phi(s, s') C_{\text{max}} (s')$$

(1)

where $N_1$ is a neighbourhood defined on the schedule searchspace. The neighbourhood works on the processing sequence of the operations. The neighbourhood $N_1(s)$ of the schedule $s$ is the set of schedules that can be obtained by interchanging two consecutive operations on the same machine (see Fig 2). The function $\phi(s, s')$ is a weighting function, in [12] $\phi(s, s') = \frac{1}{|N_1(s)|}$ is used.

Following this approach we define the robustness measures used in this paper

$$R_P (s) = \sum_{s' \in N_1(s)} \phi(s, s') P(s'),$$

(2)

where $P$ is $T_\Sigma$, $T_{\text{max}}$ or $F$ and $\phi(s, s') = \frac{1}{|N_1(s)|}$. 
5 The scheduling system

The scheduling is done using a genetic algorithm. The GA was chosen since this kind of algorithm has previously been demonstrated to be well suited for job shop scheduling, and other more traditional scheduling techniques (shifting bottleneck, branch and bound techniques) are not well suited for the robustness measures. The permutation encoding described in [2, 11] is used. In this encoding a schedule is represented by a sequence of job numbers, describing the operation processing order. The simplest way to decode a gene is the semiactive decoding: The gene \((2,3,2,\ldots)\) will be decoded "first process the first operation of job 2, then the first operation of job 3, then the second operation of job 2, etc."

The objective functions used in the worst tardiness problems were \(T_{\text{max}}\) (normal runs) and \(R_{T_{\text{max}}}\) (robust runs). In these runs two different hillclimbers were used as decoders, improving the schedules produced by semiactive decoding. The decoder used in the runs labelled "normal" was based on the makespan hillclimber used in [11, 12], but improved worst tardiness instead. If a worst tardiness of 0 was achieved the hillclimber would maximize the least earliness of the schedule. The principle behind this hillclimber is to always try to make improvements along the critical path of the schedule, each step consisting of a permutation of operations at the beginning or end of a critical block (consecutive critical operations on the same machine). The decoder used in the runs labelled "robust" is a hillclimber climbing the same neighbourhood but optimizing a pessimistic estimate of \(R_{T_{\text{max}}}\) rather than \(T_{\text{max}}\). The generalisation of the makespan hillclimbers described in [11, 12] to worst tardiness is straightforward, and will not be treated further in his paper.

In the total flow time experiments the objective functions were \(F\) (normal) and \(R_F\) (robust). The decoder used in these experiments was the tunable hybrid decoder described in [3]. This decoder is based on the Giffler-Thompson algorithm for constructing active schedules, but can be biased to favour non-delay-like schedules by excluding schedules with too much delay using a parameter \(\delta\). The value \(\delta = 0.5\) was used in both normal and robust experiments, giving some favour to non-delay schedules.

In the summed tardiness experiments the objective functions were \(T_{\Sigma}\) (normal) and \(R_{T_{\Sigma}}\) (robust). In the summed tardiness experiments the worst tardiness hillclimbing decoders were used. These were found to be superior to the tunable hybrid decoder especially for large problems, despite the fact that they were made for optimizing another (related) performance measure.

After the completion of decoding the constructed schedule was written into the gene, in such a way that a semiactive decoding of the gene would yield the schedule produced by the hillclimber. The procedure is a kind of Lamarckian learning, and is often called forcing.

A basic genetic algorithm was used, since in preliminary experiments it was found to outperform a GA resembling the grid-structured GA used in [11]. Tournament selection with a tournament size of two was used, crossover-rate was 1.0, mutation rate 0.1. The generalized order crossover (GOX) and position based
mutation (PBM) operators were used, see [11]. Elitism was not used, but the all
time best individual was remembered and returned at the end of the run.

After the completion of the genetic algorithm a hillclimber searching the $N_1$-
neighbourhood was run on the best solution found. This hillclimber was run “on top of” the decoding hillclimbers used in the tardiness experiments. In the
case of normal runs using the $T_{max}$ decoding hillclimber the schedule was made
active after the run of the hillclimber using the procedure used for decoding in
[2]. This was done since it was found in preliminary experiments to improve the
rescheduling performance of the schedules.

6 Breakdowns and rescheduling methods

The breakdowns used in the experiments were simulated in the following way.
1. A random operation of the schedule is picked uniformly.
2. The starting-time of the chosen operation is denoted the breakdown time.
3. All operations with starting-times earlier than the breakdown time are re­
moved from the problem.
4. Release times of all jobs and setup times of all machines is set to the worst of
   the breakdown time and the end of processing time of any operation being
   processed at breakdown time of the respective machine/job.
5. The setup time of the machine with the breakdown is set to the breakdown
time plus the downtime. The downtime is the time during which the broken
down machine is unavailable. In all the experiments a downtime of 80 has
been used (for all the problems the processing time of each operation is
between 1 and 100).

The effect of this way of simulating a breakdown is that a random machine breaks
down precisely at the time it was supposed to start processing an operation. The
broken down machine is unavailable for some predefined (and known) time, an
obvious interpretation is that the machine is being repaired. The scheduler is
free to reschedule any operations not yet commenced at breakdown-time.

After a breakdown has been simulated rescheduling must be performed. In
this paper this is done in four different ways:
1. Right-shifting. Simply wait for the breakdown to be repaired and use the
   scheduling order of the preschedule. This is expected to yield low quality
   results compared to other methods, but at a very low computational cost.
2. $N_1$-based rescheduling. All $N_1$-neighbours of the preschedule are generated,
   and the one best solving the rescheduling problem is used. This is the sim­
plest kind of search based rescheduling used, and is expected to yield better
results than right-shifting, still at low computational cost.
3. Reduced rescheduling. Generate a reduced problem as described in [5] by
   removing all operations not directly affected by the breakdown from the
   problem. This problem is then solved from scratch using the GA.
4. Complete rescheduling. Solve the rescheduling problem from scratch using
   the GA. This is expected to give a high schedule quality at a high computa­
tional cost.
7 Experiments

The purpose of the experiments was to investigate if the schedules found optimizing the robustness measures perform better after a breakdown and after rescheduling than schedules found optimizing the ordinary performance measures, and if the preschedule performance (without breakdowns) was decreased when the robustness measures were optimized. This was done by for each performance criterion having a series of runs in which the performance criterion was optimized (these runs have been labelled “normal”), and a series of runs in which the corresponding robustness measure was optimized (these runs have been labelled “robust”). During each run a breakdown was simulated, and the rescheduling problem solved using each of the four rescheduling methods.

The tardiness benchmark problems used were generated from the makespan problems la01, la02, la06, la07, la26, la27, la31 and la36 from [9] and ft10 and ft20 from [6]. The due dates were generated by for each problem generating a random active schedule, and setting the due date for each job to the job’s completion time minus 5% (loose problems, labelled \( \sigma = 0.95 \)), 10% (medium problems, labelled \( \sigma = 0.90 \)) or 15% (tight problems, labelled \( \sigma = 0.85 \)). This procedure was followed in order to be able to investigate whether the tightness of the problems influenced the usefulness of the robustness measures. The benchmark problems are available upon request.

The worst tardiness experiments

The worst tardiness averages after a simulated breakdown can be seen in the top part of table 1. There is a column for each tightness value, and sub-columns labelled 1-4 for each rescheduling method. The bottom part of the table contains information about the number of superior benchmarks for each of the methods. Since the tardiness distribution is unknown no statistical testing has been performed; a scheduling method is simply said to be superior to the other method for a given benchmark if the average tardiness is observed to be lower. It can be seen that in all experiments involving the loose problems \( \sigma = 0.95 \) the robust searching scheme outperforms the normal searching scheme. The performance gain is generally largest when the simple rescheduling methods 1 and 2 are used, but it is also present for the complex rescheduling methods 3 and 4. There also seems to be a significant performance gain for the more tight problems \( \sigma = 0.90 \text{ and } 0.85 \), but not as large as for the loose problems. In all cases the difference is largest for the problems with a high job/machine ratio, the problems la06, la07, la26, la27 and ft20 (not visible in the table). A similar observation was made for makespan problems in [12].

The worst tardiness averages of the preschedules in the worst tardiness experiments can be seen in table 1 in the columns labelled P. For the loose problems the worst tardiness average of the preschedules was 0 for all problems but one, and the robust scheduling scheme performed just as well as the normal scheduling scheme. For the tighter problems the situation is seen to be different. For the medium problems \( \sigma = 0.90 \) in two problems normal scheduling outperformed
Table 1. The worst tardiness experiments. Top: Average worst tardiness over all the problems. The averages for each problem have been taken over 400 runs. P denotes the performance of the preschedules, while 1-4 denote the performance after rescheduling using the techniques described in section 6. The methods normal and robust refer to the method used when generating the preschedule and rescheduling using methods 3 and 4. Bottom: Number of problems for which the normal/robust scheduling methods on average have been observed to produce results superior to the other.

<table>
<thead>
<tr>
<th>Method</th>
<th>Averages</th>
<th>( \sigma = 0.95 )</th>
<th>( \sigma = 0.90 )</th>
<th>( \sigma = 0.85 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( P ) 1 2 3 4</td>
<td>( P ) 1 2 3 4</td>
<td>( P ) 1 2 3 4</td>
</tr>
<tr>
<td>Normal</td>
<td>4.3 46.5 40.0 23.1 22.4</td>
<td>13.4 45.3 59.0 43.2 42.2</td>
<td>52.8 109.3 103.4 89.0 87.6</td>
<td></td>
</tr>
<tr>
<td>Robust</td>
<td>4.3 20.0 18.5 16.1 15.5</td>
<td>14.4 43.5 41.0 36.1 35.4</td>
<td>54.6 96.5 92.5 85.5 84.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Superior problems</th>
<th>( \sigma = 0.95 )</th>
<th>( \sigma = 0.90 )</th>
<th>( \sigma = 0.85 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>( P ) 1 2 3 4</td>
<td>( P ) 1 2 3 4</td>
<td>( P ) 1 2 3 4</td>
</tr>
<tr>
<td>Normal</td>
<td>0 0 0 0 0 2 0 0 3 3 6 0 0 4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robust</td>
<td>0 10 10 10 10 0 10 10 7 7 0 10 10 6 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The conclusion of these experiments should be that generally it is possible to produce more robust schedules for worst tardiness problems by minimizing the robustness measure \( R_{T_{\max}} \). The same conclusion holds for the flexibility of the schedules, even though the performance increase is smaller. The performance increase after breakdowns is larger for loose problems than for tight problems, and for tight problems the robustness increase can come at the cost of a slightly increased tardiness in case no breakdown is encountered.

The summed tardiness experiments

The results of the summed tardiness experiments are summarised in table 2. The top part of the table indicates total averages over all the problems of summed tardiness for the preschedules and the four different rescheduling methods. The bottom part of the table indicates the number of problems for which the average found was superior for normal/robust scheduling.

The tables indicate that for the loose problems (\( \sigma = 0.95 \)) the robust searching scheme was superior to the normal searching scheme. For all problem/rescheduling combinations but two the robust searching scheme produces results superior to those of normal searching scheme after rescheduling. This is obtained at almost no cost in preschedule performance.

Comparing this to the performance of the medium and tight problems (\( \sigma = 0.90 \) and \( \sigma = 0.85 \)) it becomes clear that as the problems become more tight the performance of the robust searching scheme deteriorates, when comparing to the normal searching scheme. For the rescheduling methods 1 and 2 the robust performance is still better than the normal performance. For rescheduling methods 3 and 4 the robust method seems slightly superior for the medium problems, while there does not seem to be a significant difference for the tight problems.

The total flowtime experiments

The results of the total flow time experiments are summarised in table 3. Qualitatively the results are similar to the results for the tight summed tardiness
Table 2. The summed tardiness experiments. The top table is equivalent of the top part of table 1, while the bottom table equivalent of the bottom part of table 1.

<table>
<thead>
<tr>
<th>Averages</th>
<th>$\sigma = 0.95$</th>
<th>$\sigma = 0.90$</th>
<th>$\sigma = 0.85$</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
</tr>
<tr>
<td>normal</td>
<td>13.3 188.7 140.0 57.1 54.3</td>
<td>13.6 314.0 244.1 125.4 120.1</td>
<td>123.3 637.1 552.6 381.2 367.9</td>
</tr>
<tr>
<td>robust</td>
<td>15.7 65.7 57.0 42.3 40.8 58.0 182.5 154.4</td>
<td>116.6 113.5 252.9 547.7 489.4</td>
<td>380.0 389.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Superior problems</th>
<th>$\sigma = 0.95$</th>
<th>$\sigma = 0.90$</th>
<th>$\sigma = 0.85$</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
</tr>
<tr>
<td>normal</td>
<td>3 0 0 1 1</td>
<td>6 0 1 4 4</td>
<td>9 1 2 6 6</td>
</tr>
<tr>
<td>robust</td>
<td>0 10 10 9 9</td>
<td>4 10 9 6 6</td>
<td>6 1 9 8 4</td>
</tr>
</tbody>
</table>

Table 3. The total flow time experiments. The table on the left is equivalent of the top part of table 1, while the table on the right is equivalent of the bottom part of table 1.

<table>
<thead>
<tr>
<th>Averages</th>
<th>$\sigma = 0.95$</th>
<th>$\sigma = 0.90$</th>
<th>$\sigma = 0.85$</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
</tr>
<tr>
<td>normal</td>
<td>15206.0 15836.8 15734.0 15418.2 15572.0</td>
<td>16418.2 15372.0</td>
<td>8</td>
</tr>
<tr>
<td>robust</td>
<td>15213.4 15774.4 15687.0 15428.4 15386.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of superior problems</th>
<th>$\sigma = 0.95$</th>
<th>$\sigma = 0.90$</th>
<th>$\sigma = 0.85$</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
<td>$P$ 1 2 3 4</td>
</tr>
<tr>
<td>normal</td>
<td>7 3 3 8 8</td>
<td>7 3 3 8 8</td>
<td>7 3 3 8 8</td>
</tr>
<tr>
<td>robust</td>
<td>3 7 7 2 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

problems. The preschedule performance of the robust runs on average are worse than the preschedules produced by the normal runs. For rescheduling using methods 1 and 2 the robust runs seem slightly superior, while they seem inferior when methods 3 and 4 are used.

8 Discussion and conclusion

In the above it has been demonstrated by experiments that neighbourhood based robustness measures can improve the after rescheduling performance of worst tardiness problems and loose summed tardiness problems, while the idea does not work well for tight summed tardiness problems and total flow time problems.

The explanation of the poor performance of the robustness measures for total flow time and for the deteriorating performance of the summed tardiness robustness measure as the problems become more tight may be that the neighbourhood based robustness idea works best for problems in which there is a single or a few critical points. In makespan problems or worst tardiness problems this will generally be the case. There will usually be one job which is the “worst”. The other jobs can be sacrificed in order to alleviate problems (breakdowns) worsening this one job. For loose summed tardiness problems there will often only be one or a few critical jobs as well (if there are only few tardy jobs). As the problems become more tight the situation becomes more diffuse. There is no longer one critical job, but a number of them making the rescheduling problem more complex. The higher complexity of the rescheduling problem may result in a low implementation cost of the preschedule becoming more important for rescheduling performance than anything else, and decreasing the probability of finding a good solution to the rescheduling problem close to the preschedule.

The single critical job property of makespan, worst tardiness and loose summed tardiness problems probably also helps the robust scheduler find low cost
preschedules; in many cases the exchange of two operations will not worsen the critical jobs, and will not change the performance measure of the schedule. One may speculate that the way the neighbourhood based robustness scheme works actually is by forcing the preschedule to have a large number of operation exchanges which do not deteriorate the performance of the schedule. It is probably impossible to achieve this for a problem with many critical jobs.

These observations are valuable if neighbourhood based robustness is to be applied to real world scheduling problems or other combinatorial optimization problems, since it gives the person working with the problem a hint as to whether neighbourhood based robustness measures will work or not.

References

Solving Extended Hybrid-Flow-Shop Problems Using Active Schedule Generation and Genetic Algorithms*

Martin Kreutz1**, Detlef Hanke2, and Stefan Gehlen1

1 ZN GmbH, Universitätsstraße 160, 44801 Bochum, Germany
2 LVE Verfahrenselektronik GmbH, Moltkestraße 29, 45128 Essen, Germany

Abstract. We propose a hybrid approach for solving hybrid-flow-shop problems based on the combination of genetic algorithms and a modified Giffler & Thompson (G&T) algorithm. Several extensions of the hybrid-flow-shop are considered and discussed in the context of a real-world example. The genome in the GA encodes a choice of rules to be used to generate production schedules via the G&T algorithm. All constraints to the scheduling task are observed by the G&T algorithm. Therefore, it provides a well suited representation for the GA and leads to a decoupling of domain specific details and genetic optimization. The proposed method is applied to the optimization of a batch annealing plant.

1 Introduction

Scheduling tasks arise in many fields of industrial production and are of large practical relevance. Several traditional techniques, mostly originating from the field of operations research, have been proposed to solve these problems. However, the complexity of these problems (most of them are NP-hard) renders many of the traditional methods impractical when larger problem instances are considered. To our knowledge Davis [3] was the first to propose a GA-based technique to address scheduling problems. Since then, GAs have been applied to this problem domain with increasing frequency, using many different representations and operators. A binary representation to encode schedules [10] allows the use of standard genetic operators but needs specialized repair operators to retain the ability to decode chromosomes as feasible schedules. In the case of a direct encoding of the schedule [2], no decoding is necessary but the genetic operators need to use much domain and constraint knowledge to appropriately mutate schedules, and to maintain feasible schedules. Most of the recent methods, therefore, employ an indirect encoding in which a schedule building algorithm is combined with genetic search through the space of potential inputs to the schedule builder. This includes permutation-based representations as well as heuristic combination methods (HMC) [6]. Our approach provides an interface to both permutations and heuristic rules. In particular, we focus on dynamic hybrid-flow-shop problems, as these have most relevance in practice. Real-world environments are subject to many sources of change which are typically treated...

* Supported by the BMBF under Grant No. 01IB802A9 (LEONET).
** email: kreutz@zn-gmbh.com
as random occurrences, such as new job releases, cancellation of jobs, machine breakdowns, etc. In dynamic scheduling problems, minimizing the total processing time (makespan) is not an appropriate criterion since the scheduling horizon is open and periodical rescheduling has to be done. Therefore, other optimization criteria has to be considered [4].

In Sec. 2 the particular problem class, which is studied, is introduced and related to traditional hybrid-flow-shop problems. Several extensions are discussed. In Sec. 3 a modification of the G&T algorithm is presented which handles the introduced extensions in a systematic manner. This algorithm is used in combination with a GA, described in Sec. 4, to generate production schedules by means of genetically optimized priority rules. The complete optimization scheme is applied to a real-world problem which is outlined in Sec. 5. The experimental results are discussed in Sec. 6 and in Sec. 7 a conclusion with an outlook is given.

2 Production Scheduling and Extended Hybrid-Flow-Shop Problems

Scheduling may be defined as the allocation of available resources to a set of operations or tasks over a planning horizon, the objective being to best satisfy one or more performance criteria, e.g. minimum makespan or idle time. A production scheduling problem, in general, involves $M$ machines or resources $\{m_1, ..., m_M\}$ and a collection of $J$ jobs $\{j_1, ..., j_J\}$, each of which may comprise a number of separate operations $\{o_{j1}, ..., o_{jO_j}\}$. In the classical sense, an operation is described by a tuple $(m, t)$, in which $m$ is the resource that is required by the operation and $t$ is the time it will take to process this operation on resource $m$. In an extended form, an operation may require more than one resource, see explanation below. Furthermore, $m$ may be chosen from a set of equivalent resources. In this case not a single resource but a resource type is required by the operation. This relates to the field of parallel-machines problems. Moreover, the operations in each job may be ordered by certain successor-predecessor relations. A feasible schedule assigns one or more resources and the corresponding start times to each operation, satisfying the constraint that a resource can only process one operation at a time, and the ordering constraints on operations within a job. In real-world problems additional issues have to be considered as an importance (or priority), a due-date, and a ready-time (or arrival time) of a job.

In the field of operations research, there exists a well developed formalism and a differentiated hierarchy of problem classes. The class of flow-shop problems is described by the condition that the operations of each job require each resource exactly one time in a fixed, predefined order. Flow-shops with more than one identical resources have been termed hybrid-flow-shops [7]. The latter class serves as a basis for further extensions: The order of operations may be only partially defined. Operations may require more than one resource at a time. Some resources are required more than once. In certain cases the same resources are required by two or more consecutive operations. Not all processing times are known in advance.
3 An Adapted Algorithm for Active Schedule Generation

For the optimization of the makespan, that is the time interval between the first job entering the system and last job leaving the system, it is sufficient to consider only active schedules.

**Definition 1.** An active schedule contains no excess idle times and has no operations which can be completed earlier without delaying other operations.

A systematic approach to generate active schedules was proposed by Giffler & Thompson [5]. The generation process is controlled by a set of dispatch or priority rules which resolves conflict situations. Although the idle time is used as an optimization criterion instead of the makespan the focus on active schedules represents an intuitive and sensible reduction of the search space. However, the original G&T algorithm is only defined for flow-shop problems. In order to employ it for hybrid-flow-shop problems some modifications have to done, see algorithm 1.

**Algorithm 1.** Extended version of the algorithm proposed by Giffler & Thompson for hybrid-flow-shop problems:

1. Determine set $N$ of all next operations, that is the set of all operations which can be scheduled next.
2. Determine the next resource $m$ for which the minimum of all potential completion times of operations $o \in N$ is attained. This minimum is called minimum completion time $t_c$.
3. Determine the conflict set $C$ for $m$ which is the subset of all jobs where the corresponding next operation requires $m$ and can be started before $t_c$.
4. Choose a job from $C$ according to a priority rule and assign $m$ to its next operation with the earliest possible start time.
5. As long as not all operations are scheduled goto step (1).

Algorithm 1 represents a true generalization of the original G&T algorithm. For the special case of a flow-shop without parallel resources both algorithms are exactly the same. For the hybrid-flow-shop it cannot be expected that the algorithm can generate all possible active schedules. For identical parallel resources, however, the following theorem can be proven.

**Theorem 1.** For identical parallel resources algorithm 1 is able to generate at least one instance of each equivalence class of active schedules. Two schedules are equivalent iff they only differ in a permutation of identical parallel resources.

**Proof.** See appendix A

With regard to an optimization method this reduction of combinatorial complexity is desirable since all employed optimization criteria are invariant under permutation of identical resources. Furthermore, the following theorem states, that only feasible and active schedules are generated.

**Theorem 2.** Every schedule generated by algorithm 1 is feasible and active.

**Proof.** See appendix B
The allocation rule in step (2) and the selection rule in step (3) ensure that only active schedules are generated. An intuitive generalization to problems where operations require more than one resource consists in iterating algorithm 1 for each operation until all required resources are assigned to this operation. An operation belongs as long to the set of next operations (see step (1) of algorithm 1) as not all resources are assigned. The following difficulties arise in the generalization of steps (2) and (3):

- In step (2) of algorithm 1 all potential minimum completion times of operations $o \in \mathcal{N}$ must be known in order to determine $t_c$. The completion time of an operation, however, is only known if all resources except the last one are assigned to this operation.
- The same problem arises in step (4). In order to determine the potential start times the completion times of the predecessor operations must be known.
- If more than one resource is required, an order or priority of the assignment has to be defined.

A possible solution is to replace the fixed allocation rule in step (2) and use priority rules for the choice of the respective resource instead. This leads to the following modification of algorithm 1

**Algorithm 2. Modification of algorithm 1.**

1. same as in algorithm 1
2. Determine all resources which are not assigned to a partially scheduled operation. Choose one of these resources according to a priority rule. Proceed as in step (2) of algorithm 1.
3. same as in algorithm 1
4. Choose a job from $C$ according to a priority rule and assign $m$ to its next operation. If all required resources are assigned to this operation the earliest possible start time is set to the maximum of the minimum start times on all involved resources and the completion time of the predecessor operation. If all resources are assigned the operation is completely scheduled.
5. same as in algorithm 1

Algorithm 2 is able to solve very general types of hybrid-flow-shop problems. The processing time of an operation need not to be known in advance. It is sufficient to be able to determine it at the moment of the assignment of a resource. Moreover, an operation may require an arbitrary number of resources and some resources may be required in two or more consecutive operations. This flexibility, however, is bought at the expense of an increase of the combinatorial complexity since in addition to the priority rules for jobs a priority rule for resources must be chosen. Furthermore, it is neither guaranteed that algorithm 2 only generates active schedules, nor that all active schedules can be generated. A general observation is that it is more likely to generate active schedules if the set of all potential resources in step (2) and the conflict set in step (3), respectively, are as large as possible. This can be achieved by a simple heuristic priority rule for the resources which sorts the resources in ascending order according to their first use in the sequence of operations, that is during the course of the processing of a job. Other heuristics may be considered depending on domain specific knowledge.
4 Optimization of Priority Rules by Means of Genetic Algorithms

4.1 Representation

One crucial issue for the successful application of genetic algorithms is the choice of the right representation for the problem in combination with the right genetic operators acting on it. In our approach we employ an indirect encoding of production schedules by means of priority rules and use the proposed algorithm 2 as a mapping from the representation space to the solution space. A solution encoded by priority rules is always feasible and observes all given constraints which is ensured by building the schedule with algorithm 2. This approach decouples the problem specific part of the scheduling task and the optimization part for which a GA is employed with rather general operators and without special repair mechanisms.

Outcomes of applying all possible sets of priority rules are represented as permutations of the involved jobs. In the case, where each operation requires each resource only once, it is sufficient to encode one priority rule for each resource type which leads to a $M \times J$ matrix, where $M$ denotes the number of resource types and $J$ the number of jobs, respectively. If more than one resource is required at a time or the same resource is required by consecutive operations, the chromosome would be even larger. A trade-off between highest flexibility and computational efficiency is to focus on the most important resource type and use heuristics for the other resources types. That is, only a permutation is used for the representation of the problem. In this form, the approach resembles other permutation based approaches with the difference that a more sophisticated schedule building method is employed which is theoretically motivated and allows for further extensions. However, basically all genetic operators which are proposed for permutation based encoding can be used.

4.2 Fitness Function

In the case of schedule optimization several objectives are conceivable. The \textit{makespan} measures the total flow time, the \textit{tardiness} penalizes jobs with completion times larger than their due dates, and the \textit{idle time} sums all periods during a resource is idle. Especially, for dynamic scheduling problems, minimizing the \textit{idle time} seems to be a well suited criterion, since it reduces the gaps that lie in between a schedule. The \textit{makespan} tends to leave large gaps in a schedule in favor of an earlier completion of the last operation, see figure 1. These gaps cannot filled during a subsequent rescheduling and therefore lead to a less efficient schedule in this case.

![Fig. 1. Gantt charts of two schedules, left: minimal makespan, right: minimal idle time.](image-url)
4.3 Mutation

New priority rules are generated by means of a set of simple mutation operators.

Shuffle: A permutation is randomly initialized. This operator is only defined for reasons of comparison with other operators.

Swap: The elements at two randomly selected positions are exchanged. With respect to the representation this is the smallest possible change.

Swap of best-matching elements: Same as Swap but favors elements which have similar flow times and require the same resource type.

Shift: Performs a circular right shift of the permutation. If \( J \) is the number of jobs, only \( J \) different mutated solutions are possible.

Shift within an interval: Same as Shift but on a randomly selected subset of all positions.

4.4 Crossover

It should be clear that random crossover is hardly meaningful for any combinatorial optimization problem, therefore is little helpful for producing good parental solutions. Thus, crossover must be specialized to, and meaningful for the problem. Since all domain specific knowledge should be encapsulated in algorithm 2, which translates the representation of a solution to a concrete schedule, only rather general crossover operators are considered [1]. An interesting operator was proposed by Lin et al. [8] which uses the G&T algorithm as a repair mechanism for recombined solutions. However, none of the examined operators show a significant increase of performance. In some cases, the performance was inferior. Therefore, the use of crossover was omitted in all experiments. The rôle of crossover in this approach is subject to further investigation.

4.5 Local Search

Small changes of a permutation may imply large changes of the quality of the solution which violates the principle of strong causality [11] and leads to an instability of the genetic search. Augmenting genetic algorithms with local search heuristics is a promising approach to increase the stability and has been incorporated in mutation operators [9], as well as in crossover operators [12]. In this GA local search is performed by stochastic hill-climbing using swap mutations.

5 A Real-World Application

In this section the optimization of a part of a steel works – a batch annealing plant – is considered. The task of this plant is the annealing of cold-rolled steel strips in order to improve their material quality. The following subtasks have to be performed: The steel strips are rolled up to coils which are assembled to stacks with regard to their similarity in geometry, material properties, heating and cooling times, etc. This leads to a bin packing or grouping problem for which an branch & bound algorithm is employed. On the basis of the geometry, material properties, and other constraints for each stack an optimal heating and cooling profile is calculated using a thermodynamic model. After all stacks have
been build and their respective heating and cooling times have been calculated. The task consists in scheduling the annealing jobs on the plant with regard to the following (partially disagreeing) criteria: optimal utilization of available resources, minimal costs of production, best compliance with the given constraints, optimal average performance for the dynamic flow-shop. The resource types of the particular annealing plant are:

- (a) annealing bases,
- (b) bell-type furnaces (heating hoods),
- (c) cooling hoods,
- (d) a crane for transportation of coils, furnaces, and hoods.

An annealing job is divided into the following operations:

- (1) transport of coils of a stack from the coil depot to an annealing base,
- (2) transport of a heating hood to an annealing base,
- (3) heating of a stack,
- (4) removal of a heating hood and transport of a cooling hood to a base,
- (5) cooling of a (previously heated) stack,
- (6) removal of a cooling hood and transport of the coils to an output depot.

A further subdivision of each operation is omitted for the sake of clarity.

The problem of building a production scheduling for the described tasks can be viewed as an instance of the extended hybrid-flow-shop problem, see Sec. 2:

- for each operation more than one parallel resource for (a) – (c) is available,
- all operations require more than one resource simultaneously, e.g. operation (2) requires an annealing base (a), a heating hood (b), and the crane (d),
- some resources are required for consecutive operations, e.g. the annealing base (a) is needed during operations (1) – (6),
- not all processing times are known in advance, e.g. the times for all transport operations depend on the current position of the respective resources.

6 Experimental Results and Discussion

The considered annealing plant is equipped with 24 annealing bases, 12 heating hoods, 12 cooling hoods, and one crane. Five different data sets containing 122 stacks of coils are available. The optimization problem is defined as a dynamic scheduling task. Starting from an "empty" plant the jobs of the first data set are scheduled. In order to simulate a realistic scenario of a continuous production it is assumed that after a certain number of time steps a rescheduling has to be performed where all operations which has not been started at this moment and all jobs of the next data set are taken into account. It should be noted, that consecutive operations which require the same resource type cannot be rescheduled with a different resource, e.g. once a heating operation has been started the successive cooling operation cannot be performed on a different annealing base. This implies constraints for the choice of the concrete resource of a required resource type during the next scheduling. These constraints can be handled by step (2) of algorithm 2. The rescheduling is repeated for each data set. All separate
schedules are optimized according to the minimum idle time, whereas the overall quality is evaluated by the total makespan. The population of the GA comprises 12 individuals including one elitist.

In order to perform small, controlled steps in the representation space the influence the used mutation operators plays a crucial role. We measure the effect of the proposed mutation operators in terms of the average deviation between the original and the mutated solution. The results are summarized in table 1.

<table>
<thead>
<tr>
<th>operator</th>
<th>deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>shuffle</td>
<td>100%</td>
</tr>
<tr>
<td>swap</td>
<td>12.3%</td>
</tr>
<tr>
<td>shift</td>
<td>16.4%</td>
</tr>
<tr>
<td>shift within an interval</td>
<td>10.8%</td>
</tr>
<tr>
<td>swap of best match</td>
<td>11.2%</td>
</tr>
</tbody>
</table>

The results show that all operators lead to a moderate change of the quality of the solution compared to the shuffle operation which corresponds to a Monte-Carlo search. Thus, when an individual undergoes mutation a significant amount of information is preserved which can be accumulated in the population and guide the evolutionary process. In order to rate the performance of a particular operator with respect to a given task and a given criterion the comparison has to be done in terms of real optimization examples. Table 2 show the performance of each operator relative to the shuffle operator for the given real-world example.

<table>
<thead>
<tr>
<th>operator</th>
<th>perform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>shuffle</td>
<td>0%</td>
</tr>
<tr>
<td>swap</td>
<td>30.5%</td>
</tr>
<tr>
<td>shift</td>
<td>-5.9%</td>
</tr>
<tr>
<td>shift within an interval</td>
<td>19.9%</td>
</tr>
<tr>
<td>swap of best match</td>
<td>29.1%</td>
</tr>
</tbody>
</table>

The best performance is achieved with the simple swap and swap of best match, whereas the shift operators yield inferior results. Although, swap of best match tries to incorporate some domain knowledge, the results differ not substantially from those of the simple swap.

7 Conclusion and Further Work

We proposed a hybrid approach for the solution of extended hybrid-flow-shop problems based on the combination of a GA and a modified G&T algorithm. This approach provides a very general framework for the combination of heuristic methods and GAs. It is possible to systematically include transportation devices into the planning and allow for operations that require more than one resource at once. Moreover, the search space is reduced by the G&T algorithm to a subset of potentially optimal solutions. The use of priority rules provides a well suited
interface for optimization algorithms such as GAs and allows the incorporation of domain specific knowledge described by rules. In this way the modified G&T algorithm serves as a representation used in the GA and decouples the domain specific part from the part of optimization. The two criteria makespan and idle time show large differences when applied to a static or a dynamic scheduling task, respectively. In the case of a single set of jobs to be scheduled the makespan is an appropriate criterion whereas for a continuous rescheduling the idle time seems to be better suited.

There are several issues for future research. It is known that the G&T algorithm only produces the subset of active schedules which includes all solutions optimal with respect to the minimal makespan. However, it has to be investigated which subset of all schedules contains the optimal solutions when idle time is considered. With regard to the GA the rôle of crossover has to be studied. And finally, although the first results are very encouraging, further investigation is needed and the method has to be applied to a broader range of applications.

References

A Proof of Theorem 1

Let \( \mathcal{P} \) denote an active production schedule, in which \( l \) operations are scheduled. It is assumed that algorithm 1 has generated a partial schedule \( \mathcal{Q} \) with \( k \) operations consistent with \( \mathcal{P} \). An operation in \( \mathcal{Q} \) is called consistent with \( \mathcal{P} \) if it has the same start and completion time in \( \mathcal{P} \) and \( \mathcal{Q} \), but is not necessarily assigned to the same resource (however, to an instance of the same class). Let \( O_k \) denote the set of operations which are already scheduled in \( \mathcal{Q} \). It is sufficient to show that for \( 0 \leq k < l \) one additional operation \( o_{k+1} \) can be scheduled in \( \mathcal{Q} \) which is consistent to \( \mathcal{P} \). Then, the proof of the statement follows by induction.

Let \( m \) be the next resource that is assigned to an operation and \( t_c \) the minimum completion time according to algorithm 1. Let \( \mathcal{M} \) denote the set of resources which are equivalent to \( m \) and which have the same completion time \( t_c \) as resource \( m \). It is to show, that a resource \( m^* \in \mathcal{M} \) exists, which is assigned in \( \mathcal{P} \) to an operation from the conflict set \( \mathcal{C} \). Then, the algorithm may, consistently with \( \mathcal{P} \), assign \( m \) to operation \( o_{k+1} \). Since \( \mathcal{P} \) is an active schedule, operation \( o_k \) has the same start and completion time (the earliest possible) in \( \mathcal{P} \).

Under the assumption, that there is no resource in \( \mathcal{M} \), which is assigned to an operation in \( \mathcal{P} \) and an arbitrary resource \( m^* \) is chosen from \( \mathcal{M} \), three cases arise:

(i): All operations in \( \mathcal{P} \) to which \( m^* \) is assigned are element of \( O_k \). Let \( o^* \) be the operation which determines the minimum completion time \( t_c \) in step 2 of algorithm 1. Then, a resource is assigned to \( o^* \) which results in a later completion time for \( o^* \) as it would be the case with resource \( m^* \). Thus, the assignment of \( m^* \) to \( o^* \) corresponds to a left shift which contradicts the assumption that \( \mathcal{P} \) is an active schedule.

(ii): In \( \mathcal{P} \) only operations \( o \) with \( o \notin \mathcal{N} \) are assigned to \( m^* \). In this case, before the start of \( o \) another operation \( o' \) must be completed which is not element of \( O_k \). This operation is not completed in \( \mathcal{P} \) before \( t_c \). The start time of \( o \), therefore, is behind \( t_c \). Thus, \( m^* \) is not assigned to any operation in the period, in which it could be assigned to \( o^* \) by algorithm 1. This would result in a left shift and contradicts the assumption that \( \mathcal{P} \) is active.

(iii): In \( \mathcal{P} \) only operations \( o \) with \( o \in \mathcal{N} \land o \notin \mathcal{C} \) are assigned to \( m^* \). According to the construction of the conflict set \( \mathcal{C} \) all start times of the remaining operations not element of \( \mathcal{C} \) are behind \( t_c \). Thus, as in case ii, \( m^* \) is not assigned to any operation in the period, in which it could be assigned to \( o^* \) by algorithm 1 which is an contradiction to the assumption of \( \mathcal{P} \) being active.

B Proof of Theorem 2

It is sufficient to show that in no step of algorithm 1 an operation is added, for which a left shift is possible. The assumption, that for operation \( o \) a left shift exists from resource \( m \) to another resource \( m' \), leads to three cases:

(i): Let \( m = m' \). After a left shift \( o \) would be the last scheduled operation for resource \( m \). This contradicts the allocation rule in algorithm 1, according to which the operation is to be scheduled with a start as early as possible.

(ii): Let \( m \neq m' \). After a left shift \( o \) would be the last scheduled operation for \( m' \). This contradicts the selection rule for the next resource to be assigned in algorithm 1.

(iii): After a left shift the start time of \( o \) would be before start of operation \( o' \) which was scheduled in a former step of algorithm 1. Let \( t' \) denote the minimum completion time of this former step. According to the definition of the conflict set \( o' \) starts before \( t' \). On the other hand, the predecessor of \( o \), operation \( o_{-1} \), is completed after \( t' \) which implies that in a feasible production schedule \( o \) must start after \( t' \). Therefore, the left shift is not allowed.
A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem

Manuel Vázquez and Darrell Whitley

Computer Science Department, Colorado State University, Fort Collins, CO 80523
vazquez,whitley@cs.colostate.edu

Abstract

A variety of Genetic Algorithms (GA’s) for the static Job Shop Scheduling Problem have been developed using various methods: direct vs. indirect representations, pure vs. hybrid GA’s and serial vs. parallel GA’s. We implement a hybrid GA, called OBGT, for solving JSSP. A chromosome representation containing the schedule itself is used and order-based operators are combined with techniques that produce active and non-delay schedules. Additionally, local search is applied to improve each individual created. OBGT results are compared in terms of the quality of solutions against the state-of-the-art Nowicki and Smutnicki Tabu Search algorithm as well as other GAs, including THX, HGA and GA3. The test problems include different problem classes from the OR-library benchmark problems and more structured job-correlated and machine-correlated problems. We find that each technique, including OBGT, is well suited for particular classes of benchmark problems, but no algorithm is best across all problem classes.

1 Introduction

The goal in the static Job Shop Scheduling Problem (JSSP) is to find the sequence of $j$ jobs to be completed on $m$ machines such that the makespan (finish time of the last operation) is minimized. Davis [1985] was the first to use Genetic Algorithms to solve Job Shop Problems. Other heuristics such as Simulated Annealing (SA) and Tabu Search (TS), and exact techniques as branch and bound have also been used for solving JSSP. Jain and Meeran [1998, 1999] provide a good description of these techniques, including generic as well as problem specific strategies. The objective of this research is to evaluate the effectiveness of order-based operators combined with the Giffler and Thompson [1960] algorithm as a repair method. Results are compared against local search techniques.
that incorporate domain-specific information about critical paths and critical blocks for solving JSSP.

The rest of this paper is organized as follows. In Section 2 a detailed description of the JSSP is given. Section 3 summarizes previous research related to our work. In Section 4 OBGT is sketched. Section 5 summarizes our results.

2 JSSP description

The JSSP is defined using the following notation:

\[ S = \{R, P, O, C\} \] is a 4-tuple representing the scheduling problem.

\[ R = \{R_1, R_2, \ldots, R_r\} \] represents the \( r \) resources or machines.

\[ P = \{P_1, P_2, \ldots, P_p\} \] represents the \( p \) products to be finished in the job shop.

Each product \( P_i = \{Op_{i1}, Op_{i2}, \ldots, Op_{iy}\} \) consists of a set of operations and an operational precedence relation (representing technological constraints).

\[ O = \{O_1, O_2, \ldots, O_o\} \] represents the orders flowing in the shop to finish the products. The orders in the job shop are represented for a tuple, \( O_j = \{P_j, D_j\} \), where \( P_j \) is the product, and \( D_j \) is the due date of the order.

\[ C = \{C_1, C_2, \ldots, C_c\} \] represents the constraints of the problem.

The objective is to produce an ordered sequence of operations to process on each machine. The processing start times of each operation are assigned so as to optimize the objective function and to satisfy the problem constraints.

3 Previous research

3.1 Chromosome representations for JSSP

There are two ways of representing a schedule: indirect and direct. In indirect representations, the chromosome contains an encoded schedule. A schedule builder is used to transform the chromosome into a feasible schedule. Indirect representations range from traditional binary representations [Nakano, et al., 1991] to domain-specific knowledge representations [Bagchi, et al., 1991].

In direct representations, the chromosome directly represents the production schedule. Bruns [1993] and Yamada and Nakano [1992] describe many ways to deal with direct representations. Direct representation performs efficiently on production scheduling, incorporates domain-specific operations easily, but also requires domain-specific recombination operators.

3.2 Types of feasible schedules in JSSP

There are four classes of feasible schedules in JSSP: inadmissible, semi-active, active and non-delay. Inadmissible schedules contain excess idle time. It is possible to improve the schedule quality by forward-shifting of operations until no excess idle time exists. Semi-active schedules contain no excess idle time, but they can be improved by shifting some operations to the front without delaying others. Active schedules contain no idle time and none of the operations can
be finished earlier without delaying other operations. The optimal schedule is guaranteed to be an active schedule. Non-delay schedules are active schedules, in which operations are placed into the schedule such that the machine idle time is minimized. No machine is kept idle if some operation can be processed.

Two methods are applied to generate feasible schedules. The Giffler and Thompson method (GT) which produces active schedules and the Non Delay algorithm (ND) which produces non-delay schedules. Both methods were originally described by Giffler et al. [1960]. In both methods, a heuristic is applied to select the next operation to be scheduled.

The Giffler and Thompson (GT) algorithm has been used in many JSSP implementations. Lin, Goodman and Punch [1997b] described the representations and the crossover operators used in previous GT-algorithm-based genetic algorithms for static JSSP. Usually, the GT algorithm is used to convert the offsprings into active schedules to guarantee feasibility.

Lin, Goodman and Punch [1997b] also created two operators inspired by the GT algorithm: THX crossover and THX mutation. They claim that these operators transmit temporal relationships present in the schedule. THX crossover selects a crossover point and uses it as a decision point in the GT algorithm to exchange information between two schedules. THX mutation builds the critical block for the schedule and two operations in the block are randomly selected and reversed.

Hart and Ross [1998] used a representation called “Heuristic Combination Method” (HCM). HCM uses an implicit representation in which each gene in the chromosome contains a heuristic that performs the decision choice at each step during the schedule generation process. Their chromosome stores pairs (method, heuristic) where method is a choice between GT and ND and heuristic is the strategy followed to select the operation to be scheduled. This method, called the Heuristically Guided GA (HGA), always produces active schedules. HGA was designed to deal with the dynamic JSSP, but is easily adapted to the static case. We previously replicated HGA results for dynamic JSSP benchmark problems [Vázquez, et al., 2000].

### 3.3 Hybrid methods and local search

JSSP is a hard problem that cannot be solved efficiently by applying any single technique and a great deal of research has focused on hybrid methods. Jain and Meeran [1998a] reviewed in detail Tabu Search, Genetic Algorithms and Simulated Annealing techniques and they proposed ways to produce hybrid solutions.

Jain, Rangaswamy and Meeran [1998b] describe and compare in detail JSSP neighborhood models and move evaluation strategies. The goal is to define a constrained neighborhood which can be evaluated quickly without losing the ability to produce improved solutions.

Bagchi and coauthors [1991] discuss the use of a GA to rapidly reduce the search space to a size that can subsequently be handled by a deterministic search algorithm. Yamada and Nakano [1992], in their solution applicable to
large-scale Job Shop problems, proposed for future research that a combination of their technique with local search heuristics would produce improvements in their algorithm’s performance.

3.4 Order-based operators for scheduling

We introduce a hybrid order-based GA, where order-based operators are combined with GT and ND methods. Our hypothesis is that the previous outstanding GAs for JSSP has been overly complicated. Order-based operators preserve the relative order of the operations to be scheduled, which is an important characteristic in scheduling problems. In the JSSP domain, order-based operators will not always produce feasible schedules. But the GT algorithm can be used to repair illegal offsprings.

Davis developed the first order-based operators that are well suited for many scheduling problems [Davis, 1985], [Davis, 1991].

a) Uniform Order-based Crossover: A number of elements are selected from one parent and copied to the offspring. The missing elements are taken in the order in which they appear in the other parent.

b) Order-based Mutation: A sub-list of elements is selected from the parent and this sub-list is randomly permuted.

Following Davis, Syswerda developed two sets of order-based operators that are well suited for many scheduling problems [Syswerda, 1991]. This includes a form of order-based crossover and mutation operators, as well as a position-based crossover and mutation operators. Syswerda’s position-based crossover is the same as Davis’ Uniform Order-based Crossover. Whitley and Yoo [1995] have shown that Syswerda’s order-based and position-based operators are in fact identical in expected behavior. In the experimentation section, only results using Davis’ operator are presented.

4 OBGT, An Algorithm for JSSP

We use permutations as a direct representation of the schedule. This allows for efficiently generating JSSP schedules and makes it easy to incorporate domain-specific operators. Figure 1 contains a description of our algorithm. Order-based operators are used to recombine the permutations and the Giffler and Thompson (GT) algorithm is used to repair illegal offsprings. This is similar to the strategy followed by Bierwirth, et al. [1999]. Fang [1994] indicates that optimal schedules are active schedules. As noted, the Giffler and Thompson (GT) algorithm generates active schedules and the Non Delay (ND) algorithm produces ND schedules. According to Fang, optimal schedules are not necessarily members of the subset of ND schedules for some JSSP instances. In our solution, the ND algorithm is used to generate half of the individuals of the initial population. The other half are GT active schedules.

We also incorporate domain-specific techniques (based on critical paths) into the GA to improve the performance of the solution. We implement the
Procedure *OBGT Algorithm*

S1 Generate Initial Population (50% ND, 50% GT)
S2 While not converged
S2.1 For all individuals from \( i = 1 \) to Popsize
S2.1.1 Cross string \( i \) with probability 'pc' (select 2nd parent randomly)
S2.1.2 Apply Mutation with probability 'pm' to both offsprings
S2.1.3 Apply GT algorithm to repair both offsprings
S2.1.4 Hillclimb both offsprings
S2.1.5 Replace worst individual with best offspring (if improvement exists)
S2.1.6 Sort population (bubble offspring into place)

Figure 1: OBGT Algorithm

Grabowski method described by Jain, et al. [1998b]. This neighborhood uses block structures, where a move is defined by repositioning an operation to either the front or the rear of a critical block.

Our Order-Based Giffler and Thompson (OBGT) for the static JSSP has the following characteristics:

- A direct chromosome representation, containing the schedule itself, is used. This includes a permutation of jobs for each machine. Each operation (i.e., job on a machine) contains its start time.
- Order-based operators were implemented.
- The GT and ND algorithms were utilized to generate active and non-delay schedules.
- Grabowski critical path method was applied to each generated offspring.
- A selection and replacement strategy similar to GENITOR [Whitley, et al., 1988] is used.

Notice from Figure 1 that our implementation is not exactly the GENITOR algorithm. Another description of our algorithm is given in [Vázquez, et al., 2000]. The population size was 100 and the probabilities of crossover and mutation were 0.7 and 0.01 respectively. (Higher crossover rate work nearly as well, and it is unclear if using a crossover probability less than 1.0 is really necessary).

5 RESULTS

OBGT is compared against THX, a Tabu Search algorithm, Mattfeld's GA3 and HGA. We implemented the following algorithms: A serial version of Lin's
[1999b] THX, Nowicki and Smutnicki [1996] Tabu Search (NS), a serial version (using an integer representation instead of binary) of the Hart and Ross [1998] HGA. In the case of Mattfeld’s GA3 [1996], we compared against his published results. NS Tabu Search and GA3 represent the respective state of the art for Tabu Search and Genetic Algorithms for solving JSSP.

In order to make fair comparisons all the algorithms are run 30 times with a population size of 100 and a maximum number of 100 generations. In the case of NS Tabu Search, the maximum number of iterations is set to 100,000.

5.1 Experiments

The classical Fisher and Thompson (FT06 ‘6x6’, FT10 ‘10x10’, FT20 ‘20x5’) Yamada and Nakano (YN1, YN2, YN3, YN4), Lawrence (LA21 ‘10x15’, LA27 ‘20x10’, LA29 ‘20x10’, LA38 ‘15x15’) and Taillard (TA051, TA052, TA053, TA054, TA055, TA056, TA057, TA058, TA059, TA060) benchmarks available in the OR-library were among the JSSP instances used. The solutions obtained by the different algorithms were compared against the best-known solutions.

All of the OR-library problems rely heavily on random generators. Assuming that ‘real-world’ JSSP instances contain a certain amount of structure and additional constraints on the problem definition, a problem generator created by Watson and co-workers is used to generate eight machine-correlated problems (MC01a, MC02a, MC03a, MC04a, MC01b, MC02b, MC03b, MC04b) and eight job-correlated problems (JC01a, JC02a, JC03a, JC04a, JC01b, JC02b, JC03b, JC04b) 2. In job-correlated problems, jobs have correlated processing times across machines. In machine correlated problems, job processing times are similar on a particular machine. A description of the problem generator can be found in [Watson, et al., 1999].

<table>
<thead>
<tr>
<th>Prob</th>
<th>BKS</th>
<th>NS</th>
<th>THX</th>
<th>HGA</th>
<th>GA3</th>
<th>OBGT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Mean</td>
<td>sd</td>
<td>Best</td>
<td>Mean</td>
<td>sd</td>
</tr>
<tr>
<td>FT06</td>
<td>55</td>
<td>55.0</td>
<td>0.0</td>
<td>55</td>
<td>55.0</td>
<td>0.0</td>
</tr>
<tr>
<td>FT10</td>
<td>930</td>
<td>930</td>
<td>943.7</td>
<td>11.3</td>
<td>930</td>
<td>1003.6</td>
</tr>
<tr>
<td>FT20</td>
<td>1165</td>
<td>1165</td>
<td>1170.9</td>
<td>6.9</td>
<td>1165</td>
<td>1237.3</td>
</tr>
</tbody>
</table>

Table 1: Fisher and Thompson Benchmark Results

Tables 1-6 present the results for the benchmark problems. The Best-Known Solutions are denoted BKS. In the case of machine-correlated and job-correlated problems, the results are compared against the lower bounds (LB).

1The benchmark problems and best-known solutions can be found at http://mscmga.ms.ic.ac.uk/job/orlib/jobshopinfo.html
2This problem generator and the description of job-correlated and machine-correlated problems is available at http://www.cs.colostate.edu/sched/generator.html
In the Fisher and Thompson benchmark (Table 1), all the algorithms except HGA find the best-known solution. HGA is not able to find the best known solution for all problems, but in average is better than all of the algorithms except GA3. GA3 presents the best average solution and HGA behaves similar to GA3 on average.

In the Yamada and Nakano benchmark (Table 2), OBGT produces consistently the best solution, but in the average case GAS is better. On this benchmark THX and HGA did not perform well and NS produces competitive solutions.

In the Lawrence series (Table 3), GAS produced the best results in 3 out of 4 problems. NS is slightly superior to OBGT. HGA and THX did not produce good results and THX is not well suited for this benchmark.

In the Taillard benchmark (Table 4), NS outperforms the rest of the algorithms. HGA and GAS produced relatively good results. The OBGT and THX results are relatively poor.

For job-correlated problems (Table 5), the best algorithm is OBGT, but NS and HGA produced also good results. THX behaves well in 10x10 instances and poorly in 20x20 ones. NS was beaten in all instances except JC04a, JC03b and JC04b. JC04a is the easiest instance (since all the algorithms find the optimal solution).

For machine-correlated problems (Table 6), OBGT and NS performs the best (NS is slightly better). THX did not work well, although THX find the best solution for the MC03a case (but in the average case produces poor solutions) and HGA produced reasonable results.

---

### Table 2: Yamada and Nakano '20x20' Benchmark Results

<table>
<thead>
<tr>
<th>Prob.</th>
<th>BKS</th>
<th>NS</th>
<th>THX</th>
<th>HGA</th>
<th>GA3</th>
<th>OBG T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
</tr>
<tr>
<td>YN01</td>
<td>888</td>
<td>918.4</td>
<td>997.7</td>
<td>948.1</td>
<td>911.9</td>
<td>952.9</td>
</tr>
<tr>
<td>YN02</td>
<td>909</td>
<td>934.3</td>
<td>987.8</td>
<td>963.2</td>
<td>940.5</td>
<td>947.5</td>
</tr>
<tr>
<td>YN03</td>
<td>903</td>
<td>924.7</td>
<td>980.3</td>
<td>942.1</td>
<td>918.8</td>
<td>951.7</td>
</tr>
<tr>
<td>YN04</td>
<td>968</td>
<td>1000.4</td>
<td>1049.3</td>
<td>1023.3</td>
<td>1012.0</td>
<td>999.6</td>
</tr>
</tbody>
</table>

### Table 3: Lawrence Benchmark Results

<table>
<thead>
<tr>
<th>Prob.</th>
<th>NS</th>
<th>THX</th>
<th>HGA</th>
<th>GA3</th>
<th>OBG T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
<td>Best Mean sd</td>
</tr>
<tr>
<td>LA21</td>
<td>1046</td>
<td>1059.5</td>
<td>1067.2</td>
<td>1075</td>
<td>1099</td>
</tr>
<tr>
<td>LA27</td>
<td>1235</td>
<td>1282.5</td>
<td>1349</td>
<td>1376.8</td>
<td>1308</td>
</tr>
<tr>
<td>LA29</td>
<td>1231</td>
<td>1333</td>
<td>1355.2</td>
<td>1417</td>
<td>1296</td>
</tr>
<tr>
<td>LA38</td>
<td>1196</td>
<td>1211.3</td>
<td>1319</td>
<td>1337.3</td>
<td>1250</td>
</tr>
</tbody>
</table>

---

### 5.2 Interpretation

In the Fisher and Thompson benchmark (Table 1), all the algorithms except HGA find the best-known solution. HGA is not able to find the best known solution for all problems, but in average is better than all of the algorithms except GA3. GA3 presents the best average solution and HGA behaves similar to GA3 on average.

In the Yamada and Nakano benchmark (Table 2), OBGT produces consistently the best solution, but in the average case GA3 is better. On this benchmark THX and HGA did not perform well and NS produces competitive solutions.

In the Lawrence series (Table 3), GA3 produced the best results in 3 out of 4 problems. NS is slightly superior to OBGT. HGA and THX did not produce good results and THX is not well suited for this benchmark.

In the Taillard benchmark (Table 4), NS outperforms the rest of the algorithms. HGA and GA3 produced relatively good results. The OBGT and THX results are relatively poor.

For job-correlated problems (Table 5), the best algorithm is OBGT, but NS and HGA produced also good results. THX behaves well in 10x10 instances and poorly in 20x20 ones. NS was beaten in all instances except JC04a, JC03b and JC04b. JC04a is the easiest instance (since all the algorithms find the optimal solution).

For machine-correlated problems (Table 6), OBGT and NS performs the best (NS is slightly better). THX did not work well, although THX find the best solution for the MC03a case (but in the average case produces poor solutions) and HGA produced reasonable results.
Table 4: Taillard '30x20' Benchmark Results

<table>
<thead>
<tr>
<th>Prob. BKS</th>
<th>NS</th>
<th>THX</th>
<th>HGA</th>
<th>GA3</th>
<th>OBGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA512760</td>
<td>2765 2775.3</td>
<td>15.3</td>
<td>3119.6</td>
<td>36.3</td>
<td>2908.8</td>
</tr>
<tr>
<td>TA522756</td>
<td>2756 2767.3</td>
<td>22.6</td>
<td>3126.9</td>
<td>34.3</td>
<td>2932.9</td>
</tr>
<tr>
<td>TA532718</td>
<td>2718 2743.6</td>
<td>17.4</td>
<td>2933.6</td>
<td>34.2</td>
<td>2822.3</td>
</tr>
<tr>
<td>TA542849</td>
<td>2849 2862.1</td>
<td>16.5</td>
<td>3054.2</td>
<td>41.2</td>
<td>2876.4</td>
</tr>
<tr>
<td>TA552762</td>
<td>2762 2785.2</td>
<td>25.4</td>
<td>3041.5</td>
<td>36.9</td>
<td>2944.7</td>
</tr>
<tr>
<td>TA562781</td>
<td>2781 2806.0</td>
<td>23.4</td>
<td>3111.9</td>
<td>31.2</td>
<td>2959.4</td>
</tr>
<tr>
<td>TA572943</td>
<td>2943 2950.0</td>
<td>13.4</td>
<td>3272.4</td>
<td>40.5</td>
<td>2990.8</td>
</tr>
<tr>
<td>TA582899</td>
<td>2899 2918.2</td>
<td>18.1</td>
<td>3252.0</td>
<td>33.6</td>
<td>2919.9</td>
</tr>
<tr>
<td>TA593255</td>
<td>3255 2680.3</td>
<td>19.6</td>
<td>3049.1</td>
<td>36.3</td>
<td>2870.1</td>
</tr>
<tr>
<td>TA602723</td>
<td>2723 2742.1</td>
<td>17.4</td>
<td>2995.2</td>
<td>36.9</td>
<td>2830.8</td>
</tr>
</tbody>
</table>

Table 5: Job Correlated Benchmark Results. JC01a, JC02a, JC03a and JC04a are '10x10' jobs. JC01b, JC02b, JC03b and JC04b are '20x20' jobs.

6 Conclusions

Order-Based operators combined with the Giffler and Thompson repair method (OBGT) is a viable alternative for solving JSSP. Results indicate that OBGT produced good results (except on the Taillard benchmark), but these results have high variance among makespan values. Future research needs to examine larger problems in order to determine the OBGT scalability.

Another important observation is that HGA is a good near-optimal strategy and it does not reflect scalability problems. HGA consumes less computational resources than OBGT and the variance among HGA results is smaller than the variance for the rest of the GAs (only GA3 produces less variance in some cases and HGA is competitive in terms of variance to NS). HGA maybe a good candidate for real-time scheduling and in dynamic scheduling applications.

During the evaluation for job-correlated and machine-correlated problems, OBGT produced the best results. Further research is necessary on problems that are more representative of real-world problems. NS-Tabu Search did not produce the best results in job-correlated problems, but its performance was good on machine-correlated problems. GA3 [Mattfeld, 1996] is highly competitive for solving JSSP. Future work should test GA3 on job-correlated and machine-correlated problems.

According to Lin and colleagues [Lin, et al., 1997a], [Lin, et al., 1997b],
Table 6: Machine Correlated Benchmark Results. MCO1a, MC02a, MC03a and MC04a are ‘10x10’ jobs. MC01b, MC02b, MC03b and MC04b are ‘20x20’ jobs.

THX produces good results in its parallel implementation (all of the results shown in this research considers a serial implementation). Hart and Ross HGA implementation uses a Parallel Island Model [Hart, et al., 1998]. GA3 is also a parallel GA and its design includes several complicated factors (population flow in the diffusion model, inheritance of attributes, population entropy, etc.).

Currently HGA is a simple sequential algorithm. We have also not been able to compare run times with algorithms such as GA3. Run times were kept minimal in the current experiments. A parallel island model of OBGT should reduce variance in the solution quality and perhaps allow OBGT to compete even better against algorithms such as GA3 on a broader range of problems.

7 Acknowledgments

Manuel Vázquez is a graduate student at Colorado State University supported by PDVSA under the program “Becas 1998-2000”. Thanks to Jean-Paul Watson and Laura Barbulescu for the problem generator and code for Nowicki and Smutnicki’s Tabu Search implementation. This effort was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References


T. Yamada and R. Nakano [1992]. “A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems”. In *Parallel Problem Solving from Nature 2 (PPSN 2).*
Representations and Operators
An Empirical Study on GAs “Without Parameters”

Th. Bäck¹,² and A.E. Eiben¹,³ and N.A.L. van der Vaart¹

¹ Leiden University, ² ICD Dortmund, ³ Free University Amsterdam

Abstract. In this paper we implement GAs that have one or more parameters that are adjusted during the run. In particular we use an existing self-adaptive mutation rate mechanism, propose a new mechanism for self-adaptive crossover rates, and redesign an existing variable population size model. We compare the simple GA with GAs featuring only one of the parameter adjusting mechanisms and with a GA that applies all three mechanisms - and is therefore almost “parameterless”. The experimental results on a carefully designed test suite indicate the superiority of the parameterless GA and give a hint on the power of adapting the population size.

Introduction

Traditional genetic algorithms have parameters that must be specified before the GA is run on a problem. The most important parameters are the strategy or control parameters: population size (N), mutation rate (p_m), and crossover rate (p_c). The optimum setting of these parameters can be different for each problem. Moreover, the optimal values can be different in different phases of one single run on a given problem. Setting parameter values correctly is, therefore, a hard task. In general, there are two major forms of setting parameter values: parameter tuning and parameter control [EHM99]. Parameter tuning is the usual approach of finding good parameter values before the run of the GA and then these static values are used during the whole run. Parameter control is different because it changes initial parameter values during the run. Parameter control itself has three variants: deterministic, adaptive and self-adaptive. Deterministic means that parameters are changed according to a rule which uses no feedback from the GA, usually it is some sort of time-varying scheme. For adaptive control, feedback does take place and the values are changed in direction or magnitude depending on this feedback. Self-adaptivity is the idea of evolution of the parameters of evolution; the values are encoded into the chromosomes and are also subject to mutation, crossover and selection. The better parameter values will survive because they produce better offspring. Throughout this paper we maintain this terminology and use the term self-adjusting if we do not want or cannot specify which form of parameter control is used.

The main objective of this paper is to investigate the feasibility of eliminating the three main parameters, N, p_m, and p_c. That is, we are looking for experimental evidence on the performance of a parameterless GA. The race is not won in advance by any of the GA variants, for we are dealing with a clear trade-off situation. On the one hand, on-line parameter adjustment causes a learning overhead, that is the GA is
solving a problem and is learning good parameter values at the same time. This might cause a performance decrease w.r.t. using steady parameters. On the other hand, adjustable parameters provide the GA with valuable flexibility. If the GA can adjust its parameters (sub)optimally it might cause increased efficiency. A priori it cannot be predicted which effect will influence the GA performance more.

Previous research on the mutation rate ($p_m$), crossover rate ($p_c$) and population size ($N$), has focused on either adapting/self-adapting one of these parameters or on adjusting two or all of these parameters in a non self-adaptive fashion (i.e. adaptive or dynamic parameters). As for crossover there has not been previous research on self-adapting the crossover rate. The technical research objectives here are threefold.

1. Designing and investigating a new method for self-adapting the crossover rate.
2. Investigating the effect of self-adjusting the parameters $p_m$, $p_c$ and $N$ separately; a self-adaptive $p_m$, a self-adaptive $p_c$ and an adaptive $N$. (Notice that a self-adaptive population size wouldn't make much sense.)
3. Investigating the combined use of self-adjusting $p_m$, $p_c$ and $N$ within a completely self-adjusting GA.

Test suite

To gain relevant experimental feedback we have carefully chosen a number of test functions. For selecting the test function we followed the guidelines after [WMRD95, BäMi97] and required that the test suite should

1) include a few unimodal functions for comparison of efficiency (convergence velocity),
2) contain nonlinear, nonseparable problems,
3) contain scalable functions,
4) have a standard form,
5) include a few multimodal functions of different complexity with a large number of local optima,
6) not have only multimodal functions that have a regular arrangement of local optima, (because this might favour GA-operators that exploit the regularity),
7) contain high-dimensional functions, because these are better representatives of real-world applications.

The following test suite of five functions conforms to the rules listed above: $f_1$ is the sphere model after De Jong [DeJo75], $f_2$ is the generalized Rosenbrock function [Rose60,HoBa91], $f_3$ is the generalized Ackley function [Ackl87,BäSc93], $f_4$ is the generalized Rastrigin function [TöZi89,HoBa91], and $f_5$ is the fully deceptive six-bit function [Deb97] (this is the only function in the test set that uses 6 bits/variable). In order to comply to a standard form; they all have dimension $n=10$ and use 20 bits/variable (except $f_5$).
\[ f_1(x) = \sum_{i=1}^{n} x_i^2 \]

\[ f_2(x) = \sum_{i=1}^{n-1} \left( 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2 \right) \]

\[ f_3(x) = -20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e \]

\[ f_4(x) = 10n + \sum_{i=1}^{n} \left( x_i^2 - 10\cos(2\pi x_i) \right) \]

\[ f_5(x) = n - \sum_{i=1}^{n} \begin{cases} 0.92 \frac{4}{1.00} (x_i - 4) & \text{if } x_i \geq 4 \\ 4 (x_i - 4) & \text{if } x_i < 4 \end{cases} \]

\[ , x_i \text{ is # of 1's } \in \text{ gene } i \]

**Algorithms**

The basic traditional GA used for this research is a steady-state GA using Gray coding, a mutation rate of \( p_m = 1/l \), and a crossover rate of \( p_c = 0.90 \). The population size is \( N = 60 \). The chromosome length will be 220 bits (80 bits for \( f_3 \)), which consists of 10x20=200 bits (10x6=60 bits for \( f_5 \)) for the ten function variables (dimension \( n=10 \)) and at the end 2x10=20 bits for the two self-adaptive parameters \( p_m \) and \( p_c \). Although the \( p_m \) and \( p_c \) are not used by the TGA, they are added and calculated for two reasons. The first reason is that, compared to the other GA’s, any (dis)advantage from a different length of the bitstring is ruled out. Second reason is that now, at the end of a TGA run, one can check if the values of \( p_m \) and \( p_c \) are totally random, as they should be. We use uniform crossover and parents are selected through tournament selection with tournament size 2. The replacement strategy is delete worst; the two worst members of a population are deleted to make place for two new members. The initialization of the population for each function comes from random bits stored in 30 files. These files will be used for each different GA. This guarantees that every GA and every function starts with the same population and no (dis)advantage is caused by a different initialization. The GA terminates when the (known) optimum is found or the maximum number of fitness evaluations is reached. The maximum number of evaluations is 100,000 for \( f_1 \), \( f_3 \) and \( f_5 \) and 500,000 for \( f_2 \) and \( f_4 \).
Self-adaptive mutation
This mechanism is implemented after Bäck [Bäck92a, Bäck92b] and Fogarty and Smith [FoSm96]: a self-adaptive mutation rate between 0.001 (< 1/l) and 0.25 is encoded in extra bits at the tail of every individual. For each member in the starting population the rate will be completely random within that range. Mutation then takes place in two steps. First only the bits that encode the mutation rate are mutated and immediately decoded to establish the new mutation rate. This new mutation rate is applied to the main bits (those encoding a solution) of the individual. The GA using this mechanism is called self-adaptive mutation GA (SAMGA).

Self-adaptive crossover
To our knowledge a stand-alone self-adaptive $p_c$ on a GA has not been used before. Previous research was done on adaptive $p_c$ e.g. [Davi89] or on self-adapting $p_c$ simultaneously with self-adapting the $p_m$ in a trade-off [Juls95]. In our method a self-adaptive crossover rate between 0 and 1.0 is coded in extra bits at the tail of every individual (initialized randomly). When a member of the population is selected for reproduction by the tournament selection, a random number $r$ below 1 is compared with the member's $p_c$. If $r$ is lower than $p_c$, the member is ready to mate. If both selected would-be parents are ready to mate two children are created by uniform crossover, mutated and inserted into the population. If it is not lower, the member will only be subject to mutation to create one child which undergoes mutation and survivor selection immediately. If both parents reject mating, the two children are created by mutation only. If one parent is willing to mate and the other one does not, then the parent that is not in for mating is mutated to create one offspring, which is inserted in the population immediately. (Note that in such a case we deviate from the (N+2) scheme and use (N+1) instead.) The willing parent is on hold and the next parent selection round only picks one other parent.

Adaptive population size
Following the ideas of Arabas, Michalewicz and Mulawka with GAVaPS [Mich94, p.70], every new individual is allocated a (remaining) lifetime or life span according to the individuals fitness by a bi-linear scheme. Each cycle, the remaining lifetime (RLT) of all the members in the population is decremented by one. However, we make an exception for the fittest member, whose RLT is left unchanged. If the RLT of an individual reaches zero it is removed from the population. The bi-linear formula in GAVaPS is for functions to be maximized. Therefore we adjust this formula for the functions in this investigation. In this formula MinLT and MaxLT stand for the allowable minimum and maximum lifetime of an individual. The other variables are linked with the current state of the search. These variables are fitness(i), AvgFit, BestFit and WorstFit. They stand for the fitness of individual i, average fitness, best fitness and worst fitness of the current living population. The individual for which the remaining lifetime (RLT) is being calculated, is considered to be already inserted into the population in this scheme and therefore (may) influence AvgFit, WorstFit and BestFit.
The values for MinLT and MaxLT are set to 1 and 11 in this study, because initial runs with different values indicated that MaxLT=11 delivers good performance. (One could say that choosing a population size N is now shifted to choosing a maximum lifetime MaxLT.) The function RLT(i) gives a (remaining) lifetime of 1 to 11 to each new individual i that enters the population. If its fitness(i) is better than the average fitness, it gets a life span of 7 to 11; if it is worse it gets a life span of 1 to 6. In the higher subrange the much-better-than-average individuals get higher values than the somewhat-better-than-average individuals. The same procedure applies for the lower subrange: the worst individuals get much lower lifetimes than the almost-average individuals. The initial population consists of 60 individuals.

Note that our GA is different from GAVaPS in that it is a steady-state GA instead of a generational GA. The evolution process is long, so it is likely that eventually every individual will die of old age.

**Performance measures**

With every one of the five different GA’s, 30 runs will be done per test function, which will make a total of 750 runs. In all GAs the best fitness and the average fitness of the population is monitored until a run terminates. For the self-adjusting GA’s the relevant parameters (p_m, p_c and population size) will also be monitored during each run. The speed of optimization will be measured by the Average number of Evaluations on Success (AES), how many evaluations did it take - on average - for the successful runs to find the optimum. The Success Rate (SR) shows how many of the runs were successful in finding the optimum. If the GA is somewhat unsuccessful (SR < 30), the measurement MBF (Mean Best Fitness) shows how close the GA can come to the optimum. If the SR=30, then the MBF will be 0, because every run found the optimum. The MBF includes the data of all 30 runs in it, the successful and the unsuccessful ones.

**Experimental results**

In this section we present the main results of five algorithm variants. A full overview of experiments can be found in [vdVaart99]. The simple TGA will serve as a benchmark for the other variants. Three GAs use one extension to the TGA: the GA with self-adaptive mutation rate only (SAMGA), the GA with self-adaptive crossover...
rate only (SAXGA), and the GA with adaptive population size only (APGA). Finally, we study the GA featuring all extensions, called SAMXPGA.

<table>
<thead>
<tr>
<th></th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>AES st. dev.</td>
<td>14,669±1,934</td>
<td>-</td>
<td>16,042±6,502</td>
<td>163,498±61,614</td>
<td>-</td>
</tr>
<tr>
<td>MBF St. dev.</td>
<td>0</td>
<td>0.281±0.317</td>
<td>0</td>
<td>0</td>
<td>0.64±0.103</td>
</tr>
<tr>
<td>Avg $p_m$ st.dev.</td>
<td>0.121±0.076</td>
<td>0.126±0.061</td>
<td>0.131±0.065</td>
<td>0.123±0.054</td>
<td>0.13±0.048</td>
</tr>
<tr>
<td>Avg $p_c$ st.dev.</td>
<td>0.454±0.223</td>
<td>0.473±0.25</td>
<td>0.559±0.259</td>
<td>0.545±0.222</td>
<td>0.504±0.18</td>
</tr>
</tbody>
</table>

Table 1. SR, AES, MBF, avg. $p_m$ and avg. $p_c$ for the TGA at the end of all runs. The self-adaptive $p_m$ and $p_c$ are not used by the TGA, but are added to the table to see if they are truly random.

Note that the interval for $p_m$ is [0.001..0.25] and for $p_c$ [0..1.0]. If they are really randomly divided, the average $p_m$ and $p_c$ should be respectively around 0.1245 and 0.5. Table 1 indicates that they are indeed random and therefore have no influence on, or are influenced by, the TGA.

<table>
<thead>
<tr>
<th></th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>AES st. dev.</td>
<td>18,345±2,847</td>
<td>-</td>
<td>17,042±2,392</td>
<td>378,178±84,301</td>
<td>-</td>
</tr>
<tr>
<td>MBF st. dev.</td>
<td>0</td>
<td>0.376±0.463</td>
<td>0</td>
<td>1.758±2.869</td>
<td>0.688±0.1</td>
</tr>
<tr>
<td>Avg $p_m$ st.dev.</td>
<td>0.0021±0.0017</td>
<td>0.0029±0.0021</td>
<td>0.0024±0.0015</td>
<td>0.002±0.0014</td>
<td>0.0023±0.0016</td>
</tr>
<tr>
<td>Avg $p_c$ st.dev.</td>
<td>0.471±0.247</td>
<td>0.47±0.241</td>
<td>0.467±0.285</td>
<td>0.458±0.259</td>
<td>0.492±0.254</td>
</tr>
</tbody>
</table>

Table 2. SR, AES, MBF, avg. $p_m$ and avg. $p_c$ for the SAMGA at the end of all runs. The self-adaptive $p_c$ is not used by the SAMGA, but is added to the table to see if it is truly random.
### Genetic Algorithm with Self-Adaptive Crossover (SAXGA)

<table>
<thead>
<tr>
<th></th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>AES st.dev.</td>
<td>16,033 ±1,878</td>
<td>-</td>
<td>21,486 ±14,050</td>
<td>170,820 ±75,891</td>
<td>-</td>
</tr>
<tr>
<td>MBF st.dev.</td>
<td>0</td>
<td>0.355 ±0.342</td>
<td>0</td>
<td>0</td>
<td>0.581 ±0.09</td>
</tr>
<tr>
<td>Avg $p_m$ st.dev.</td>
<td>0.1191 ±0.0592</td>
<td>0.12 ±0.0628</td>
<td>0.1358 ±0.0646</td>
<td>0.1319 ±0.067</td>
<td>0.1404 ±0.0459</td>
</tr>
<tr>
<td>Avg $p_c$ st.dev.</td>
<td>0.68 ±0.188</td>
<td>0.57 ±0.266</td>
<td>0.565 ±0.25</td>
<td>0.617 ±0.246</td>
<td>0.514 ±0.161</td>
</tr>
</tbody>
</table>

Table 3. SR, AES, MBF, avg. $p_m$ and avg. $p_c$ for the SAXGA at the end of all runs. The self-adaptive $p_m$ is not used by the SAXGA, but is added to the table to see if it is truly random.

### Genetic Algorithm with Adaptive Population Size (APGA)

<table>
<thead>
<tr>
<th></th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>AES st.dev.</td>
<td>9,645 ±1,240</td>
<td>-</td>
<td>19,919 ±14,903</td>
<td>163,494 ±58,433</td>
<td>-</td>
</tr>
<tr>
<td>MBF st.dev.</td>
<td>0</td>
<td>0.133 ±0.186</td>
<td>0</td>
<td>0</td>
<td>0.587 ±0.136</td>
</tr>
<tr>
<td>Avg $p_m$ st.dev.</td>
<td>0.1409 ±0.0703</td>
<td>0.1291 ±0.0777</td>
<td>0.0964 ±0.072</td>
<td>0.1412 ±0.0704</td>
<td>0.1089 ±0.0606</td>
</tr>
<tr>
<td>Avg $p_c$ st.dev.</td>
<td>0.558 ±0.258</td>
<td>0.586 ±0.257</td>
<td>0.432 ±0.278</td>
<td>0.496 ±0.252</td>
<td>0.485 ±0.293</td>
</tr>
<tr>
<td>Avg remaining lifetime st.dev.</td>
<td>6.36 ±0.73</td>
<td>5.86 ±0.75</td>
<td>6.23 ±0.57</td>
<td>6.41 ±0.7</td>
<td>5.04 ±1.55</td>
</tr>
<tr>
<td>Avg population size st.dev.</td>
<td>14.1 ±2.66</td>
<td>13.5 ±2.6</td>
<td>12.7 ±2.02</td>
<td>12.8 ±2.45</td>
<td>7.8 ±2.84</td>
</tr>
</tbody>
</table>

Table 4. SR, AES, MBF, avg. $p_m$, avg. $p_c$, avg. RLT and avg. population size for the SAPGA at the end of all runs. The self-adaptive $p_m$ and $p_c$ are not used by the SAPGA, but are added to the table to see if they are truly random.
Table 5. SR, AES, MBF, avg. pm, avg. pc, avg. RLT and avg. population size for the SAMXPGA at the end of all runs

<table>
<thead>
<tr>
<th></th>
<th>f₁</th>
<th>f₂</th>
<th>f₃</th>
<th>f₄</th>
<th>f₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>AES</td>
<td>16,505 ±4,906</td>
<td>-</td>
<td>16,821 ±8,869</td>
<td>143,933 ±94,496</td>
<td>-</td>
</tr>
<tr>
<td>MBF</td>
<td>0</td>
<td>0.199 ±0.244</td>
<td>0</td>
<td>0</td>
<td>0.568 ±0.11</td>
</tr>
<tr>
<td>Avg pm</td>
<td>0.0028 ±0.0023</td>
<td>0.0205 ±0.0261</td>
<td>0.0038 ±0.0057</td>
<td>0.0032 ±0.0024</td>
<td>0.024 ±0.0403</td>
</tr>
<tr>
<td>Avg pc</td>
<td>0.495 ±0.297</td>
<td>0.622 ±0.27</td>
<td>0.563 ±0.213</td>
<td>0.629 ±0.286</td>
<td>0.597 ±0.291</td>
</tr>
<tr>
<td>Avg remaining lifetime</td>
<td>6.13 ±0.92</td>
<td>5.75 ±1.47</td>
<td>6.32 ±0.96</td>
<td>6.17 ±0.92</td>
<td>4.8 ±1.83</td>
</tr>
<tr>
<td>Avg population size</td>
<td>13.4 ±5.36</td>
<td>10.8 ±4.33</td>
<td>11.4 ±3.83</td>
<td>11.6 ±5.09</td>
<td>9.2 ±5.7</td>
</tr>
</tbody>
</table>

Table 6. Global comparison of GA versions on the test suite

<table>
<thead>
<tr>
<th></th>
<th>TGA</th>
<th>SAMGA</th>
<th>SAXGA</th>
<th>APGA</th>
<th>SAMXPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>f₂</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>f₃</td>
<td>1</td>
<td>2½</td>
<td>5</td>
<td>4</td>
<td>2½</td>
</tr>
<tr>
<td>f₄</td>
<td>2½</td>
<td>5</td>
<td>4</td>
<td>2½</td>
<td>1</td>
</tr>
<tr>
<td>f₅</td>
<td>4</td>
<td>5</td>
<td>2½</td>
<td>2½</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>12½</td>
<td>22½</td>
<td>18½</td>
<td>11</td>
<td>10½</td>
</tr>
</tbody>
</table>

Table 6. Global comparison of GA versions on the test suite

The ranking of the GAs on each function

In order to give a clear overview on the performance of all GA variants we rank the GAs for each function. In particular, we award the fastest GA, or the GA closest to the minimum, one point, the second fastest GA two points, and so on, so the worst performing GA for a given function gets five points. If for a particular function two GA’s finish real close to each other, we award them equally: add the points for both those rankings and divide that by two. After calculating these points for each function
and each GA variant we add the points for all the functions to form a total for each
GA. The GA with the least points has the best overall performance.

Conclusions

The performance of the self-adapting parameters $p_m$ and $p_c$ is disappointing when
they are on their own (in SAMGA and SAXGA). The most likely reason is that time
spent on searching good parameter values is time taken away from finding the
optimum. Another important reason for the SAMGA is that it does not use the 1/5
success rule as Fogarty and Smith did [FoSm96]. They generate five individuals with
mutation and choose only to insert the best one into the population. This gives extra
survival pressure and that seems to give better values for $p_m$ at a faster rate. The
SAXGA is not as bad as the SAMGA. It does perform quite well on the relatively
easy functions $f_1$ and $f_2$, but on the other three functions it finishes one but last or last.
Adapting the population size in the SAPGA, on the other hand, is very effective. The
maximum lifetime of 11 sets high pressure on survival and this keeps the population
size small. This clearly benefits the search, for the SAPGA is the second best GA in
this research and not far from being the best.

The overall competition ends in a close finish with the SAMXPGA as number one
and the SAPGA right on its heels. It did surprise us using adaptive population sizes
proved to be the key feature to improve the basic TGA. Alone, or in combination with
the self-adaptive variation operators, the mechanism to adjust the population size
during a run causes a consequent performance improvement w.r.t. the benchmark GA.

The main objective of this paper has been the study of a GA that has no parameters
– or at least is freed from the three main parameters $p_m$, $p_c$ and $N$. Our experiments
showed that such a GA outperforms the four other variants we examined in this study.
Nevertheless, it could be hypothesized that the main source of this victory is the
adaptation of the population size ($N$). The self-adaptive $p_m$ and $p_c$ do contribute to its
good performance (compare APGA with SAMXPGA) but we also conjecture that it
works the other way around too. Namely, that the high selective pressure in the
relatively small adaptive populations helps to get better $p_m$ and $p_c$ values.

These outcomes give a very strong indication that, in contrary to past and present
practice (where quite some effort is devoted to tuning or on-line controlling of the
application rates of variance operators), studying control mechanisms for variable
population sizes should be paid more attention.

Our future research is directed to verifying these outcomes on a larger test suite,
including real-world problems and a detailed analysis of varying population sizes
(comparing deterministic time-varying schemes with adaptive control).
References

Using Dynastic Exploring Recombination to Promote Diversity in Genetic Search

Carlos Cotta, José M. Troya

Dpto. de Lenguajes y Ciencias de la Computación, Univ. de Málaga
Campus de Teatinos (3.2.49), 29071 - Málaga - Spain
{ccottap, troya}@lcc.uma.es

Abstract. A family of recombination operators is studied in this work. These operators are based on keeping and using certain information about the past evolution of the algorithm to guide the recombination process. Within this framework, several recombination operators are specifically designed to preserve diversity within the population, while avoiding implicit mutations. The empirical evaluation of these operators on instances of two test problems ($k$–EMP and permutation flowshop) shows an improvement of the results with respect to other classical operators. This improvement seems to related to the increasing degree of epistasis of the problem.

1 Introduction

One of the most distinctive features of genetic algorithms with respect to other related techniques such as evolution strategies or evolutionary programming is the emphasis put on the use of recombination operators. In the most classical scenario, recombination is given an exploitative rôle: its purpose is to combine valuable portions of solutions independently discovered, while the more exploratory task of introducing new genetic material corresponds to the mutation operator [11]. In this context, the loss of diversity is one of the main problems that can take place, resulting in the stagnation of the algorithm (which turns to be incapable of producing new promising solutions) and the increasing resampling of solutions [4] (with the subsequent waste of computational resources).

These problems are usually tackled by means of rising the mutation rate [10] or by using non-panmictic populations\(^1\) [17, 19]. Despite both options can be effective, they require determining and adjusting several parameters (rate of change of the mutation rate, interconnection topology, migration frequency, etc.). This is generally a difficult step, and in spite of active research being conducted to assist setting these parameters (e.g., see [2, 3]), trial-and-error is still very often used by many researchers. In this work, an alternative (and complementary) approach is presented. This approach is termed Dynastic Exploring Recombination and comprises a family of recombination operators that consider the past

\(^1\) Populations with spatial structure: islands, grids, etc.
evolution of the algorithm, trying to build promising unexplored macroformae [14]. This family of operators has been successfully evaluated on two problems: the $k$–EMP problem (a tunable-epistasis theoretical problem) and makespan minimisation in permutation flowshop problems [12].

The remainder of the paper is organised as follows. First, the basis of Dynastic Exploring Recombination is shown (Sect. 2). Subsequently, an empirical evaluation of these recombination operators is presented, focusing on aspects related to diversity and epistasis (Sect. 3). Finally, some conclusions are extracted and future work is outlined (Sect. 4).

2 Dynastic Exploring Recombination

Before defining Dynastic Exploring Recombination (DER) some previous concepts must be stated. Such background is presented is Subsect. 2.1. Next, some insight on the internal mechanics of recombination is provided in Subsect. 2.2. Finally, random transmission and dynastic exploring operators are introduced in Subsect. 2.3.

2.1 Background

Let $\Xi = \{\psi_1, \ldots, \psi_n\}$ be a set of $n$ independent equivalence relations defined over a discrete search space $\mathcal{S}$ verifying that for every possible pair of solutions $x$ and $y$, there exists at least an equivalence relation in $\Xi$ such that $x$ and $y$ belong to different equivalence classes. In this case, $\Xi$ covers the search space $\mathcal{S}$, and every $x \in \mathcal{S}$ can be univocally represented as $x = \{\eta_1, \ldots, \eta_n\}$, where $\eta_i$ is the equivalence class (for simplicity, the same symbol is used both to denote an equivalence class and for labelling it) to which $x$ belong under $\psi_i$. Thus, $x = \{\eta_1, \ldots, \eta_n\} \iff x \in \bigcap_{i=1}^n \eta_i$. Each of these equivalence classes $\eta_i$ is termed a basic forma [14].

Let $x = \{\eta_1, \ldots, \eta_n\}$ and $y = \{\zeta_1, \ldots, \zeta_n\}$ be two feasible solutions. A recombination operator $X$ can be defined as a function $X : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \rightarrow [0,1]$, where $X(x,y,z)$ is the probability of generating $z$ when $x$ and $y$ are recombined using $X$. Now, the Immediate Dynastic Span [15] of $x$ and $y$ with respect to a recombination operator $X$ is $\Gamma_X^I(\{x,y\}) = \{z \mid X(x,y,z) > 0\}$, i.e., the set of solutions that can be obtained when $X$ is applied on $x$ and $y$. On the other hand, the Dynastic Potential $\Gamma(\{x,y\})$ of $x$ and $y$ is defined as $\Gamma(\{x,y\}) = \{z \mid \forall \xi : z \in \xi \Rightarrow (x \in \xi) \lor (y \in \xi)\}$, where $\xi$ is any basic forma. Hence, the dynastic potential of two individuals is the set of offspring that can be constructed using nothing but the information contained in the parents.

If $\Gamma_X^I(\{x,y\}) \subseteq \Gamma(\{x,y\})$, $X$ is termed a transmitting operator [15]. As already mentioned in Sect. 1, this property of transmission captures the most classical rôle of the recombination operator. Next subsection will study the functioning mechanics of these operators.
2.2 The Mechanics of Recombination

A transmitting recombination can be generally considered as a process in which information is incrementally taken from the parents to construct the offspring: starting from a totally unspecified solution, properties of the any of the parents are selected and assigned to the child until a fully-specified solution is obtained. This incremental process is not necessarily linear but may exhibit a more complex behaviour, including some kind of backtracking in the assignment of properties (e.g., Dynastically Optimal Recombination [5]). With this consideration in mind, any transmitting recombination operator $X_t$ can be defined as

$$X_t(x, y, z) = \prod_{i=1}^{n} \delta_{X_t}(\Psi_{i-1}, \xi_i, x, y),$$

where $z = \{\xi_1, \ldots, \xi_n\}$ and $\delta_{X_t}$ is a function $\delta_{X_t} : 2^S \times 2^S \times S \times S \rightarrow \{0, 1\}$. In this function, the first parameter represents the partially-specified solution at a certain step, i.e., $\Psi_0 = S$, $\Psi_i = \Psi_{i-1} \cap \xi_i$. Since $\Sigma$ covers $S$, it is easy to see that $\Psi_n = \{z\}$. Now, $\delta_{X_t}(\Psi, \xi, x, y)$ is the probability that $X_t$ selects forma $\xi$, given that the partially-specified solution so far is $\Psi \ni z$. Clearly, a transmitting operator must verify that

$$[(x \notin \xi) \land (y \notin \xi)] \Rightarrow \delta_{X_t}(\Psi, \xi, x, y) = 0. \quad (2)$$

The nature of $\delta_{X_t}$ allows classifying transmitting operators into different categories. In particular, this work will focus on the case in which no backtracking in the assignment of properties is performed. The resulting family of operators is hence characterised for having a linear functioning mechanics, thus introducing a lower computational overhead in the algorithm. Some elements of this family are studied in the next subsection.

2.3 Random Transmission vs. Exploring Transmission

The simplest way of performing transmission is at random. Operators working in this way are comprised within the so-called Random Transmitting Recombination (RTR) [15], which is defined as follows:

$$\text{RTR}(x, y, z) = \begin{cases} 0 & \text{if } z \in \Gamma(\{x, y\}) \\ \frac{1}{|\Gamma(\{x, y\})|} & \text{otherwise} \end{cases} \quad \text{for } z \in \Gamma(\{x, y\})$$

Thus, RTR returns a random individual in the dynastic potential of the recombined solutions. If $\Sigma$ is orthogonal, i.e., if all combinations of formae induced by different equivalence relations are feasible, it is easy to see that $\Gamma(\{x, y\}) = \prod_{i=1}^{n} \{\eta_i, \zeta_i\}$, that is, the $n$-dimensional Cartesian product of all basic formae to which $x$ or $y$ belong. In this situation, RTR is defined by the following $\delta_{\text{RTR}}$:

$$\delta_{\text{RTR}}(\Psi, \xi, x, y) = \begin{cases} 0 (x \notin \xi) \land (y \notin \xi) \\ 1 (x \in \xi) \land (y \in \xi) \\ 1/2 \text{ otherwise} \end{cases} \quad (4)$$
Notice that $RTR \equiv UX$ if each $\psi_i$ induce two equivalence classes. In a more general situation (non-orthogonal separability [14]), it is necessary to consider higher-level units termed compatibility sets [5] into which solutions are structured. Using the notation $\xi \triangleright \Psi$ to denote that, given $\Psi = \cap_{j=1}^{1} \theta_j$, $\exists j : \xi \equiv \theta_j$, where $\xi$ and $\theta_j$ are formae induced by the same equivalence relation $\psi_i \in \Xi$, each of these compatibility sets is inductively defined as:

$$\eta_{j_1} \triangleright K(\eta_{j_1}, x, y)$$

$$\left[ \cap \{x, y\} \right] \cap K(\eta_{j_1}, x, y) \cap \varpi(\eta_{j_k}, x, y) = \emptyset \Rightarrow \eta_{j_k} \triangleright K(\eta_{j_1}, x, y),$$

where $x \in \eta_{j_1}$ and $\varpi(\eta_{j_k}, x, y)$ is the dual forma of $\eta_{j_k}$, that is, $\varpi(\eta_{j_k}, x, y) = \zeta_{j_k}$ if $x \in \eta_{j_k}$, $y \in \zeta_{j_k}$ and both $\eta_{j_k}$ and $\zeta_{j_k}$ are formae induced by the same $\psi_r$ (idem changing $x$ by $y$). For the sake of simplicity, it is assumed that $K(\eta, x, y) = K(\eta, y, x)$ whenever $y \in \eta, x \notin \eta$. Now, RTR is determined by the following function $\delta_{RTR}^{K}$:

$$\delta_{RTR}^{K}(\Psi, \xi, x, y) = \begin{cases} 0 & \left[ (x \notin \xi) \land (y \notin \xi) \right] \lor \left[ \exists \xi \triangleright \Psi : \varpi(\xi, x, y) \triangleright K(\zeta, x, y) \right] \\ 1 & \left[ (x \in \xi) \land (y \in \xi) \right] \lor \left[ \exists \xi \triangleright \Psi : \xi \triangleright K(\zeta, x, y) \right] \\ 1/2 & \text{otherwise} \end{cases}$$

Thus, a random selection is done over compatibility sets rather than over isolated formae. As mentioned in Sect. 1, defining a uniform probability distribution over these compatibility sets may be inappropriate for several reasons. First, resampling becomes a problem as the algorithm converges [4]. Second, it is sensitive to stochastic errors due the finite size of the population (e.g., valuable genetic material may be lost before it is exploited).

In order to avoid these undesirable effects, recombination operators that consider the past evolution of the algorithm can be used. Such past evolution is comprised in the histogram function $h_A$ of algorithm $A$ under consideration. This function is defined as $h_A : S \rightarrow 2^N \times \mathbb{R}$, such that $h_A(x) = \{(e_1, f_1), \cdots, (e_k, f_k)\}$ implies that element $x$ has been evaluated in iterations $e_1, \cdots, e_k$, and has respectively obtained the fitness values $f_1, \cdots, f_k$ (typically $\forall i[f_i = f]$). Several ways exist for using the information provided by the histogram function. For example, it might be useful to avoid generating solutions that were already considered in the past. However, such an operator would be computationally expensive and would in general exhibit a non-linear behaviour. A much more efficient template is shown below (orthogonality is assumed for simplicity):

$$\delta_{DER}(\Psi, \xi, x, y) = \begin{cases} 0 & \left[ (x \notin \xi) \land (y \notin \xi) \right] \lor \left[ \phi(\varpi(\xi, x, y), h_A) < \phi(\xi, h_A) \right] \\ 1 & \left[ (x \in \xi) \land (y \in \xi) \right] \lor \left[ \phi(\xi, h_A) < \phi(\varpi(\xi, x, y), h_A) \right] \\ 1/2 & \text{otherwise} \end{cases}$$

$\text{This definition of compatibility sets slightly differs from [5] since separability implies independence with respect to the partially specified solution } \Psi.$
where $\phi$ is a function that returns a metric of forma $\xi$ with respect to the past history of the algorithm. According to this definition, DER (Dynastic Exploring Recombination) would try to combine formae that locally minimise such metric. We are especially concerned about promoting diversity in the population and hence we have considered the following possibilities:

- $\phi_1(\xi, h_A) = \sum_{x \in \xi} |h_A(x)|$, that is, the number of times a solution $x \in \xi$ has been evaluated.

- $\phi_2(\xi, h_A) = \sum_{x \in \xi} \sum_{E \in h_A(x)} f$, that is, the accumulated fitness of all solutions in $\xi$ evaluated so far.

These two metrics are intended to direct the search towards regions of the search space not yet explored or at least not so explored as other regions. Thus, both $\text{DER}_{\phi_1}$ and $\text{DER}_{\phi_2}$ try to mitigate the exploitative side of recombination, boosting exploration.

In addition to these two metrics, a third metric $\phi_3$ has been considered. This metric is defined as $\phi_3(\xi, h_A) = \phi_2(\xi, h_A) / \phi_1(\xi, h_A)$, that is, the average fitness of solutions in $\xi$ evaluated so far. This is an interesting measure since it fits the traditional vision of genetic algorithms as (above-average) schema processors, sharing some similitude with Rosete et al.'s explicit schema processing [16].

Notice that the bookkeeping involved in calculating any of these metrics is very simple and does not require storing every visited solution. In fact, it suffices to keep a table with the accumulated values of these metrics for each basic forma (the size of this table would be $n \times m$, where $n$ is the number of genes -equivalence relations in $E$- and $m$ is the number of alleles per gene - basic formae per equivalence relation-). Subsequently, gene values are picked according to the distribution probability shown in Eq. (8).

3 Experimental Results

This section describes the experimental evaluation of the previously described operator. First, the experimental setup used is described. The empirical results obtained are subsequently reported.

3.1 Experimental Setup

The test suite used for testing the operators described comprises instances of two families of problems: $k$-epistatic minimal permutation problem and makespan minimisation in permutation flowshop problems.

The $k$-Epistatic Minimal Permutation ($k$-EMP) problem is a generalisation of the Minimal Permutation (MP) problem [5]. The latter is a minimisation problem defined by a $n \times n$ matrix $M = \{m_{ij} \mid 1 \leq i, j \leq n\}$ such that each row of $M$ is a permutation of the elements $\{0, \ldots, n-1\}$ and no column has more than one zero. Subsequently, a permutation $p = p_1 p_2 \cdots p_n$ is evaluated as $\text{MP}(p) = \sum_{1 \leq i \leq n} m_{i,p_i}$. 
The constraints posed on $M$ ensure that there is a unique permutation (the minimal permutation) whose fitness value is 0. The $k$-$\text{EMP}$ problem adds epistatic relations to the above expression. To be precise, it is defined as

$$k - \text{EMP}(p) = \sum_{1 \leq i \leq n} \left[ m_{i,p_i} \cdot \prod_{j=\min(1,i-k)}^{i-1} \alpha(p_i,p_j) \right]. \quad (9)$$

In the instances considered in this work, the coefficients $\alpha(p_i,p_j)$ are drawn from a uniform distribution in $[1,2]$.

As to the second problem, it is a well-known member of the $\mathcal{NP}$-hard class. It involves determining the order in which a set of jobs must be fed into a production chain composed of a number of machines. Provided that each job requires exclusive use of each machine for a certain time, the goal is to minimise the total completion time of the jobs [6,12].

The solution space for both problems can be adequately represented in terms of non-orthogonal separable formae. More precisely, solutions for both problems are permutations of a set of elements, being the properties of these permutations appropriately grasped by means of both position and block formae. The former are defined as assignments of elements to individual positions, while the latter are defined as the intersection of a compact set of adjacent positions. Because of space limitations, we refer to [6,7] for a formal description of these formae.

All experiments have been conducted with a steady-state [18] genetic algorithm (popsize = 100, $p_c = .9$, $p_m = 1/n$, where $n$ is the dimensionality of the problem) using ranking selection ($\eta^+ = 2.0$) and the swap mutation operator [13]. Each run of the algorithm comprises $10^5$ evaluations of the target function.

### 3.2 Empirical Evaluation

Table 1 shows results for different recombination operators on $k$-$\text{EMP}$ instances of different dimensionalities and degrees of epistasis. As it can be seen, DER operators (with the exception of DER$_{\phi_3}$) achieve a very good performance with respect to transmitting operators such as UCX (RTR$^{\text{pos}}$) and UBX (RTR$^{\text{block}}$), as well as with respect to classical operators such as PMX and OX. In these experiments, the improvement is clearer when epistasis is increased. This seems reasonable, since exploitative behaviours are more appropriate in the case of low (or null) epistasis in order to proceed towards near-optimal solutions. In this context, any mechanism introduced for promoting exploration simply slows down convergence. The scenario is different in the case of higher epistasis: keeping diversity becomes an important issue, and DER is better than RTR at this. Fig. 1 illustrates this fact, using population-entropy as a measure of diversity [8].

An important fact that must be noted is that DER keeps a higher diversity than UCX while manipulating the same basic units, and without introducing implicit mutation. An operator such as OX may provide a diversity measure similar or higher than DER, but this is done at the expense of introducing a considerable amount of implicit mutation. In this case, such implicit mutation is clearly detrimental as shown in Table 1.
Table 1. Results averaged for ten runs of different recombination operators on $k$–EMP instances of 50 and 75 elements.

<table>
<thead>
<tr>
<th>Operator</th>
<th>50 elements</th>
<th>k = 0</th>
<th>k = 1</th>
<th>k = 2</th>
<th>k = 5</th>
<th>k = 10</th>
<th>k = 15</th>
<th>k = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>OX</td>
<td>134.4</td>
<td>184.4</td>
<td>271.6</td>
<td>614.0</td>
<td>2465.7</td>
<td>8582.5</td>
<td>25331.8</td>
<td></td>
</tr>
<tr>
<td>UCX (RTR$_{\text{pos}}$)</td>
<td>50.5</td>
<td>72.5</td>
<td>99.9</td>
<td>282.3</td>
<td>1060.5</td>
<td>4880.9</td>
<td>16303.6</td>
<td></td>
</tr>
<tr>
<td>UBX (RTR$_{\text{block}}$)</td>
<td>60.0</td>
<td>78.5</td>
<td>118.8</td>
<td>355.4</td>
<td>1256.1</td>
<td>4709.1</td>
<td>14877.1</td>
<td></td>
</tr>
<tr>
<td>PMX</td>
<td>61.3</td>
<td>91.2</td>
<td>104.9</td>
<td>312.6</td>
<td>1274.9</td>
<td>5103.0</td>
<td>17899.8</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>88.6</td>
<td>75.0</td>
<td>102.8</td>
<td>276.2</td>
<td>966.8</td>
<td>3657.5</td>
<td>10483.2</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>61.0</td>
<td>83.4</td>
<td>109.9</td>
<td>269.1</td>
<td>1015.8</td>
<td>3728.4</td>
<td>8154.3</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>53.0</td>
<td>80.6</td>
<td>109.5</td>
<td>272.3</td>
<td>930.2</td>
<td>2820.6</td>
<td>7960.8</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>53.6</td>
<td>71.9</td>
<td>102.7</td>
<td>267.3</td>
<td>943.4</td>
<td>3856.8</td>
<td>10874.4</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>88.6</td>
<td>114.0</td>
<td>162.0</td>
<td>413.7</td>
<td>1810.3</td>
<td>8431.0</td>
<td>31418.4</td>
<td></td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>61.0</td>
<td>97.5</td>
<td>143.0</td>
<td>377.2</td>
<td>1612.3</td>
<td>6279.6</td>
<td>28170.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>75 elements</th>
<th>k = 0</th>
<th>k = 1</th>
<th>k = 2</th>
<th>k = 5</th>
<th>k = 10</th>
<th>k = 15</th>
<th>k = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>OX</td>
<td>358.0</td>
<td>529.9</td>
<td>787.0</td>
<td>1927.1</td>
<td>8521.9</td>
<td>31552.1</td>
<td>111858.8</td>
</tr>
<tr>
<td>UCX (RTR$_{\text{pos}}$)</td>
<td>144.6</td>
<td>227.1</td>
<td>325.2</td>
<td>858.7</td>
<td>4309.8</td>
<td>18243.2</td>
<td>74282.2</td>
</tr>
<tr>
<td>UBX (RTR$_{\text{block}}$)</td>
<td>193.0</td>
<td>263.3</td>
<td>369.8</td>
<td>1059.6</td>
<td>4916.9</td>
<td>18388.9</td>
<td>75397.1</td>
</tr>
<tr>
<td>PMX</td>
<td>193.3</td>
<td>259.8</td>
<td>377.1</td>
<td>1009.2</td>
<td>4460.0</td>
<td>17644.6</td>
<td>76752.7</td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>174.0</td>
<td>261.3</td>
<td>396.5</td>
<td>1023.5</td>
<td>4193.0</td>
<td>15629.9</td>
<td>56070.4</td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>180.7</td>
<td>246.3</td>
<td>338.5</td>
<td>950.8</td>
<td>3988.1</td>
<td>16997.7</td>
<td>54932.3</td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>192.0</td>
<td>256.2</td>
<td>375.2</td>
<td>1028.7</td>
<td>4189.1</td>
<td>17043.4</td>
<td>63446.8</td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>180.7</td>
<td>245.4</td>
<td>354.3</td>
<td>904.8</td>
<td>3950.5</td>
<td>17486.6</td>
<td>56186.0</td>
</tr>
<tr>
<td>DER$_{\text{pos}}$</td>
<td>213.3</td>
<td>300.4</td>
<td>449.7</td>
<td>1148.3</td>
<td>6320.1</td>
<td>24476.2</td>
<td>122873.6</td>
</tr>
<tr>
<td>DER$_{\text{block}}$</td>
<td>202.4</td>
<td>294.2</td>
<td>394.8</td>
<td>1149.6</td>
<td>5420.5</td>
<td>21559.1</td>
<td>91910.5</td>
</tr>
</tbody>
</table>

Fig. 1. Loss of diversity in a run of the GA for different recombination operators (UCX vs. DER$_{\phi_1}$. The graph corresponds to a 20-EMP instance. Variants in DER$_{\{\phi_1\phi_2\}}$ behave very similar to DER$_{\phi_1}$. 
The fact that DER$_{\phi_3}$ provide comparatively worse results can be explained as follows: unlike DER$_{\phi_1}$ and DER$_{\phi_2}$, DER$_{\phi_3}$ does not receive any negative feedback from the histogram function $h_A$. Thus, if DER$_{\phi_3}$ directs the search towards a certain region of $S$ in which several formae have above-average fitness, it is simply boosting the theoretical behaviour of the algorithm as a forma/schema processor, increasing the rate of convergence (see Fig. 2 - right) and hence the possibility of falling into local optima. On the contrary, both DER$_{\phi_1}$ and DER$_{\phi_2}$ receive negative feedback, and try to direct the algorithm to regions of $S$ not yet explored. Moreover, they try to avoid strongly-exploited regions. As mentioned above, this compensatory strategy can be more useful in the presence of epistasis.

These results are confirmed on flowshop scheduling instances taken from the OR-Library [1] (see Table 2). It must be noted that this problem has a high degree of epistasis, since scheduling a task at a given position affects all tasks subsequently scheduled. For this reason, both DER$_{\phi_1}$ and DER$_{\phi_2}$ yield the best results on these instances.

4 Conclusions

This work has studied the use of recombination operators specifically designed to promote exploration. This is done by keeping information about the past evolution of the algorithm in order to compensate the exploitative side of recombination. Unlike some related approaches (e.g., CHC [9]), rather than preventing the recombination of similar parents, the generation of different descendants is sought. It is important to consider that, according to the No Free Lunch Theorem [20], there exist problems in which the past information could mislead the algorithm. This seems to be the case of low-epistatic problems, in which exploitation is a better strategy and hence this compensation is detrimental.
Table 2. Results averaged for ten runs of different recombination operators on permutation flowshop instances.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Rec19</th>
<th>Rec25</th>
<th>Rec31</th>
<th>Rec37</th>
<th>Rec39</th>
<th>Rec41</th>
</tr>
</thead>
<tbody>
<tr>
<td>OX</td>
<td>2121.8</td>
<td>2571.6</td>
<td>3144.1</td>
<td>5223.3</td>
<td>5296.9</td>
<td>5249.7</td>
</tr>
<tr>
<td>UCX (RTR_{pos})</td>
<td>2126.6</td>
<td>2566.7</td>
<td>3127.0</td>
<td>5192.1</td>
<td>5270.6</td>
<td>5206.4</td>
</tr>
<tr>
<td>UBX (RTR_{block})</td>
<td>2124.3</td>
<td>2575.2</td>
<td>3124.5</td>
<td>5197.2</td>
<td>5275.6</td>
<td>5205.8</td>
</tr>
<tr>
<td>PMX</td>
<td>2123.7</td>
<td>2577.2</td>
<td>3120.0</td>
<td>5208.0</td>
<td>5282.7</td>
<td>5207.2</td>
</tr>
<tr>
<td>DER_{pos}</td>
<td>2119.4</td>
<td>2557.8</td>
<td>3122.3</td>
<td>5159.5</td>
<td>5268.9</td>
<td>5189.0</td>
</tr>
<tr>
<td>DER_{block}</td>
<td>2119.5</td>
<td>2553.0</td>
<td>3122.2</td>
<td>5179.9</td>
<td>5337.2</td>
<td>5189.8</td>
</tr>
<tr>
<td>DER_{pos}</td>
<td>2116.9</td>
<td>2551.8</td>
<td>3111.1</td>
<td>5172.1</td>
<td>5272.8</td>
<td>5188.6</td>
</tr>
<tr>
<td>DER_{block}</td>
<td>2115.6</td>
<td>2563.0</td>
<td>3115.0</td>
<td>5174.3</td>
<td>5247.7</td>
<td>5194.2</td>
</tr>
<tr>
<td>DER_{pos}</td>
<td>2134.9</td>
<td>2607.5</td>
<td>3147.0</td>
<td>5210.0</td>
<td>5307.3</td>
<td>5245.5</td>
</tr>
<tr>
<td>DER_{block}</td>
<td>2125.6</td>
<td>2570.2</td>
<td>3137.0</td>
<td>5211.2</td>
<td>5279.5</td>
<td>5258.6</td>
</tr>
</tbody>
</table>

The experimental results are encouraging. It is empirically corroborated that some members of the DER family keep a higher level of diversity. Furthermore, performance improves in the presence of appreciable levels of epistasis, scenario in which convergence to local optima becomes more of an issue.

Several lines of future work are open. Firstly, it is clearly necessary to conduct more experiments so as to verify these results on other test problems. Secondly, it is interesting to study the relationship and interplay between these operators and spatially-structured evolution models. These two strategies can complement each other as mentioned in Sect. 1. In this sense, factors such as the convenience of using decentralised histogram functions are worth studying. Work is already in progress in this area.

Acknowledgments

The authors wish to thank the anonymous reviewers for their interesting comments. Limitation of space has prevented us for further elaborating on some of the points they judiciously comment.

This work is partially supported by the Spanish Comisión Interministerial de Ciencia y Tecnología (CICYT) under grant TIC99-0754-C03-03.

References


Adaptive Control of the Mutation Probability by Fuzzy Logic Controllers*

Francisco Herrera and Manuel Lozano
Department of Computer Science and Artificial Intelligence
University of Granada
18071 - Granada, Spain

Abstract. A problem in the use of genetic algorithms is premature convergence, a premature stagnation of the search caused by the lack of population diversity. The mutation operator is the one responsible for the generation of diversity and therefore may be considered to be an important element in solving this problem. A solution adopted involves the control, throughout the run, of the parameter that determines its operation: the mutation probability.

In this paper, we study an adaptive approach for the control of the mutation probability based on the application of fuzzy logic controllers. Experimental results show that this technique consistently outperforms other mechanisms presented in the genetic algorithm literature for controlling this genetic algorithm parameter.

1 Introduction

Population diversity is crucial to a genetic algorithm’s ability to continue the fruitful exploration of the search space. If the lack of population diversity takes place too early, a premature stagnation of the search is caused. Under these circumstances, the search is likely to be trapped in a local optimum before the global optimum is found. This problem, called premature convergence, has long been recognised as a serious failure mode for genetic algorithms (GAs) ([11]).

The mutation operator may be considered to be an important element for solving the premature convergence problem, since it serves to create random diversity in the population. The diversity levels introduced by this operator depend directly on the value for the mutation probability \( p_m \). However, finding robust values for this parameter that allow the premature convergence problem to be avoided in any problem is not a trivial task, since its interaction with the performance of a GA is complex and the optimal choice is problem dependent ([2]). Furthermore, different \( p_m \) values may be necessary during the course of a run for inducing an optimal exploration/exploitation balance. For these reasons, different techniques have been suggested that dynamically adjust \( p_m \) during the course of evolving a solution ([2, 3, 4, 12, 23, 24]). They try to offer suitable diversity levels for avoiding premature convergence and improving the results.

One of these techniques lies in the application of fuzzy logic controllers (FLCs) ([10]) for adapting \( p_m \) depending on either the current state of the search or

* This research has been supported by DGICYT PB98-1319.
other GA related parameters. Although much work on the adaptation of $p_m$ by FLCs has appeared, we consider that it suffers from some important deficiencies. On the one hand, most approaches adapt $p_m$ along with other GA parameters (population size ($N$), crossover probability ($p_c$), etc.) ([19, 22, 25, 26]), and so, the specific effects on GA performance derived from the adaptation of $p_m$ were not studied. Only in [20], an experimental study was presented, which was aimed at isolating the effects of the adaptation by FLCs of $N$, $p_c$, and $p_m$. It showed that the adaptation of $p_m$ contributes most to high performance. On the other side, there is a generalised lack of empirical comparisons between mechanisms for adapting $p_m$ based on FLCs and other types of mechanisms proposed for controlling this GA parameter ([2, 3, 4, 12, 23, 24]).

The goal of this paper is to report on an investigation about the adaptation of $p_m$ by means of FLCs that attempts to overcome the deficiencies previously remarked. In order to do this, we propose and study an adaptive mechanism based on an FLC that controls $p_m$ depending on: 1) a measure that describes the advance produced by a GA over the last generations, and 2) the $p_m$ value used during these generations. According to these two factors, the FLC returns a new $p_m$ value that will be used during the next generations. Its objective is to control the population diversity for escaping of a possible convergence premature or for allowing a favourable evolution to be even better.

The paper is set up as follows. In Section 2, we review different techniques for controlling $p_m$ presented in the literature. In Section 3, we specify the inputs, output, data base, and rule base of the proposed FLC. In Section 4, we compare the results of the proposal on a given test suite against the ones obtained using other mechanisms for controlling $p_m$ and the ones reached using different fixed $p_m$ values. Finally, some concluding remarks are offered in Section 5.

### 2 Techniques for Controlling the Mutation Probability

According to [8], the mechanisms presented for controlling parameters associated with GAs may be assigned to the following three categories:

- **Deterministic control.** It takes place if the values of the parameters to be controlled are altered by some deterministic rule, without using any feedback from the GA. Usually, a time-varying schedule is used.

- **Adaptive control.** It takes place if there is some form of feedback from the GA that is used to determine the direction and/or magnitude of the change to the parameters to be controlled. The rules for updating parameters that are used by this type of control and, by the previous one, are termed *absolute adaptive heuristics* ([1]) and, ideally, capture some lawful operation of the dynamics of the GA over a broad range of problems.

- **Self-adaptive control.** The parameters to be controlled are encoded onto the chromosomes of the individual and undergo mutation and recombination.

Next, we describe mechanisms for the control of $p_m$ that belong to each one of these categories.
2.1 Deterministic Control of \( p_m \)

A direction followed by GA research for the variation of \( p_m \) lies in the specification of an externally specified \emph{schedule} which modifies it depending on the time, measured by the number of generations. A time-dependency of \( p_m \) was first suggested by Holland ([18]) himself, although he did not give a detailed choice of the parameter for the time-dependent reduction of \( p_m \). Moreover, there is a theoretical argument for introducing a time-dependent schedule of \( p_m \) ([2]).

One of the most considered schedules consists in the decreasing of \( p_m \) during the GA run ([4, 12]). This schedule follows the absolute adaptive heuristic \emph{"to protect the exploration in the initial stages and the exploitation later"}, which has been considered for designing other search techniques, such as simulated annealing.

We consider a linear function to control the decrease of \( p_m \), following the idea presented in [4]. It constrains \( p_m(t) \) so that \( p_m(0) = p_m^h \) and \( p_m(T) = p_m^l \) if a maximum of \( T \) generations are used:

\[
p_m(t) = p_m^h - \frac{p_m^h - p_m^l}{T} \cdot t \quad 0 \leq t \leq T.
\]

2.2 Adaptive Control at Individual Level of \( p_m \)

In [23], a technique for the adaptive control at individual level of \( p_m \) was proposed, in which \( p_m \) is varied depending on the fitness values of the solutions. Each chromosome \( C_i \) has its own associated \( p_m \) value, \( p_m^i \), which is calculated as (maximization is assumed):

\[
p_m^i = k_1 \cdot \frac{f_{\text{max}} - f_i}{f_{\text{max}} - \bar{f}} \text{ if } f_i \geq \bar{f}, \quad \text{and } p_m^i = k_3 \text{ if } f_i < \bar{f},
\]

where \( f_i \) is the chromosome's fitness, \( f_{\text{max}} \) is the population maximum fitness, \( \bar{f} \) is the mean fitness, and \( k_1 \) and \( k_3 \) are 1. In this way, high-fitness solutions are protected \( (p_m^i = 0) \), whilst solutions with subaverage fitnesses are totally disrupted \( (p_m^i = 1) \). Furthermore, this technique increases \( p_m \) when the population tends to get stuck at a local optimum and decreases it when the population is scattered in the solution space.

2.3 Self-Adaptive Control of \( p_m \)

In [4, 24], a self-adaptive technique for controlling \( p_m \) is presented, in which the genotype of an individual consists of a binary-coded solution and an associated individual real-coded mutation probability, which evolves with the solution.

Next, we describe a technique that follows this idea. An extra gene, \( p_m^i \), is added to the front of each bitstring, \( C_i \), which represents the mutation probability for all the genes in this bitstring. The values of \( p_m^i \) are allowed to vary from \( p_m^i \) to \( p_m^h \). The following steps are considered for mutating the genes in a chromosome \( C_i \):

1. Apply a \emph{meta-mutation} on \( p_m^i \) obtaining \( p_m^i' \). This is carried out by choosing a randomly chosen number from the interval \([p_m^i - \delta, p_m^i + \delta]\), where \( \delta \) is a control parameter.
2. Mutate the genes in $C_i$ according to the mutation probability $p'^i_m$.
3. Write the mutated genes (including $p'^i_m$ value) back to the chromosome.

Crossover is presently applied only to the binary vector and has no impact on $p'^i_m$. Each offspring resulting from crossover receives the $p'^i_m$ value of one of its parents. The initial $p'^i_m$ values are generated at random from $[p'_m, p'_m]$.

3 Adaptive Control of $p_m$ by Fuzzy Logic Controllers

The interaction of GA control parameter settings and GA performance is generally acknowledged as a complex relationship which is not completely understood. Although there are ways to understand this relationship (for instance, in terms of stochastic behaviour), this kind of understanding does not necessarily result in a normative theory. FLCs are particularly suited to environments that are either ill-defined or are very complex. The adaptive control of GA parameters is one such complex problem that may benefit from the use of FLCs.

Applications of FLCs for the adaptive control of GA parameters are found in [16, 17, 19, 20, 22, 25, 26]. Their main idea is to use an FLC whose inputs are any combination of GA performance measures or current control parameters and whose outputs are GA control parameters. Current performance measures of the GA are sent to the FLC, which computes new control parameter values that will be used by the GA.

In this section, we present an FLC for the adaptive control of $p_m$ for a generational GA, during its run. The FLC is fired at each $G$ generations and $p_m$ is fixed over the generations in these time intervals. The FLC takes into account the $p_m$ value used during the last $G$ generations and the improvement achieved on $f_b$ (fitness of the best element found so far). Then, it computes a new value for $p_m$, which will be used during the next $G$ generations. Its goal is to observe the effects of a $p_m$ value on GA performance during $G$ generations, and produce a new $p_m$ value that properly replies against a possible poor rate of convergence, or that allows performance to be improved even more (in the case of past suitable progress). The FLC uses fuzzy rules capturing adaptive strategies that attempt to accomplish this task (an example is: use a higher value for $p_m$ when observing no progress on $f_b$, with the aim of introducing diversity).

The proposed FLC has two inputs:

- The current mutation probability, $p'^i_m$, which will be kept in the interval $[0.001, 0.01]$. This interval was chosen since it contains a wide spectrum of $p_m$ values that were considered frequently in the GA literature (e.g. $p_m = 0.001$ ([9]) and $p_m = 0.01$ ([15])).

- A convergence measure (minimization is assumed): $CM = \frac{f^*_b}{f^*_{b_{G}}}$, where $f^*_b$ is the fitness of the current best element found so far and $f^*_{b_{G}}$ is the fitness of the best element found before the last $G$ generations. If an elitist strategy is used, $CM$ will belong to $[0,1]$. If $CM$ is high, then convergence is high, i.e. no progress was made during the last $G$ generations, whereas if it is low, the GA found a new best element, which consistently outperforms the previous one.
The set of linguistic labels associated with $p_m^n$ is \{Low, Medium, High\}. The meanings of these labels are depicted in Figure 1.b (for each linguistic term, there is a triangular fuzzy set that defines its semantic, i.e., its meaning). The set of linguistic labels for $CM$ is \{Low, High\}. Their meanings are shown in Figure 1.a.

The output of the FLC is the new $p_m$ value, $p_m^n \in [0.001, 0.01]$, which will be considered during the following $G$ generations. The set of linguistic labels associated with $p_m^n$ is \{Low, Medium, High\}. Their meanings are in Figure 1.c.

**Fig. 1.** Meaning of the linguistic terms associated with the inputs and the output

Fuzzy rules describe the relation between the inputs and output. Table 1 shows the fuzzy rule base used by the FLC presented.

**Table 1.** Fuzzy rule base for the control of $p_m$

<table>
<thead>
<tr>
<th>Rule</th>
<th>$CM$</th>
<th>$p_m^o$</th>
<th>$p_m^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>4</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>5</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>6</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
</tbody>
</table>

The absolute adaptive heuristic underlying fuzzy rules 1, 2, and 4-6 is: "decrease $p_m$ when progress is made, increase it when there are no improvements". If a stationary state is detected ($CM$ high), there is a possible cause: $p_m^o$ is too low, which induces a premature convergence, with the search process being trapped in a local optimum. With the previous heuristic, this problem would be suitably tackled, since $p_m$ would be greater and so, more diversity is introduced with the possibility of escaping from the local optimum. However, another possible cause of a poor performance may be the use of a too high value for $p_m^o$, which does not allow the convergence to be produced for obtaining better individuals. Fuzzy rule 3 was included for dealing with this circumstance, since it proposes the use of a low $p_m$ value when $CM$ is high and $p_m^o$ is high.

4 Experiments

Minimization experiments on the test suite, described in Subsection 4.1, have been carried out in order to study the behaviour of the proposed FLC in the
previous section for controlling $p_m$. In Subsection 4.2, the algorithms built in order to do this are described, and finally, in Subsection 4.3, the results are shown and analysed.

4.1 Test Suite

For the experiments, we have considered six frequently used test functions:

**Sphere model** ($f_{SpPh}$) ([9]):

$$f_{SpPh}(x) = \sum_{i=1}^{n} x_i^2,$$

where $n = 10$ and $-5.12 \leq x_i \leq 5.12$. The fitness of the optimum is $f_{SpPh}(x^*) = 0$. This test function is continuous, strictly convex, and unimodal.

**Generalized Rosenbrock's function** ($f_{Ros}$) ([9]):

$$f_{Ros}(x) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2),$$

where $n = 2$ and $-5.12 \leq x_i \leq 5.12$. The fitness of the optimum is $f_{Ros}(x^*) = 0$. $f_{Ros}$ is a continuous and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom. This feature will probably cause slow progress in many algorithms since they must continually change their search direction to reach the optimum.

**Generalized Rastrigin's function** ($f_{Ras}$) ([3]):

$$f_{Ras}(x) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i),$$

where $n = 10$, $a = 10$, and $\omega = 2\pi$. The fitness of its global optimum is $f_{Ras}(x^*) = 0$. This function is a scalable, continuous, separable, and multimodal, which is produced from $f_{SpPh}$ by modulating it with $a \cdot \cos(\omega \cdot x_i)$.

**One-Max function** ($f_{One}$). For a string of binary digits, the fitness of a given string is the number of ones the string contains. The aim is to obtain a string containing all ones. A string length of 400 was used for the purposes of this study. To determine an individual’s fitness, the value of this function is subtracted from 400 (maximum value), in order to assign a fitness of zero to the optimum, and handle the problem by means of minimization.

**Fully deceptive order-3 function** ($f_{Dec}$) ([14]). In deceptive problems there are certain schemata that guide the search toward some solution that is not globally competitive. It is due since the schemata that have the global optimum do not bear significance and so they may not proliferate during the genetic process. The deceptive problem used consists of the concatenation of 13 subproblems of length 3 (a 39-bit problem). The fitness for each 3-bit section of the string is given in Table 2. The overall fitness is the sum of the fitness of these deceptive subproblems. To obtain an individual’s fitness, the value of this function is subtracted from 390 (maximum value). Therefore, the optimum has a fitness of zero.

**Royal Road** ($f_{RR}$) ([13]). This is a 200-bit problem that is comprised of 25 contiguous blocks of eight bits, each of which scores 8 if all of the bits are set to one. Although there is no deception in this problem there is an amount of epistasis. Again, an individual’s fitness is calculated by subtracting the value of this function from 200 (maximum value), being zero the fitness for the optimum.

<table>
<thead>
<tr>
<th>Chromosomes</th>
<th>000 001 010 100 110 011 101 111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>28 26 22 14 0 0 0 30</td>
</tr>
</tbody>
</table>
4.2 Algorithms

For experiments, we have considered a generational GA model that applies a simple crossover operator and a mutation clock operator. The selection probability calculation follows linear ranking ([5]) \((q_{\text{min}} = 0.5)\) and the sampling algorithm is the stochastic universal sampling ([6]). The elitist strategy ([9]) is considered as well.

The general features of the FLC are the following: the \(\text{min} \) operator is used for conjunction of clauses in the \(IF\) part of a rule, the \(\text{min} \) operator is used to fire each rule and the center of gravity weighted by matching strategy as the defuzzification operator is considered. This setting was chosen from [7], where experiments with several ones were tried, being the most effective.

Different algorithms were implemented which are differentiated according to the way followed for obtaining \(p_m\). They are shown in Table 3.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA1</td>
<td>(p_m = 0.001 ) fixed during the run.</td>
</tr>
<tr>
<td>GA2</td>
<td>(p_m = 0.005 ) fixed during the run.</td>
</tr>
<tr>
<td>GA3</td>
<td>(p_m = 0.01 ) fixed during the run.</td>
</tr>
<tr>
<td>GA-RAN</td>
<td>(p_m) is randomly chosen from ([0.001, 0.01]) for each generation.</td>
</tr>
<tr>
<td>GA-DET</td>
<td>Deterministic Control of the Mut. Prob. (Sect. 2.1).</td>
</tr>
<tr>
<td>GA-AIL</td>
<td>Adaptive Control at Individual-level of the Mut. Prob (Sect. 2.2).</td>
</tr>
<tr>
<td>GA-SELF</td>
<td>Self-Adaptive Control of the Mut. Prob ((\delta = 0.001)) (Sect. 2.3).</td>
</tr>
<tr>
<td>GA-FLC</td>
<td>Adaptive Control of the Mut. Prob. by FLC ((G = 50)) (Sect. 3).</td>
</tr>
</tbody>
</table>

Since we attempt to compare GA-FLC against other techniques for controlling \(p_m\) (GA-DET, GA-IL, and GA-SELF), we have considered that these techniques should handle the same range of possible \(p_m\) values than GA-FLC \(([0.001, 0.01])\). In order to do this, for the deterministic control of \(p_m\) (Sect. 2.1), we constrain \(p_m(t)\) so that \(p_m(0) = 0.01\) and \(p_m(T) = 0.001\). For the adaptive control at individual-level of \(p_m\) (Sect. 2.2), a transformation was made from the interval considered by this technique \(([0, 1])\) into \([0.001, 0.01]\). Finally, for the self-adaptive control of \(p_m\) (Sect. 2.3), we consider \(p^l_m = 0.001\) and \(p^h_m = 0.01\).

The parameters of the test functions \(f_{sp}, f_{ros}, f_{ras}\) were encoded into bit strings using binary reflected Gray coding, with a number of binary genes assigned to each one of 20. The population size is 60 individuals and the crossover probability \(p_c = 0.6\). We executed all the algorithms 30 times, each one with a maximum of 100000 evaluations.

4.3 Results

Table 4 shows the results obtained. The performance measures used are the following: 1) \(A\) performance: average of the best fitness function found at the end of each run, 2) \(SD\) performance: standard deviation, and 3) \(B\) performance: number of runs in which the algorithm achieved the best possible fitness value: \(2.4e - 10\) for \(f_{sp}\), \(1.5e - 9\) for \(f_{ros}\), \(4.8e - 8\) for \(f_{ras}\) (they are not zero due to the use of the Gray coding), and zero for \(f_{one}, f_{dec},\) and \(f_{rr}\).
Table 4. Results

<table>
<thead>
<tr>
<th>Alg.</th>
<th>( f_{Sph} )</th>
<th>( f_{Ros} )</th>
<th>( f_{Ras} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>SD</td>
<td>B</td>
</tr>
<tr>
<td>GA1</td>
<td>2.4e-10</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
<tr>
<td>GA2</td>
<td>7.7e-08</td>
<td>1.1e-07</td>
<td>0</td>
</tr>
<tr>
<td>GA3</td>
<td>8.0e-06</td>
<td>2.0e-05</td>
<td>0</td>
</tr>
<tr>
<td>GA-RAN</td>
<td>2.2e-07</td>
<td>3.4e-07</td>
<td>0</td>
</tr>
<tr>
<td>GA-DET</td>
<td>2.8e-10</td>
<td>1.2e-10</td>
<td>25</td>
</tr>
<tr>
<td>GA-IL</td>
<td>2.4e-10</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
<tr>
<td>GA-SELF</td>
<td>2.4e-10</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
<tr>
<td>GA-FLC</td>
<td>2.4e-10</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg.</th>
<th>( f_{One} )</th>
<th>( f_{Dec} )</th>
<th>( f_{RR} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>SD</td>
<td>B</td>
</tr>
<tr>
<td>GA1</td>
<td>0.0e+00</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
<tr>
<td>GA2</td>
<td>8.7e+00</td>
<td>2.4e+00</td>
<td>0</td>
</tr>
<tr>
<td>GA3</td>
<td>3.2e+01</td>
<td>3.1e+00</td>
<td>0</td>
</tr>
<tr>
<td>GA-RAN</td>
<td>1.2e+01</td>
<td>2.1e+00</td>
<td>0</td>
</tr>
<tr>
<td>GA-DET</td>
<td>6.7e-02</td>
<td>2.5e-01</td>
<td>28</td>
</tr>
<tr>
<td>GA-IL</td>
<td>4.3e+01</td>
<td>2.5e+00</td>
<td>0</td>
</tr>
<tr>
<td>GA-SELF</td>
<td>2.3e-01</td>
<td>4.2e-01</td>
<td>23</td>
</tr>
<tr>
<td>GA-FLC</td>
<td>0.0e+00</td>
<td>0.0e+00</td>
<td>30</td>
</tr>
</tbody>
</table>

With regards to the GA versions with fixed \( p_m \) values (GA1, GA2, and GA3), we may underline a very reasonable fact, the best A and B measures for each test function are reached with different \( p_m \) values:

- For the easy test functions, \( f_{Sph} \) and \( f_{One} \), with the lowest value, \( p_m = 0.001 \).
- For the functions with intermediate complexity, \( f_{Ras} \) and \( f_{RR} \), with the moderate value, \( p_m = 0.005 \).
- For the most complex functions, \( f_{Ros} \) and \( f_{Dec} \), with the highest value, \( p_m = 0.01 \).

A GA does not need too much diversity to reach the optimum of \( f_{Sph} \) and \( f_{One} \) since there is only one optimum which could be easily accessed. On the other side, for \( f_{Dec} \), the diversity is fundamental for finding a way to lead towards the optimum. Also, in the case of \( f_{Ros} \), diversity can help to find solutions close to the parabolic valley, and so avoid slow progress.

Now, considering the results of the GAs based on techniques for controlling \( p_m \), we may observe that the one based on FLCs, GA-FLC, has the most robust behaviour with regards to the A and B measures, since for each function, it returns results that are very similar to the ones of the most successful GAs with fixed \( p_m \) values (the case of \( f_{Sph}, f_{One}, f_{Ros}, \) and \( f_{Dec} \)), or better than all them (the case of \( f_{RR} \) and \( f_{Ras} \), for which it obtains very good results: 27 for B and 6.1e-8 for A). None of the remaining algorithms allows a better operation to be achieved.

In order to ascertain whether GA-FLC achieves this robust operation due to its adaptive control of \( p_m \), and not for applying different \( p_m \) values during the run, we compare its results with the ones of GA-RAN, which works in this way (it selects, at random, a \( p_m \) value for each generation). We may observe that GA-FLC clearly improves the results of GA-RAN. This indicates that its adaptation ability is responsible for the performance improvement, i.e., it allows suitable \( p_m \) values to be used for producing a robust operation for test functions with different difficulties.
Finally, we need to point out that the GA applying a deterministic control of $p_m$, GA-DET, offers the most significant resistance against the results for GA-FLC. This good behaviour may be caused by the use of the heuristic “first exploration, exploitation later”, which, when considered for controlling other type of parameters associated with GAs, has produced successful results as well. An example is the non-uniform mutation operator for real-coded GAs ([21]). It controls the mutation steps size following this strategy, showing high performance.

5 Conclusions

This paper presented a technique for the adaptive control of $p_m$ based on the use of FLCs. In particular, an FLC has been designed, which takes into account the $p_m$ value used during the last generations and a measure that quantifies the progress performed by the GA during these generations, and returns a new $p_m$ that will be used for attempting to gain a better evolution quality during the next generations.

The principal conclusions derived from the results of experiments carried out are the following:

- The technique presented is the most effective one for controlling $p_m$ as compared with other techniques proposed in the GA literature that have been considered for the experiments.

- The adaptation ability of this technique allows suitable $p_m$ values to be used for producing a robust operation for test functions with different difficulties.

Therefore, we may conclude that the adaptive control of $p_m$ by means of FLCs is a suitable way for improving the results of GAs. This conclusion show promise in the use of this technique for future applications and extensions. They include the following: 1) design of FLCs for the adaptive control at the individual level of $p_m$, and 2) study the adaptation of $p_m$ by FLCs for other types of GAs (real-coded GAs, genetic programming, etc.).

References


A Comparison of Two Representations for the Fixed Charge Transportation Problem

Jens Gottlieb and Christoph Eckert

Department of Computer Science, Technical University of Clausthal
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
{gottlieb,ceckert}@informatik.tu-clausthal.de

Abstract. In the past years several evolutionary algorithms have been applied to the fixed charge transportation problem (FCTP). Although suffering from a lack of locality, the Prüfer number representation has recently been suggested for the FCTP, and – to our surprise – it has been reported to yield good results. Since this disagrees with the common intuition that locality is an important prerequisite of successful evolutionary search, we analyse the Prüfer number representation in greater detail. The results show that the Prüfer number representation is superior to random search but clearly inferior to the permutation representation with respect to solution quality, which is explained by our locality analysis.

1 Introduction

The fixed charge transportation problem (FCTP) is stated as

\[
\text{minimize } \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij} + f_{ij} g(x_{ij})
\]

with

\[
g(x_{ij}) = \begin{cases} 
1 & \text{if } x_{ij} > 0 \\
0 & \text{else}
\end{cases}
\]

for \(i \in S\) and \(j \in D\)

subject to

\[
\sum_{j \in D} x_{ij} = a_i \quad \text{for } i \in S
\]

\[
\sum_{i \in S} x_{ij} = b_j \quad \text{for } j \in D
\]

\[
x_{ij} \geq 0 \quad \text{for } i \in S \text{ and } j \in D,
\]

with \(S = \{1, \ldots, m\}\) and \(D = \{1, \ldots, n\}\) denoting the sets of sources and destinations, respectively. Each source \(i \in S\) has a capacity \(a_i\), each destination \(j \in D\) has a demand \(b_j\), and \(x_{ij}\) is the amount to be shipped from \(i\) to \(j\). The variable and fixed costs are given by \(c_{ij}\) and \(f_{ij}\), respectively, and the goal is to determine a feasible transportation plan \(x = (x_{ij})\) with minimum costs. As the FCTP is NP-complete several heuristics have been developed, e.g., a tabu search based procedure [SAMD98] and evolutionary algorithms (EAs) [GP98, LGI98].

Vignaux and Michalewicz proposed EAs based on a matrix representation and a permutation representation for the linear transportation problem [VM91].
Gottlieb and Paulmann developed alternative matrix operators for the FCTP [GP98], which outperformed both EAs suggested in [VM91]. Recently, Gen et al. suggested using Prüfer numbers to encode solution candidates for the FCTP [LGI98] and its bicriteria variant [GL99]. They reported good results for this representation, which was originally proposed in [PK94].

In our opinion, the good results reported by Gen et al. for Prüfer numbers and the FCTP must be questioned for two reasons. First, they compared their approach to the matrix representation of Vignaux and Michalewicz only, an approach that has been outperformed by at least one order of magnitude [GP98]. Second, in addition to the early remarks of Palmer and Kershenbaum, who associated only relatively little locality to Prüfer numbers [PK94], Rothlauf and Goldberg reported that the locality of Prüfer numbers is highly irregular on the entire solution space [RG99]. Hence it is questionable whether Prüfer numbers are really competitive to other representations providing locality. Therefore, we present a direct comparison between the permutation representation and the Prüfer number representation, with special emphasis on locality properties.

Section 2 presents the representations and general results. Our empirical locality analysis is discussed in section 3, and conclusions are given in section 4.

2 Two Representations for the FCTP

It is known that the global optimum of the FCTP is a basic solution of the corresponding linear transportation problem. Therefore, it is beneficial to restrict evolutionary search to basic solutions only. Both decoders, which are described in sections 2.1 and 2.2, produce basic solutions and hence they are able to restrict the evolutionary search to the most promising part of the search space.

2.1 The Permutation Representation

A transportation plan \((x_{ij})\) consists of \(m \cdot n\) entries, which can be indexed by \(\{1, \ldots, m \cdot n\}\). An individual's genotype is a permutation of \(\{1, \ldots, m \cdot n\}\) that is decoded to its phenotype as follows: the decoder traverses the variables \(x_{ij}\) in the order determined by the genotype, and assigns to each \(x_{ij}\) the maximum value with respect to (3) and (4). It is easily verified that the decoder generates a basic solution and all basic solutions are represented by at least one permutation.

2.2 The Prüfer Number Representation

According to [GL99, LGI98], we denote sources by \(S = \{1, \ldots, m\}\) and destinations by \(D = \{m + 1, \ldots, m + n\}\). The transportation graph of the FCTP consists of the nodes \(S \cup D\) and the edges \(\{(i, j) \mid i \in S, j \in D\}\), which represent the possible transportation routes between sources and destinations. On the one hand, a basic solution \(x\) identifies a spanning tree in the transportation graph. On the other hand, a spanning tree induces (i) a basic solution \(x\) — in this case
the tree is called a transportation tree - or (ii) an unfeasible solution \( x \) that violates some constraints. Spanning trees can be represented by Prüfer numbers: a Prüfer number is a sequence of \( m + n - 2 \) (not necessarily distinct) nodes from \( S \cup D = \{1, \ldots, m + n\} \), which is constructed from a spanning tree as follows.

**Procedure: Convert spanning tree to Prüfer number**

**Step 1:** Let \( i \) be the lowest numbered leaf node in tree \( T \), and let \( j \) be its predecessor.

**Step 2:** Append \( j \) to the right of the Prüfer number \( P(T) \), and remove the node \( i \) and the edge \( \{i, j\} \) from \( T \).

**Step 3:** Stop if only two nodes remain to be considered; otherwise return to step 1.

While the construction of a Prüfer number from an arbitrary spanning tree is unique, an arbitrary sequence of \( m + n - 2 \) numbers from \( S \cup D \) does not necessarily represent a spanning tree in the transportation graph. The following cases may occur when considering a randomly generated Prüfer number:

1. The Prüfer number is feasible, i.e. it actually represents a spanning tree \( T \).
   (a) The induced solution \( x \) is feasible, i.e. \( T \) is a transportation tree.
   (b) The induced solution \( x \) violates some constraints (3) and (4).

2. The Prüfer number is unfeasible, i.e. it does not identify a spanning tree.

In case 1, the Prüfer number identifies a complete spanning tree, but some destination's demand may not be satisfied (case 1b) in the associated solution \( x \) and thus, \( x \) must be completed arbitrarily. Case 2 is due to the fact that Prüfer numbers were designed to represent trees in complete graphs, whereas the transportation graphs under consideration contain only edges connecting sources and destinations. Therefore, it may happen that an arbitrary Prüfer number induces edges connecting two sources or two destinations. In this case, the information contained in the Prüfer number is not sufficient to generate a complete spanning tree, and some "repair" algorithm is needed to complete the tree. Based on these considerations, we devised the following algorithm to generate a transportation tree.

**Procedure: Convert Prüfer number to transportation tree**

**Step 1:** Let \( P(T) \) be the Prüfer number and \( \hat{P}(T) \) be the set of nodes that are not contained in \( P(T) \).

**Step 2:** Let \( j \) be the lowest numbered node in \( \hat{P}(T) \). If \( P(T) \cup \{j\} \subseteq S \), then goto repair 1. If \( P(T) \cup \{j\} \subseteq D \), then goto repair 2.

**Step 3:** Let \( i \) be the leftmost digit of \( P(T) \), such that \( i \) and \( j \) are not in the same set \( S \) or \( D \). Add edge \( \{i, j\} \) to tree \( T \).

**Step 4:** Remove \( j \) from \( \hat{P}(T) \) and \( i \) from \( P(T) \). If \( i \) is not contained in the remaining part of \( P(T) \), insert \( i \) into \( \hat{P}(T) \).

**Step 5:** Swap \( i \) and \( j \) if \( i \in D \). Set \( x_{ij} \leftarrow \min\{a_i, b_j\} \), \( a_i \leftarrow a_i - x_{ij} \) and \( b_j \leftarrow b_j - x_{ij} \).

**Step 6:** If there are digits left in \( P(T) \), goto step 2. Otherwise, there are exactly two nodes \( i, j \) in \( \hat{P}(T) \). If \( \{i, j\} \subseteq S \) goto repair 1. If \( \{i, j\} \subseteq D \) goto repair 2.

**Step 7:** Assume \( i \in S \) and \( j \in D \), and set \( x_{ij} \leftarrow \min\{a_i, b_j\} \), \( a_i \leftarrow a_i - x_{ij} \) and \( b_j \leftarrow b_j - x_{ij} \). If all constraints are satisfied, stop; else goto repair 2.
Procedure: Repair 1

Step 1: Remove from $\bar{P}(T)$ all nodes of $S$.

Step 2: Let $j$ be the lowest numbered node in $\bar{P}(T)$ and $i$ be the leftmost digit of $P(T)$. Add edge $\{i, j\}$ to $T$ and assign $x_{ij} \leftarrow \min\{a_i, b_j\}$, $a_i \leftarrow a_i - x_{ij}$ and $b_j \leftarrow b_j - x_{ij}$. Remove $j$ from $\bar{P}(T)$ and $i$ from $P(T)$.

Step 3: If there are no digits left in $P(T)$ or $\bar{P}(T)$, goto repair 2; else goto step 2.

Procedure: Repair 2

Step 1: Create a permutation of all edges between unsatisfied sources and demands.

Step 2: Decode this permutation as described in section 2.1. Add those edges to $T$, which correspond to $x_{ij} > 0$. Transform $T$ into a spanning tree by removing/inserting edges $\{i, j\}$ with $x_{ij} = 0$.

Note that the first repair procedure makes use of the remaining information in the Prüfer number, while the second repair procedure satisfies the constraints in an arbitrary fashion in order to obtain a basic solution. While our algorithm always produces transportation trees, Gen et al. discarded unfeasible Prüfer numbers by eliminating them from evolutionary search [GL99, LGI98]. For this purpose, they proposed a simple feasibility test, which is stated as follows: Prüfer number $P(T)$ is feasible if and only if $|S| + |M| = |D| + |N|$, where $M = \{i \in P(T) \mid i \in S\}$ and $N = \{j \in P(T) \mid j \in D\}$ are multisets. The criterion is equivalent to $|M| = n - 1$ and $|N| = m - 1$, which leads us to

$$F(m, n) = \frac{m^{n-1}n^{m-1}(m+n-2)}{(m+n)^{m+n-2}}$$

as the fraction of feasible Prüfer numbers among all Prüfer numbers for problem dimensions $m, n$. Since even for feasible Prüfer numbers repairing may be needed (case 1b), $F(m, n)$ is an upper bound for the probability that an arbitrary Prüfer number can be directly converted into a transportation tree. In order to examine the relative frequencies of the cases 1a, 1b and 2 we randomly generated 1 000 000 Prüfer numbers for selected FCTP benchmarks1 that were examined in [GP98, SAMD98]. The results presented in table 1 show that $F(m, n)$ is an accurate predictor of case 1’s frequency. The frequency of case 2 is extremely

\[ \text{available at http://www.in.tu-clausthal.de/~gottlieb/benchmarks/fctp/} \]

\[ \text{Table 1. Frequencies of cases for decoding Prüfer numbers on selected FCTP instances} \]

<table>
<thead>
<tr>
<th>Problem</th>
<th>m</th>
<th>n</th>
<th>Frequencies [%]</th>
<th>$F(m, n)$ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td></td>
<td>1a</td>
<td>1b</td>
</tr>
<tr>
<td>bal8x12</td>
<td>8</td>
<td>12</td>
<td>0.01</td>
<td>3.77</td>
</tr>
<tr>
<td>ran4x64</td>
<td>4</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ran14x18</td>
<td>14</td>
<td>18</td>
<td>0</td>
<td>5.34</td>
</tr>
<tr>
<td>ran16x16</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>14.44</td>
</tr>
<tr>
<td>n370e</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
high, which indicates that the basic Prüfer number decoding is very ineffective. Hence the original EA of Gen et al. is expected to spend most of its time in discarding unfeasible Prüfer numbers, which underlines the importance of the repair procedures. Without repairing, even generating the initial population will fail for instances with high shape ratio \( \max(m, n) / \min(m, n) \).

2.3 Comparison of the Representations

We employ the following general setup: population size 100 (200 for \( n^{370e} \)), parent selection by tournaments of size 2, crossover probability 1, mutation probability 1, steady-state replacement with duplicate elimination (worst individual is replaced by offspring, if it is not already represented by some genotype in the population), and a limit of 1 000 000 non-duplicate solutions. Table 2 shows the considered representations and operator setups, which also include randomized variants in order to shed light on the relation to random search. Our comparison is based on 12 complete EA runs for each instance. The obtained solution quality is measured by the gap, defined as \( \min^{EA}/opt - 1 \) where \( \min^{EA} \) and \( opt \) are the best solution found by the EA and the best known solution, respectively. The efficiency of the EAs is measured by the duplicate ratio, which is the ratio of rejected duplicates among all generated solutions.

<table>
<thead>
<tr>
<th>EA</th>
<th>representation</th>
<th>crossover</th>
<th>mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE</td>
<td>permutation</td>
<td>uniform order based</td>
<td>swap</td>
</tr>
<tr>
<td>RPE</td>
<td>permutation</td>
<td>random child</td>
<td>random child</td>
</tr>
<tr>
<td>PN</td>
<td>Prüfer number</td>
<td>one-point</td>
<td>inversion</td>
</tr>
<tr>
<td>RPN</td>
<td>Prüfer number</td>
<td>random child</td>
<td>random child</td>
</tr>
</tbody>
</table>

Table 2. Considered evolutionary algorithms

The obtained average results are shown in table 3. Both randomized variants achieve comparable gaps – the worst among the considered EAs – and identical duplicate ratios, since they exhibit a uniform distribution among all basic solutions. As any meaningful EA should outperform random search concerning solution quality, both PE and PN seem to be reasonable. However, PE dominates the gap reported for PN by one order of magnitude, except for the smallest instance \( ba18x12 \) for which even the randomized variants yield small gaps. We claim that the solution quality depends on the achieved degree of locality.

While random search yields duplicate ratio 0 and PN produces only few duplicates, PE yields a significantly higher duplicate ratio. As PE and RPE induce the same redundancy – i.e. the relation between the sizes of genotype and phenotype space respectively – the high duplicate ratio of PE is caused by the locality properties of the employed operators. Although not that obvious, the same relation also holds for PN and RPN. Thus, locality influences the duplicate ratio and hence the efficiency of evolutionary search. While the duplicate ratio of PE is acceptable for smaller instances, the extremely high duplicate ratio for \( n^{370e} \) is a serious drawback of PE, since about half of all solutions are rejected.
### Table 3. Obtained gap and duplicate ratio for selected FCTP instances

<table>
<thead>
<tr>
<th>Problem</th>
<th>gap [%]</th>
<th>duplicate ratio [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PE</td>
<td>RPE</td>
</tr>
<tr>
<td>bal8x12</td>
<td></td>
<td>2.32</td>
</tr>
<tr>
<td>ran4x64</td>
<td>0.54</td>
<td>25.40</td>
</tr>
<tr>
<td>ran14x18</td>
<td>4.19</td>
<td>32.86</td>
</tr>
<tr>
<td>ran16x16</td>
<td>2.61</td>
<td>31.47</td>
</tr>
<tr>
<td>n370e</td>
<td>18.67</td>
<td>91.82</td>
</tr>
</tbody>
</table>

3 **Locality Analysis**

In order to evaluate the considered representations and operators concerning their degree of locality, we apply the methodology proposed by Gottlieb and Raidl [GR99], which successfully characterized locality properties of decoder-based evolutionary algorithms for the multidimensional knapsack problem. The approach is adapted to the FCTP by defining appropriate measures.

#### 3.1 Distance Metrics

Locality means that small changes in genotype cause small changes in phenotype. Therefore, similarity measures for both genotypes and phenotypes are needed. In the case of the FCTP, we define the *phenotypic distance* as

\[
d_P(x, y) := \sum_{i \in S} \sum_{j \in D} \delta(x_{ij}, y_{ij})
\]

with

\[
\delta(x_{ij}, y_{ij}) = \begin{cases} 
1 & \text{if } x_{ij} \neq y_{ij} \\
0 & \text{otherwise}
\end{cases}
\]

for transportation plans \(x, y\). This distance measures the number of different phenotypic properties. While the phenotypic distance is problem-dependent, the *genotypic distance* is representation-independent in order to allow a comparison of different representations. The genotypic distance is defined implicitly:

1. Two identical genotypes have distance 0, i.e. \(d_G(X, X) = 0\) for genotype \(X\).
2. Two distinct genotypes \(X, Y\) have distance \(d_G(X, Y) = 1\) if the probability of producing \(Y\) from \(X\) with a single mutation is greater than 0.
3. In general, a genotypic distance \(d_G(X, Y) = k\) for genotypes \(X, Y\) and \(k \geq 1\) means that at least \(k\) mutations are necessary to transform \(X\) to \(Y\).

Note that both \(d_P\) and \(d_G\) satisfy the metric conditions for the considered EAs.

#### 3.2 The Mutation Operator

Suppose an arbitrary genotype \(X\) and its offspring \(X^m\) generated by mutation, and let \(x\) and \(x^m\) be the corresponding phenotypes, then the *mutation innovation*

\[
MI := d_P(x, x^m)
\]
describes how much phenotypic “innovation” is generated by the mutation operator. MI is a random variable whose distribution can be estimated empirically in order to characterize the locality of mutation. We randomly generated 1 000 000 genotypes for each EA and FCTP instance, applied mutation to each, and calculated the mutation innovation for the genotypes. The average results for two interesting aspects of MI’s distribution are shown in table 4.

The measure \(P(MI = 0)\) indicates the probability that mutation does not modify the phenotype. High values of \(P(MI = 0)\) for PE show that the mutation frequently does not cause phenotypic changes, which is one reason for the high duplicate ratio. The mutation operators of RPE and RPN always modify the phenotypes, and PN is comparable to random search with respect to \(P(MI = 0)\).

The measure \(E(MI|MI > 0)\) reflects the expected innovation for the case that at least one phenotypic property has changed. High values signalize a low degree of locality, and an upper bound is expected for a mutation operator performing random jumps in the phenotype space. This agrees with our estimations as RPE and RPN exhibit the highest \(E(MI|MI > 0)\). The low values obtained for PE signalize locality for this representation. Although superior to random search, PN’s degree of locality is inferior to PE by several orders of magnitude.

| Problem | \(P(MI = 0)\) [%] | \(E(MI|MI > 0)\) |
|---------|-----------------|-----------------|
| PE      | 68.93          | 5.77            |
| RPE     | 0.00           | 26.81           |
| PN      | 0.55           | 6.82            |
| RPN     | 0.00           | 8.59            |
| ba18x12 | 64.41          | 5.71            |
| ran4x64 | 74.82          | 90.01           |
| ran14x18| 75.37          | 51.63           |
| ran16x16| 91.61          | 285.46          |
| n370e   | 91.61          | 285.29          |

Table 4. Empirically estimated \(P(MI = 0)\) and \(E(MI|MI > 0)\)

### 3.3 The Crossover Operator

Given the parental genotypes \(X^{p_1}\) and \(X^{p_2}\), crossover produces an offspring \(X^c\); let \(x^{p_1}, x^{p_2}, x^c\) be the corresponding phenotypes. The effects of meaningful crossover operators depend on the similarity of the parents. In order to estimate the genotypic similarity of the parents, we assume that \(X^{p_2}\) is produced by applying \(k \geq 1\) consecutive mutations to \(X^{p_1}\), which implies \(d_G(X^{p_1}, X^{p_2}) \leq k\). The **crossover innovation** is defined as

\[
CI_k := \min(d_P(x^c, x^{p_1}), d_P(x^c, x^{p_2})),
\]

the phenotypic distance of \(x^c\) to the nearer parent. Further, the **crossover loss**

\[
CL_k := \sum_{i \in S} \sum_{j \in D} \delta(x^c_{ij}, x^{p_1}_{ij}, x^{p_2}_{ij})
\]

with

\[
\delta(x^c_{ij}, x^{p_1}_{ij}, x^{p_2}_{ij}) = \begin{cases} 
0 & \text{if } x^c_{ij} = x^{p_1}_{ij} \text{ or } x^c_{ij} = x^{p_2}_{ij} \\
1 & \text{otherwise}
\end{cases}
\]
represents the number of phenotypic properties that neither stem from $x^{p_1}$ nor from $x^{p_2}$. While the crossover innovation $CI_k$ reflects the ability of introducing new phenotypic properties, the crossover loss $CL_k$ characterizes the amount of phenotypic information that is lost during crossover. In order to analyse crossover for some parent similarity $k$, we randomly generated 1 000 000 genotypes $X^{p_1}$ for each EA and FCTP instance. The associated parents $X^{p_2}$ are produced by applying $k$ mutations\(^2\) to each $X^{p_1}$, and crossover yields offspring $X^c$ of the parents. This procedure enables an empirical estimation of the distributions of $CI_k$ and $CL_k$. Figure 1 shows the obtained estimations of measures based on $CI_k$ and $CL_k$ for selected values of $k$ and the representative instance ran16x16.

The measure $P(CI_k = 0)$ reflects the probability that crossover produces an offspring whose phenotype is identical to one of its parents. Intuitively, a meaningful crossover yields higher values of $P(CI_k = 0)$ for similar parents, i.e. for small $k$. In the case of PE, this is a reason for the high duplicate ratio. The remaining EAs cause a negligible risk of generating duplicates.

As locality properties are characterized by the case $CI_k > 0$, i.e. the offspring differs from both parents, $E(CI_k|CI_k > 0)$ and $\sigma(CI_k|CI_k > 0)$ are of particular interest. Concerning $E(CI_k|CI_k > 0)$, only PE’s crossover provides locality since small values are observed for small $k$ and higher values emerge for larger $k$. Known to provide no locality at all, random search yields extremely high values. Despite the increase of $E(CI_k|CI_k > 0)$ for larger $k$, PN provides only weak locality due to the high values for small $k$. PN yields a higher degree of locality than RPE and RPN, but also a significantly lower level of locality than PE.

The corresponding standard deviations $\sigma(CI_k|CI_k > 0)$ must be interpreted carefully. On the one hand, the lowest deviation is obtained for the randomized variants since the resulting offspring is always generated in a random fashion, resulting in some kind of “uniformity”. On the other hand, a significantly higher deviation is exhibited by PN: while some offspring preserve parental properties, it also happens that the child does not inherit any properties and hence crossover resembles a kind of random jump in the phenotype space. The deviation reported for PE coincides with the behaviour described by $E(CI_k|CI_k > 0)$.

The crossover loss describes the ability to make use of parental phenotypic information. As $CI_k = 0$ implies $CL_k = 0$, we restrict our investigations to the case $CI_k > 0$. Any meaningful crossover induces a small value of $E(CL_k|CI_k > 0)$ for small $k$, and yields increasing values for higher $k$. PE provides locality since its crossover preserves most parental properties for small $k$ and yields higher values for high $k$. Although better than random search, only weak locality can be associated with PN due to the high $E(CL_k|CI_k > 0)$ for small $k$. There is a subtle difference between RPE and RPN, which is explained by the mutation operator used to generate the second parent. As the mutation operator of PE provides locality, the parent similarity is high for small $k$ and RPE. Generating a random child in this case yields a high crossover loss. Since the mutation operator

\(^2\) In order to allow an analysis of the crossover employed by RPE and RPN we use the corresponding mutation operators of PE and PN since this yields a reasonable parent similarity $k$. Note that otherwise identical results are obtained for all $k \geq 1$. 
Fig. 1. Empirically estimated $P(CI_k = 0)$, $E(CI_k | CI_k > 0)$, $\sigma(CI_k | CI_k > 0)$ and $E(CL_k | CI_k > 0)$ for ran16x16 and different parent similarities $d_C(X^{P_1}, X^{P_2}) \leq k$

of PN destroys much information, parent similarity is much lower for small $k$ and RPN, compared to RPE. Therefore the crossover loss of RPN is smaller. However, we observe convergence at the same level for RPE, RPN and high $k$.

In general, the crossover analysis showed that PE provides locality, PN yields weak locality, and random search provides no locality at all. This coincides with the mutation analysis, which also indicated that only PE achieves a reasonable degree of locality. Although the operators of PN sometimes cause small changes, they often perform random jumps, explaining PN’s high gap compared to PE.

4 Conclusions

This study presented a detailed comparison of two decoders which were suggested for the fixed charge transportation problem (FCTP). We successfully adapted the methodology that was recently introduced in [GR99] to characterize locality.

In general, the results confirmed our hypotheses made concerning the Prüfer number representation in an impressive fashion. While the permutation representation provides locality, Prüfer numbers yield only weak locality but are nevertheless superior to random search. The results show that locality has a strong impact on the obtained solution quality for the considered evolutionary algorithms. Prüfer numbers are not suitable to encode FCTP solutions since
their locality properties prevent a reasonable exploitation of phenotypic properties by evolutionary search. We believe that the superiority of Prüfer numbers to the matrix representation from [VM91] reported by Gen et al. [GL99, LGI98] is caused by the fact that the decoder generates basic solutions only. The matrix representation provides a higher degree of locality but tends to produce solutions with many positive entries [GP98], which is fatal due to the fixed costs. As the matrix based search spends most of its time in regions that do not contain the global optimum, the solution quality is inferior to the Prüfer number representation, which explores basic solutions with little fixed costs only.

Besides its locality, the permutation representation has a major drawback: its redundancy causes a high duplicate ratio and hence a waste of CPU time, in particular for large instances. We conclude that evolutionary search based on direct encoding of basic solutions and adequate variation operators is the most promising alternative since it provides locality and a compact encoding. The direct representation from [GP98] is attractive from this viewpoint as it actually outperforms the permutation representation. But limited experiments showed that, in its current stage, it cannot compete with the tabu search approach of Sun et al. [SAMD98]. Hence much work is needed to improve evolutionary algorithms for the FCTP, possibly by incorporating local optimization.

References


Invariance, Self-Adaptation and Correlated Mutations in Evolution Strategies

Nikolaus Hansen

Technische Universität Berlin
Fachgebiet für Bionik und Evolutionstechnik, Sekr. ACK 1
Ackerstr. 71–76, 13355 Berlin, Germany
E-mail: hansen@bionik.tu-berlin.de
URL: www.bionik.tu-berlin.de/user/niko/

Abstract. A conceptual objective behind the self-adaptation of the mutation distribution is to achieve invariance against certain transformations of the search space. In this paper, a priori invariances of a simple evolution strategy and invariances, which can be introduced by self-adaptation, are identified. In principle, correlated mutations can achieve invariance against any linear transformation of the search space. Correlated mutations, as typically implemented, are investigated with respect to both a priori and new invariances. Simulations reveal that neither all a priori invariances are retained, nor the invariance against linear transformation is achieved.

1 Introduction

The evolution strategy (ES) \[9,13\] addresses the search problem of minimizing a nonlinear objective function \( f : S \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(x) \). Search steps are taken by recombination of already evaluated search points and mutation. The mutation is usually carried out by adding a realization of a normally distributed random vector. A dynamic control of certain parameters of the normal mutation distribution is of major importance and is a common feature in evolution strategies. This is often called adaptation or self-adaptation.

A main objective of the adaptation of parameters of the mutation distribution can be interpreted as to achieve invariance against certain transformations of the search space. This is exemplified in the following.

Let the mutation step for the object parameter vector, \( x \in \mathbb{R}^n \), at generation \( g = 0, 1, 2, \ldots \) be

\[
x^{(g+1)} = x^{(g)} + \sigma^{(g+1)} \mathcal{N}(0, I)
\]

where \( \sigma \in \mathbb{R}^+ \) denotes the step size and \( \mathcal{N}(0, I) \) is a component wise independently \((0,1)\)-normally distributed random vector (\(I\) denotes the unity matrix as covariance matrix). Consider an adaptation of step size \( \sigma \) minimizing \( f : x \mapsto q(x) \) with \( q : \mathbb{R}^n \to \mathbb{R} \) and initial values \( x^{(0)} = a \) and \( \sigma^{(0)} = 1 \). With a sensible step size adaptation the following should be achieved.
- The strategy shows exactly the same behavior on \( f : x \mapsto q(10x) \) with initial values \( x^{(0)} = 0.1a \) and \( \sigma^{(0)} = 0.1 \). This means (in general) invariance against a scalar multiplication of the object parameter vector and will be discussed below.
- Assume \( \sigma^{(0)} = 10 \) to be optimal. Then, with initial value \( \sigma^{(0)} = 1 \), the step size increases within the next generations to become (nearly) optimal. This is a quite obvious objective of step size adaptation.

Generalizing this perspective, two basic demands on the adaptation of any parameters of the mutation distribution can be stated.

1. Introducing (new) invariances.
2. Performing, after an adaptation phase, comparable to the best fixed parameter setting in this situation on (all) relevant objective functions \([3,15]\).

Together with a sufficiently fast adaptation rate these properties are of major importance for a sensible adaptation mechanism.

The invariance which can be achieved by the adaptation of certain strategy parameters is usually easy to identify (see below). The fundamental starting hypothesis of this paper is, that this invariance is a necessary condition for a successful and reliable adaptation. Starting from the best strategy parameter setting, with adaptation turned on, this invariance ensures a predictable behavior of the strategy (compare the example above assuming \( \sigma^{(0)} \) to be optimal). Far from being a sufficient demand, this is probably the strongest non-empirical result one can hope for when evaluating an adaptation procedure. The relevance of this hypothesis can be reviewed in Fig. 2 (Sect. 4.2).

Concerning the parameters of the normal mutation distribution, which are dynamically controlled, in the most general case the complete covariance matrix is adapted. It is not too difficult to prove, that the choice of a certain covariance matrix can be identified with the choice of a linear transformation of the object parameter vector and vice versa [3]: Assume two linear transformations, \( T_A \) and \( T_B \), and \( y = T_A x_A = T_B x_B \). The “genotypes” \( x_A \) and \( x_B \) can be interpreted as different codings for the same “phenotype”, \( y \), which is used to evaluate the fitness. The effect of different codes becomes evident, when mutation is applied:

\[
y_{\text{new}} = T_B (x_B + \mathcal{N}(0, I)) \overset{T_B \text{ linear}}{=} T_B x_B + T_B \mathcal{N}(0, I) \\
T_A \overset{T_A \text{ bijective}}{=} T_A x_A + T_A T_A^{-1} T_B \mathcal{N}(0, I) \\
T_A \overset{T_A \text{ linear}}{=} T_A (x_A + T_A^{-1} T_B \mathcal{N}(0, I))
\]

Using a new coding, \( T_A x_A \) instead of \( T_B x_B \), is equivalent with introducing a certain linear transformation, \( T_A^{-1} T_B \), for \( \mathcal{N}(0, I) \), which is equivalent with choosing a different covariance matrix, \( T_A^{-1} T_B (T_A^{-1} T_B)^k \) instead of \( I \), for the mutation distribution.

While the choice of the initial strategy parameters (the initial covariance matrix) clearly introduces a dependency on the given coding, this dependency may vanish for \( g \to \infty \), if the algorithm adapts the covariance matrix and is
invariant against linear transformations of the object parameters (and strategy parameters accordingly). Therefore the adaptation of the complete covariance matrix should strive for becoming independent from any linear transformation of the object parameters.

The importance of the interaction between problem encoding (e.g. linear transformation of the object parameters) and operators (e.g. the mutation operator) and invariance against certain transformations is in general well recognized [8,11,14,15]. Why invariance is important from a practical point of view is also discussed in the next section.

The objective of this paper is to identify invariance properties of the ES and investigate correlated mutations, as typically applied, with respect to these invariance properties.

2 Invariance

Invariance properties of a search algorithm with respect to changes of the objective function are extremely attractive. They raise the probability to get similar (or even identical) results on similar, not yet evaluated objective functions and increase the predictability of the strategy behavior. Any evaluation of search algorithms is based on the implicit assumption of getting similar results on not yet evaluated functions.

A simple (1, λ)- or (1+λ)-evolution strategy has the following invariance properties, where \( q : \mathbb{R}^n \to \mathbb{R} \) is chosen arbitrarily [3]:

**Translation:** On \( f : x \mapsto q(x - a) \) invariance against \( a \in \mathbb{R}^n \), if the initial object parameter \( x^{(0)} \) is chosen properly.

**Rotation:** On \( f : x \mapsto q(Ux) \) invariance against the orthogonal matrix \( U \), if the initial object parameter \( x^{(0)} \) is chosen properly. This means invariance against the chosen (orthonormal) coordinate system and includes invariance against rotation and reflection of the search space.

**Order-preserving transformation:** On \( f : x \mapsto g(q(x)) \) invariance against the monotonically increasing, i.e. order-preserving, function \( g : \mathbb{R} \to \mathbb{R} \).

All three invariances can be achieved simultaneously: On \( f : x \mapsto g(q(Ux - a)) \) the simple ES is invariant against \( g, U \), and \( a \) as described above. Invariance can be lost if more complex operators are introduces in the ES. Two common cases immediately come in mind where invariance against rotation is lost, because the introduced operator depends on the given coordinate axes:

- Individual step sizes.
- Discrete recombination on object parameters (parent number \( \mu \) greater than one).

In general, the advantage of introducing such operators must be weighted carefully against the disadvantage of losing certain invariance properties.

Strategy parameter control can yield additional invariances. When individual step sizes are introduced, on \( f : x \mapsto q(Dx) \) invariance against the full rank
diagonal matrix $D$ can be achieved (in exchange with rotation invariance). In other words invariance against a scaling with respect to the given coordinate system.

It is worth noting that the formulation of the adaptation mechanism plays an important role. Only a suitable formulation can achieve new invariances. Referring to (1), consider $f : x \mapsto q(x)$ with initial values $\sigma(0) = 1$ and $x(0) = a$. If step size control is applied and $\sigma$ is varied by adding/subtracting a constant value, say 0.05, the strategy cannot perform identical on $f : x \mapsto q(10x)$ with $\sigma(0) = 0.1$ and $x(0) = 0.1a$, because $q(x^{(1)})$, where $x^{(1)} = a + (1+0.05)N(0, I)$, is different from $q(10x^{(1)})$, where $x^{(1)} = 0.1a + (0.1+0.05)N(0, I)$. In the latter the change of $\sigma$ appears to be much larger. If $\sigma$ is varied by multiplying/dividing by a constant value, say 1.05, the strategy can perform identical on $f : x \mapsto q(10x)$. In this case $q(x^{(1)})$, where $x^{(1)} = a + (1 \cdot 1.05)N(0, I)$, is identical with $q(10x'^{(1)})$, where $x'^{(1)} = 0.1a + (0.1 \cdot 1.05)N(0, I)$.

In contrast to an individual step size adaptation, in certain cases new invariances can be introduced without giving up existing ones. The following two additional invariances can be achieved by strategy parameter control, if initial object and strategy parameters are chosen accordingly:

Scalar multiplication: On $f : x \mapsto q(c \cdot x)$ invariance against the scalar $c \neq 0$, if the initial step size is chosen properly. The typical mutative step size control in evolution strategies achieves the invariance against scalar multiplication.

Linear transformation: On $f : x \mapsto q(A \cdot x)$ invariance against any full rank $n \times n$-matrix $A$, if the initial covariance matrix of the mutation distribution is chosen properly. When the complete covariance matrix of the mutation distribution is adapted, invariance against linear transformation can be achieved.

The question arises, whether the typically applied mutative adaptation scheme for the complete covariance matrix [12, 13], usually referred to as correlated mutations, actually achieves invariance against linear transformation (and retains invariances of a simple ES). On the one side this scheme, as formalized below, is invariant against translation, order-preserving transformation and scalar multiplication. On the other side former investigations indicate that the scheme has lost invariance against rotation [6, 7]. Nevertheless it is still unclear, whether invariance against linear transformation is achieved if object and strategy parameters are transformed accordingly. In Sect. 4 the invariance properties against rotation and linear transformation are investigated in detail.

### 3 Correlated Mutations

Correlated mutations were proposed in [12] and have become a standard algorithm [1, 2] which seems especially promising for difficult problem instances. This algorithm is denoted with CORR-ES. For this paper a $(15/2\mathbb{I}, 100)$-CORR-ES
is implemented, which denotes an evolution strategy with 15 parents, 100 offspring and with intermediate two parent recombination. In the CORR-ES the normal distribution with zero mean is parameterized through \( n \) step sizes, \( \sigma_i \), and \( n(n - 1)/2 \) inclination angles, \( \alpha_j \). At generation \( g \) for each offspring two parents are selected. The component wise arithmetic mean of step sizes, angles and object parameter vector of the two parents \( \sigma_i^{(g)}, \alpha_j^{(g)} \) and \( \mathbf{x}^{(g)} \) are starting-points for the mutation. For step sizes and angles the mutation reads component wise

\[
\begin{align*}
\sigma_i^{(g+1)} &= \sigma_i^{(g)} \cdot \exp \left( \mathcal{N} \left( 0, \frac{1}{2n} \right) + \mathcal{N}_i \left( 0, \frac{1}{2\sqrt{n}} \right) \right) \\
\alpha_j^{(g+1)} &= \left( \alpha_j^{(g)} \right) + \mathcal{N}_j \left( 0, \left( \frac{5}{180} \pi \right)^2 + \pi \right) \mod 2\pi - \pi
\end{align*}
\]

The random number \( \mathcal{N}(0,1/(2n)) \) in (2), denoting a normal distribution with zero mean and variance \( 1/(2n) \), is only drawn once for all \( i = 1, \ldots, n \). The modulo operation ensures the angles to be in the interval \( -\pi \leq \alpha_j^{(g+1)} < \pi \), which is, to my experience, only of minor relevance. The object parameter mutation reads

\[
\mathbf{x}^{(g+1)} = \mathbf{x}^{(g)} + R \left( \alpha_1^{(g+1)}, \ldots, \alpha_{n(n-1)/2}^{(g+1)} \right) \begin{pmatrix}
\sigma_1^{(g+1)} \\
\vdots \\
\sigma_{n}^{(g+1)}
\end{pmatrix} \cdot \mathcal{N}(0, I)
\]

The \((0, I)\)-normally distributed random vector is transformed with a diagonal matrix determined by \( \sigma_1^{(g+1)}, \ldots, \sigma_n^{(g+1)} \). The result is rotated in all two-dimensional subspaces spanned by canonical unit vectors, denoted with \( R(\cdot) \). The resulting distribution is a normal distribution and any normal distribution with zero mean can be generated this way \[10\]. Therefore the algorithm implements an adaptation of the complete covariance matrix. Note, that for a replicable definition of the algorithm the order of the chosen subspaces has to be defined as well. (The order can have a considerable impact on the performance result as can be concluded from the results shown below.) In this paper the coordinate numbers are chosen in the ordering \((1,2), (1,3), \ldots, (1,n), (2,3), (2,4), \ldots, (2,n), \ldots (n-1,n)\). Initial values are chosen \( \mathbf{x}^{(0)} = (1, \ldots, 1)^t \), \( \sigma_i^{(0)} = 1 \) for \( i = 1, \ldots, n \) and \( \alpha_j^{(0)} = 0 \) for \( j = 1, \ldots, n(n-1)/2 \), if not stated otherwise.

4 Invariance and Correlated Mutations

The rotation procedure of CORR-ES operates with respect to the given coordinate system. This leads to the assumption that the algorithm is not independent

\[1\] Rotation invariance and invariance against linear transformation can only be achieved, if discrete recombination on object parameters is avoided.
FOR $i = 1 \text{ TO } n$
1. Draw components of $u_i$ independently $(0, 1)$-normally distributed
2. $u_i := u_i - \sum_{j=1}^{i-1} \langle u_i, u_j \rangle u_j$ ($\langle \cdot, \cdot \rangle$ denotes the canonical scalar product)
3. $u_i := u_i / \|u_i\|

ROF

Fig. 1. Algorithm to generate a random orthogonal matrix $U = [u_1, \ldots, u_n]$ [6].

of the given coordinate system and therefore not invariant against rotation and linear transformation. Therefore, in this section, invariance is evaluated quantitatively by experiments on the following test functions.

4.1 Test Functions

All used objective functions are linear transformations of $f_{\text{sphere}}$, that is, they can be written in the form $f : x \mapsto f_{\text{sphere}}(A \cdot x)$, where $A$ is a certain full rank $n \times n$-matrix. Table 1 gives the used functions.

Table 1. Test Functions to be minimized, $k = 1, \ldots, n$

<table>
<thead>
<tr>
<th>Function</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{sphere}}(x)$</td>
<td>$\sum_{i=1}^{n} x_i^2$</td>
</tr>
<tr>
<td>$f_{\text{cigar}}(k)(x)$</td>
<td>$\sum_{i=1}^{n} (a_i x_i)^2$, where $a_i = \begin{cases} 1 &amp; \text{if } i = k \ 100 &amp; \text{otherwise} \end{cases}$</td>
</tr>
<tr>
<td>$f_{\text{tablet}}(k)(x)$</td>
<td>$\sum_{i=1}^{n} (a_i x_i)^2$, where $a_i = \begin{cases} 100 &amp; \text{if } i = k \ 1 &amp; \text{otherwise} \end{cases}$</td>
</tr>
<tr>
<td>$f_{\text{ellip}}(k)(x)$</td>
<td>$\sum_{i=1}^{n} (a_i x_i)^2$, where $a_i = \begin{cases} 100^{\frac{1}{k-1}} &amp; \text{if } i &lt; k \ 1 &amp; \text{if } i = k \ 100^{\frac{1}{k-1}} &amp; \text{if } i &gt; k \end{cases}$</td>
</tr>
</tbody>
</table>

For $k = 1, \ldots, n$ the differences between the functions, e.g., $f_{\text{cigar}}(k)$ are due to permutations of the coordinate axes. These permutations can be interpreted as orthogonal linear transformations of the object parameter space, which leave $\sigma_{i=1,\ldots,n} = 1$ and $x^{(0)} = (1, \ldots, 1)^t$, as given in Sect. 3, unchanged. For $n = 5$ the coefficients $a_i$ of $f_{\text{ellip}}(k)$, where $k = 1, \ldots, 5$, are permutations of the set \{1, 100^\frac{1}{4}, 100^\frac{1}{3}, 100^\frac{1}{2}, 100\}.

Additionally $f_{\text{cigar}}(U)$, and correspondingly $f_{\text{tablet}}(U)$ and $f_{\text{ellip}}(U)$, is defined as

$$f_{\text{cigar}}(U)(x) := f_{\text{cigar}}(1)(Ux)$$

with $x^{(0)} = U^{-1}(1, \ldots, 1)^t$, where $U$ is a random orthogonal (i.e., unitary) matrix chosen anew for each run (see Fig. 1). This implements an arbitrary rotation (and reflection) of the search space and basically leaves the topology of the function unchanged. Apart from $f_{\text{cigar}}(U)$, $f_{\text{tablet}}(U)$ and $f_{\text{ellip}}(U)$ all functions are separable and can be solved without correlated mutations.
Fig. 2. Testing invariance against linear transformation. Shown are the mean number of function evaluations (abscissa) to reach a certain function value (ordinate) from 20 runs. The numbers in the right figure correspond to the numbers in Table 2. A strategy which is invariant against linear transformations must show identical graphs in each figure, because the initial values for object and strategy parameters are transformed according to $f_{\text{sphere}}$ in the left and according to $f_{\text{tablet}}$ in the right. Left and right error bars depict $\sqrt{\text{mean}_{x_i < x} (x_i - \overline{x})^2}$ and $\sqrt{\text{mean}_{x_i > x} (x_i - \overline{x})^2}$.

### Table 2. Functions and initial values for Fig. 2, right

<table>
<thead>
<tr>
<th>1: $f_{\text{tablet}(1)}$</th>
<th>$\sigma_i^{(0)}_{i=1,...,5} = (1,1,1,1,1)$</th>
<th>$x^{(0)} = (1,1,1,1,1)^t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2: $f_{\text{tablet}(5)}$</td>
<td>$\sigma_i^{(0)}_{i=1,...,5} = (1,1,1,1,1)$</td>
<td>$x^{(0)} = (1,1,1,1,1)^t$</td>
</tr>
<tr>
<td>3: $f_{\text{sphere}}$</td>
<td>$\sigma_i^{(0)}_{i=1,...,5} = (100,1,1,1,1)$</td>
<td>$x^{(0)} = (100,1,1,1,1)^t$</td>
</tr>
<tr>
<td>4: $f_{\text{sphere}}$</td>
<td>$\sigma_i^{(0)}_{i=1,...,5} = (1,1,1,1,100)$</td>
<td>$x^{(0)} = (1,1,1,1,100)^t$</td>
</tr>
</tbody>
</table>

All simulations are carried out with problem dimension $n = 5$. Larger dimensions yield more pronounced effects but have a too large CPU time consumption to yield all needed results in the given time. Because invariance is investigated, there seems no particular need to do simulations in a wide range of problem dimensions.

### 4.2 Simulations

All simulations were carried out with the $(15/21, 100)$-CORR-ES and $n = 5$. In Fig. 2, left, simulations on $f_{\text{sphere}}$ are shown, compared to simulations on $f_{\text{tablet}}$ and $f_{\text{eili}}$, where object and strategy parameters are linearly transformed accordingly to $f_{\text{sphere}}$. For $f(x) = \sum_i (a_i x_i)^2$ this means $x^{(0)} = (a_{i1}, \ldots, a_{in})^t$ and $\sigma_i^{(0)} = a_i^{-1}$ for $i = 1, \ldots, n$ (and $\alpha_j^{(0)} = 0$ for $j = 1, \ldots, n(n - 1)/2$). This initial mutation distribution is optimal with respect to the objective function topology. It transforms the function into the sphere model and, if held constant, yields progress rates like on $f_{\text{sphere}}$. Even though the initial distribution is chosen optimal, performance on $f_{\text{tablet}}$ and $f_{\text{eili}}$ is worse by a factor of six to nine compared to $f_{\text{sphere}}$. The CORR-ES is not able to keep the deviation from the
optimal initial distribution sufficiently small and is not invariant against linear transformations.

In the right of Fig. 2 simulations on \( f_{\text{tablet}(1)} \) are shown together with simulations on \( f_{\text{tablet}(5)} \) and \( f_{\text{sphere}} \), where \( x_i^{(0)} \) and \( \sigma_i^{(0)} \) are transformed accordingly to \( f_{\text{tablet}(1)} \) or \( f_{\text{tablet}(1)} \), as shown in Table 2. Even though the initial distribution is wrong, performance on \( f_{\text{sphere}} \) is only slightly affected and five times better than on \( f_{\text{tablet}(1)} \). The CORR-ES clearly “remembers” the given coordinate system with the given scaling, which is quite advantageous (only) on \( f_{\text{sphere}} \).

Invariance against rotation (precisely against orthogonal transformation) is exploited in Fig. 3. For example, the difference between \( f_{\text{cigar}(1)} \) and \( f_{\text{cigar}(3)} \) is an exchange of the coordinate axes one and three, which is a special orthogonal transformation. Simulation results on \( f_{\text{ell}} \) are similar to those on \( f_{\text{tablet}} \) and omitted due to limited space. In the CORR-ES the rotation procedure is the only part which can interact with an exchange of coordinate axes in the objec-
tive function (assuming in particular \( \sigma_1^{(0)} = \ldots = \sigma_n^{(0)} \), see Sect. 4.1). Therefore, exchanging coordinate axes is equivalent with exchanging the corresponding coordinate numbers in the ordering of subspaces used in the rotation procedure.

In the left of Fig. 3 initial angles are zero. Only on \( f_{\text{cigar}(3)} \) the exchange of coordinate axes has a remarkable impact on the performance. Compared to the axis-parallel versions, \( f_{\text{tablet}(1;3;5)} \), on the arbitrarily oriented version, \( f_{\text{tablet}U} \), the performance is slightly worse with considerably larger variance. Recall that each of these runs is performed on a different function because the orthonormal basis \( U \) is chosen anew for each run.

If the initial angles are chosen uniformly randomly in \([-\pi, \pi]\) alike for all initial parents, the coordinate system dependency becomes more pronounced (right column in Fig. 3). On both functions an exchange of coordinate axes in the problem formulation has a remarkable impact on the performance (up to a factor of seven here). In particular the performance for \( k = 1 \) is remarkably improved. Recall that these objective functions are completely separable. If the coordinate system is chosen arbitrarily (like in \( f_{\text{cigar}U} \)), performance is worst.

The original intention in the CORR-ES to get, after a transition phase, progress rates like on \( f_{\text{sphere}} \) \([13, \text{p. 243}]\) cannot be met in any of these simulations. The observed progress rates are worse by a factor between 2.5 \( f_{\text{tablet}(1)} \), Fig. 3 right) and 70 \( f_{\text{cigar}(1)} \), Fig. 3 left) compared to \( f_{\text{sphere}} \).

Interpreting previous results \([6,7]\) and further simulations not shown here, the author suspects the performance differences to become (much) larger when \( n \) is increased and/or the ratio between \( \mu \) and \( n^2 \) becomes considerably smaller than \( 15/5^2 \) (which equals \( \mu/n^2 \) in the shown results).

5 Conclusion

Invariance is an attractive, conceptual objective of strategy parameter control. It is a prerequisite for a predictability of the adaptation success and enhances the transferability of performance results to real world search problems. With respect to this objective the CORR-ES (i.e. correlated mutations as typically applied \([12,13]\), see Sect. 3) reveals considerable deficiencies and cannot satisfy the original intention. In contrast, the so-called covariance matrix adaptation \([4,5]\) is invariant against any linear transformation of the search space. The disadvantage, associated with this invariance, is the possibility that the search can in principle degenerate into a subspace. The impossibility of a static degeneration in the CORR-ES must be regarded as its major advantage. It can become relevant on highly disturbed and separable objective functions.

The replicability of the results with the CORR-ES can strongly depend on the initial strategy parameters (angles), implementational details usually not explicitly defined (order of rotations) and subtle changes in the formulation of the objective function. Therefore the interpretation of results from this strategy variant must be done very carefully.
Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft under grant Re 215/12-1. The author would like to thank Andreas Ostermeier and Iván Santibáñez-Koref for many essential suggestions.

References

Theoretical Analysis of Simplex Crossover for Real-Coded Genetic Algorithms

Takahide Higuchi\textsuperscript{1}, Shigeyoshi Tsutsui\textsuperscript{2}, and Masayuki Yamamura\textsuperscript{1}

\textsuperscript{1} Graduate School of Interdisciplinary Science and Engineering, Tokyo Institute of Technology
4259 Nagatsuta, Midori-ku, Yokohama 226-8502, JAPAN
\{higuchi,my\}@es.dis.titech.ac.jp

\textsuperscript{2} Department of Management and Information Science, Hannan University
5-4-33 Amamihigashi, Matsubara, Osaka 580, JAPAN
\tsutsui@hannan-u.ac.jp

Abstract. In this paper, we perform theoretical analysis and experiments on the Simplex Crossover (SPX), which we have proposed. Real-coded GAs are expected to be a powerful function optimization technique for real-world applications where it is often hard to formulate the objective function. However, we believe there are two problems which will make such applications difficult: 1) performance of real-coded GAs depends on the coordinate system used to express the objective function, and 2) it costs much labor to adjust parameters so that the GAs always find an optimum point efficiently. The result of our theoretical analysis and experiments shows that a performance of SPX is independent of linear coordinate transformation and that SPX always optimizes various test function efficiently when theoretical value for expansion rate, which is a parameter of SPX, is applied.

1 Introduction

Evolutionary Algorithms on real number representation are expected to be a powerful function optimization technique for real-world applications\cite{1, 2, 6, 4}. We focus on EAs with recombination operators (i.e. GAs with crossover). Since it is known that performance of GA much depends on a design of the crossover operator, various crossover operators have been proposed\cite{2, 6, 4} etc.). They show good optimizing performance on various test functions which are difficult to be optimized with GAs which does not employ real numbers as its coding scheme.

Performance of real-coded GAs depends on coordinate system used to express the objective function. \cite{8, 6} and \cite{4} reports that linear coordinate transformation, such as rotation of coordinate axes and scaling(normalization) of objective variables, leads much performance down on real-coded GAs. In real-world applications, the objective functions are always formulated in arbitrary form without regard to how GAs will work on. We sometimes don't know the coordinate system used to express objective function since only objective variables are given. In
such cases, it is difficult to find a certain coordinate transformation which transforms the objective function to something like the normal form that is known to be optimized efficiently. From these reasons, performance of GAs should be independent of coordinate transformation if possible.

In Section 2 we discuss why real-coded GAs have such dependency on coordinate system, and then we derive a necessary and sufficient condition for making a performance of Real-coded GAs independent of linear coordinate transformation. Simplex Crossover (SPX) [10] is shown to satisfy this condition. Furthermore performance of SPX and its independence from linear coordinate transformation is evaluated with various test functions in Section 5.

Another problem of GAs is difficulty in adjusting parameters so that they optimize various objective functions effectively. This adjustment costs too much since it has been done in a trial-and-error manner. In order to simplify this adjustment, some useful guideline for parameter adjustment is proposed. In section 4, We derive a formula which gives a theoretical value for expansion rate, a parameter of SPX, based on recent theoretical analysis [3], [5]. Validity of the theoretical expansion rate is verified with an experiment in Section 5.

2 Analysis on GA's dependency on coordinate system

Kita and Yamamura have proposed a functional specialization hypothesis for designing GAs [5]. According to the hypothesis, selection operators and crossover operators have different functionalities each other as follows:

- Crossover operators yield offsprings so that the distribution of them will be as close as one of current population possible.
- Selection operators control distribution of population so that it reaches optimum point.

Furthermore, Kita et al. analyzed statistical properties of Unimodal Normal Distribution Crossover (UNDX) [3] and they showed that UNDX optimizes various function effectively when statistics such as a mean value and a covariance of population is preserved under certain parameter setting of UNDX. This is one of instances where the function specialization hypothesis can hold.

We believe that these results are very useful since they can be powerful guideline for designing of real-coded GAs. We consider real-coded GAs from a viewpoint of the functional specialization hypothesis as follows:

- Selection operators are driven only by random number generator and fitness distribution of population. It is clear that certain sets of individuals which share the same region of certain fitness landscape have the same fitness distribution even if these sets belong to different coordinate system. Hence it is also clear that selection operators are independent of coordinate system.
- The function optimization with a real-coded GA can be explained as a process to find an optimal point of a certain fitness landscape. Certain coordinate system for the landscape is given by formulation of the landscape, or
implicitly given when only objective variables are known. Though, the coordinate system can be manipulated with some coordinate transformation, such as normalization of objective variables for example. Hence independence from coordinate system is equivalent to independence from coordinate transformation.

Theorem 1 follows these discussion.

**Theorem 1**

Suppose that the initial population is uniformly distributed over the whole fitness landscape, and that operation of the generation alternation model is independent of coordinate system. The necessary and sufficient condition for performance of real-coded GA to be independent of linear coordinate transformation is that $X \cdot A_k$ is commuted to $A_k \cdot X$, where $X$ is a crossover operator and $A_k$ is a linear coordinate transformation.

**Proof.** Let $R$ be a random numbers. Let $g_{kn}, p_{kn},$ and $c_{kn}$ be population, parents, and offsprings in $n$th generation respectively. Let $S_R$ and $X_R$ be a selection of parents and crossover operator using $R$ respectively. We suppose that there is population $g_n$ on a certain fitness landscape $L$ and $f(i_n)$ gives a fitness value of individual $i_n \in g_n$. Let $A_k$, where $k$ is an arbitrary integer, be linear coordinate transformations. $g_{kn}$ is given by $g_{kn} = A_k \cdot g_n$. A fitness value of the individual on transformed coordinate system $f_k(i_{kn})$ is given by $f_k(i_{kn}) = f(A_k^{-1} \cdot i_{kn})$, where $i_{kn} \in g_{kn}$. Let $N_{R,f_k}(x)$ be a selection operator that selects some individuals from a set of individuals $x$.

Using them, a sequence of operations of during $n$th generation can be written as follows

$$
\begin{align*}
    p_{kn} &= S_R \cdot g_{kn} \\
    c_{kn} &= X_R \cdot p_{kn} \\
    g_{k(n+1)} &= g_{kn} - p_{kn} + N_{R,f_k}(p_{kn} + c_{kn})
\end{align*}
$$

(1)

Here we assume that $A_k \cdot X$ is commuted to $X \cdot A_k$. Under this assumption, following equations hold.

$$
    c_{kn} = X_R \cdot A_k \cdot S_R \cdot g_n = A_k \cdot X_R \cdot S_R \cdot g_n
$$

$$
    N_{R,f_k}(p_{kn} + c_{kn}) = N_{R,f_k}(A_k \cdot S_R \cdot g_n + A_k \cdot (X_R \cdot S_R \cdot g_n))
    = A_k \cdot N_{R,f}(S_R \cdot g_n + X_R \cdot S_R \cdot g_n)
$$

These equation leads

$$
    A_k^{-1} \cdot g_{k(n+1)} = g_{(n+1)}
$$

(2)

Equations in the above are illustrated in figure 1. (2) means that population of $(n + 1)$th generation which is derived from population $g_n$ are on the same region of certain landscape $L$ as long as $A_k \cdot X$ is commuted to $X \cdot A_k$, where $A_k$ is an arbitrary linear transformation. Since both $g_0$ and $g_{k0}$ are same distribution (i.e. uniform distribution over the whole fitness landscape) as long as $A_k$ is linear
transformation, it is obvious by induction that all \( g_k(n+1) \) and \( g_k \) are on the same region of \( L \), and hence obviously we always get same fitness distribution from \( g_k(n+1) \) and \( g_{(n+1)} \) as long as \( A_k \) is linear transformation.

We have proved a necessary condition in the above. Proof of a sufficient condition is almost clear. □

3 Simplex Crossover

This section describes Simplex Crossover\(^1\) (SPX, [10]) and investigate its statistical properties. It is almost clear that SPX satisfies theorem 1 by following definition.

3.1 Definition

Suppose that search space is \( R^n \). Individuals are represented as \( n \)-dimensional continuous vectors. The procedure of SPX is described as follows:

1. Choose \((n+1)\) parent vectors \(P_0 \cdots P_n\) by some selection schema. We choose randomly in the following.
2. Determine the center of gravity of parent vectors \(G = \sum_{i=0}^{n} P_i\)
3. Generate \(r_k\), which is a random number given by following equation;

\[
r_k = (u(0,1))^{\frac{1}{k+1}} \quad (k = 0, \cdots, n-1)
\]

where \(u(0,1)\) is uniformly distributed random number between 0 and 1. We also define following nominal variables in order to simplify description of our investigation on statistical property of SPX.

\[
r_k = \begin{cases} 
0 & (k < 0) \\
1 & (k \geq n) 
\end{cases}
\]

\(^1\) There is another crossover which is also named Simplex Crossover[7]. This is completely different from our idea.
4. Calculate $x_k$ and $C_k$ by

$$x_k = G + \varepsilon (P_k - G) \quad (k = 0, \cdots, n)$$

(5)

$$C_k = \begin{cases} 0 & (k = 0) \\ r_k^{-1}(x_{k-1} - x_k + C_{k-1}) & (k = 1, \cdots, n) \end{cases}$$

(6)

where $\varepsilon$ is the expansion rate, a parameter of SPX.

5. An offspring $C$ is given by

$$C = x_n + C_n$$

(7)

4 Statistical Properties of SPX

As we discussed in Section 2, we focus on the shift of distribution of individuals based on the functional specialization hypothesis. In this section we analyze statistical properties of SPX based on [3] in order to give useful design guidelines for parameter of SPX.

Suppose that the number of parents is large enough so that they can be handled as a probability distribution. We denote statistical properties of individuals as follows:

- A mean value of distribution of parents is given by $\langle x \rangle = \bar{x}$.
- An element of covariance matrix of parents can be written as $\gamma_{ij} = \langle (x_i - \bar{x}_i)(x_j - \bar{x}_j) \rangle$

These leads theorem 2 and 3.

Theorem 2

The mean vector of parents and the mean vector of offsprings generated with SPX are the same. i.e $\langle C \rangle = \langle P \rangle$.

Proof. $\langle P_k \rangle = \langle P \rangle$ since parents are many enough. This and (5) lead $\langle x_{k-1} - x_k \rangle = 0$. Using this and (6), we easily obtain $\langle C_k \rangle = \langle r_{k-1} \rangle \cdot \langle C_{k-1} \rangle = 0$ by induction. Using this and (7), a mean vector of offsprings $\langle C \rangle$ is given by $\langle C \rangle = \langle C_n + P_n \rangle = \langle P \rangle$. □

Theorem 3

Let $(m + 1)$ is a number of parents. A covariance matrix of distribution of offsprings that generated with SPX are given by

$$\{\gamma^C_{ij}\} = \frac{1}{m + 1} \left( 1 + \varepsilon^2 \frac{m}{m + 2} \right) \{\gamma^P_{ij}\}$$

(8)

where $\{\gamma^P_{ij}\}$ is a covariance matrix of parents.

Proof is described in Appendix A.
According to Theorem 2, SPX always preserves a mean vector of population. According to theorem 3, SPX preserves a covariance matrix of population when expansion rate of SPX (\( \varepsilon \)) is

\[
\varepsilon = \begin{cases} 
\sqrt{m+2} & (m = 1, 2, \ldots) \\
\text{any} & (m = 0)
\end{cases}
\]

where \((m + 1)\) is a number of parents. We propose to using this value as a theoretical expansion rate of SPX. For example, the theoretical expansion rate for SPX that needs 21 parents is \(\sqrt{22} = 4.690 \cdots\). The validity of the theoretical expansion rate is verified in next section.

5 Experiments

In this section, we make experiments with various test functions in order to show that SPX always optimizes various function effectively when the theoretical expansion rate is applied. We also show that an optimizing performance of SPX is not affected by linear coordinate transformation through comparing performance of SPX and UNDX[6]. UNDX efficiently optimizes various unimodal and multimodal functions with strong epistasis though, it does not satisfies theorem 1. This is the reason why we compare them.

5.1 Design of Experiments

Here we compose a genetic algorithm used to following experiments. we employed the minimal generation gap model (MGG)[9] with slight modification. Since original MGG model assume that the number of parents is always two, we modified it so that it will work with crossover operators which needs more than three parents. The procedure of GA is described as follows:

1. Generate an initial population randomly and uniformly
   Initial population is uniformly distributed in a domain for which the objective function is defined.
2. Choose \(n+1\) individuals randomly for SPX, three individuals for UNDX as parents
3. Choose two individuals randomly from parents
4. Generate offsprings with the crossover.
5. Alternate generation:
   Parents chosen at procedure 3 are replaced with a best offspring and another one chosen by roulette-wheel selection.
6. Repeat procedures 2 through 5 until a certain condition is fulfilled.

We did not used the mutation since we focus on the difference of characteristics between crossover operators.

Table 1 summarizes test functions we used. \(F_3\) and \(F_5\) is generated from \(F_2\) and \(F_4\) respectively with linear coordinate transformation. Performance of GAs which satisfy theorem 1 should be independent of such linear coordinate transformation.
sphere-d function
\[ F_{1d}(x) = \sum_{i=1}^{n} (x_i - d)^2 \quad (-5.12 < x_i < 5.12) \]

Rastrigin-d function
\[ F_{2d}(x) = 10n + \sum_{i=1}^{n} \{ (x_i - d)^2 - 10\cos(2\pi(x_i - d)) \} \quad (-5.12 < x_i < 5.12) \]

This function is multimodal. There is no epistasis among parameters.

Rotated-Rastrigin-d function
\[ F_{3d}(x) = F_{2d}(Mx), \text{ where } M \text{ is a linear matrix that performs random rotation of coordinate system around the origin. The rotation yields epistasis among parameters.} \]

Rosenbrock function
\[ F_{4}(x) = \sum_{i=2}^{n} \{ 100(x_1 - x_i^2)^2 + (x_i - 1)^2 \} \quad (-2.048 < x_i < 2.048) \]

This function is unimodal, and have strong epistasis among parameters. This function is derived by performing linear coordinate transformation on F5 so that all objective variable \( x_i \) will be normalized.

Scaled-Rosenbrock function
\[ F_{5}(x) = \sum_{i=2}^{n} \{ 100(x_1 - (ix_i)^2)^2 + (ix_i - 1)^2 \} \quad (-2.048/i < x_i < 2.048/i) \]

Table 1. Test functions

5.2 Result 1: Verifying the Expansion Rate of SPX

In this section, we applied SPX to test functions at various expansion rates around the theoretical expansion rate (\( \varepsilon_o \)).

Test functions we used in this experiment are \( F_{1,0}, F_{2,0} \) and \( F_4 \). The dimension of these test functions (\( n \)) is 10, 20 and 30. The population size is \( n \times 15 \) for unimodal functions (\( F_{1,0} \) and \( F_4 \)), \( n \times 90 \) for multimodal functions (\( F_{2,0} \)). SPX generates \( n \times 10 \) offsprings. 25 experiments are performed for each test functions, dimensions and expansion rates.

An optimum point of \( F_{1,0} \) and \( F_{2,1,0} \) is \((1.0, \cdots, 1.0)\), whereas one of \( F_{1,0} \) and \( F_{2,0} \) is \((0, \cdots, 0)\). Note that we use \( F_{1,0} \) and \( F_{2,1,0} \) instead of \( F_{1,0} \) and \( F_{2,0} \) which are often used. Such a expansion rate that will reduce covariance of population promotes a convergence to the mean of individual vectors, that is origin of coordinate system at the initial population. For such cases, we will get wrong result if the test functions have their optimum point in the origin of coordinate system.

Table 2 shows number of trials which succeed in finding the optimum (SUC), and average number of evaluation that is spent to find the optimum (AVG).

The result shows that SPX with theoretical expansion rate gives best performance on \( F_2 \) and \( F_4 \), and good performance on \( F_1 \). Smaller expansion rate causes premature convergence on \( F_2 \) and \( F_3 \), and a performance of SPX with larger expansion rate is worse, especially on multimodal function.
5.3 Result 2: An Effect of Coordinate Transformation

Here we observe performance of SPX and UNDX in order to verify that SPX is independent of linear coordinate transformation.

Test functions that we used in this experiment are 20 dimensional $F_{20}$, $F_{30}$, $F_4$, and $F_5$. Population size is 300 for unimodal functions ($F_4$ and $F_5$), 500 for multimodal functions ($F_2$ and $F_3$). The expansion rate of SPX we used is $\sqrt{22}$, which is the theoretical expansion rate under this setting. Parameters of UNDX are $\alpha = 0.5$ and $\beta = 0.35$, which are also theoretical value according to [3], and known that the optimizing performance of UNDX is good when these value are applied. Both crossover operator generates 200 offsprings. 25 trials are performed for each test functions.

Figure 2 and 3 shows progress of search. The result shows SPX is independent of both coordinate rotation and scale of coordinate axes, whereas performance of UNDX depends very much on scale of coordinate axes.

6 Conclusions

In this paper we proved a necessary and sufficient condition for making performance of real-coded GAs independent of any linear coordinate transformation and showed that SPX satisfies it. We also showed that SPX optimizes various test functions efficiently and its performance is independent of any linear coordinate transformation. We derived a formula that gives the theoretical expansion rate and showed that it gives robust optimizing performance on various test functions independently of its dimension. We believe that the functional special-
Fig. 2. Function $F_{20}$ (left) and $F_{30}$ (right)

Fig. 3. Function $F_{40}$ (left) and $F_{50}$ (right)

ization hypothesis we follow can be applicable on other EAs with recombination operators.

References


A Appendix A: Proof of Theorem 3

Let us suppose that number of parents is (m+1). (6) can be written as

\[ C = \sum_{k=0}^{m} t_k P_k \]  

with

\[ t_k = \frac{\varepsilon(r_m \cdots r_k (1 - r_{k-1})) + \frac{1}{m+1}(1 - \varepsilon)}{m+1} \]  

From (11) we obtain \( \sum_{k=0}^{m} t_k = 1 \). Using them and theorem 2, it can be written as

\[ x_i^C - x_i^C = \left( \sum_{k=0}^{m} t_k x_i^P_k \right) - x_i^P = \sum_{k=0}^{m} t_k (x_i^P_k - x_i^P) \]

where \( x_i^C \) is an element of offspring vector \( x^C \).

Since parents are randomly chosen, \( \langle (x_i^a - x_i^b)(x_j^b - x_j^b) \rangle = 0 \) is calculated when \( a \neq b \). These lead following formula:

\[ \gamma_{ij}^C = \langle (x_i^C - \overline{x_i^C})(x_j^C - \overline{x_j^C}) \rangle = \left\langle \sum_{k=0}^{m} t_k^2 (x_i^P_k - \overline{x_i^P})(x_j^P_k - \overline{x_j^P}) \right\rangle = \left\langle \sum_{k=0}^{m} t_k^2 \right\rangle \gamma_{ij}^P \]  

Average of \( r_k \) and one of \( r_k^2 \) are easily calculated as

\[ \langle r_k \rangle = \begin{cases} 0 & (k < 0) \\ \frac{k+1}{k+2} & (k = 0, \cdots, m-1) \\ 1 & (k \geq m) \end{cases} \]

\[ \langle r_k^2 \rangle = \begin{cases} 0 & (k < 0) \\ \frac{k+1}{k+2} & (k = 0, \cdots, m-1) \\ 1 & (k \geq m) \end{cases} \]

Using them , (11) and (3),

\[ \left\langle \sum_{k=0}^{m} t_k^2 \right\rangle = \frac{1}{m+1} \left( 1 + \varepsilon^2 \frac{m}{m+2} \right) \]  

is calculated. This and (12) leads (8). \( \square \)
Applying Self-Organised Criticality to Evolutionary Algorithms

Thiemo Krink, Peter Rickers and René Thomsen
EvAlife Group, Dept. of Computer Science, University of Aarhus, Denmark
{krink,rickers,thomsen}@daimi.au.dk, http://www.evalife.dk

Abstract. Complex systems are typically composed of a large number of locally interacting components that operate at a critical state between chaos and order, which is known as self-organised criticality. A common feature of this state is the exponential (power law) relationship between the frequency of an event and the size of its impact, such as the event of an earthquake and its strength on the Richter scale. Most state transitions in a component of a complex system only affect its neighbourhood, but once in a while entire avalanches of propagating state transitions can lead to a major reconfiguration of the system. In evolution, this system behaviour has been identified in species extinction on an evolutionary time-scale, where avalanches correspond to mass extinction. In this paper, we applied the concept of self-organised criticality (SOC) to control mutation on the individual level and extinction on the population level in the context of evolutionary algorithms (EA). Our results show that the SOC EAs clearly outperform standard EAs and a previously introduced mass extinction model. Interestingly, the great performance of our SOC EAs is based on a very simple modification of standard EAs and implies almost no additional computational costs.

1 Introduction

The choice of optimal operators for selection and mutation is a widely discussed topic in the field of evolutionary algorithms. Regarding selection, the common choices are the tournament selection [4] and roulette wheel selection [12]. Towards the end of a run, these models have the disadvantage that only few individuals are replaced by mutated and recombined individuals. Thus, only very few 'new ideas' based on mutated and recombined genes are introduced to the system at a point where only genetic diversity could prevent the stagnation of the optimisation process. One possibility to avoid this problem is to use so-called (mass-) extinction models, which have been introduced recently [6],[10],[11]. In mass extinction models, major parts of the population are regularly replaced compared to a gradual substitution of single individuals in standard EAs. The inspiration for these models is based on evolutionary theory and paleontological findings in fossil records, which revealed that mass extinction has been a common phenomenon in the process of evolution, such as the major extinction at the end of the Permian era (~250 Myr ago), when about 96% of all species
became extinct [14]. The motivation for this approach is that the replacement process exploits highly fit solutions and random explorations of other domains. Marín and Solé [10],[11] presented a macroevolutionary algorithm (MA), which models the dynamics of species extinction by an ecological model of species dependencies. In their approach, all individuals of a species die if the majority of the neighboured species has a better fitness. Thus mass extinction is induced when a new and very good solution is found. One drawback of the MA approach is that the model is quite complicated to implement and it requires substantial additional computation time. Furthermore, this model produces only better results when compared to a standard EA in one out of three test cases [10]. This model partly resembles aspects of Per Bak's species extinction model [1]. Here, species represented by random numbers are arranged in a circle. At each time step, the lowest number, and the numbers at its two neighbours, are each replaced by new random numbers. Greenwood et al. [6] presented a simpler, but very effective mass extinction model where the mortality of individuals is controlled by a 'stress' parameter. Here, a random value is drawn from a uniform distribution, which determines the size of the extinction. This model was verified in a comparison with a standard EA for one multi-modal test function.

Another very important aspect in evolutionary algorithms with float-encoded genes are operators. Here, Gaussian mutation is a common choice. Regarding the crucial setting of the Gaussian mutation variance, there are three choices: (i) fixed variance [8]; (ii) variable (decreasing) variance [12]; and (iii) self-adaptive variance (for a survey see [3]). Though variable mutation variance can vastly improve the optimisation compared to a fixed variance in some test cases [12], the former may fail in context of other test functions (see section 5.1). The reason for this is that the optimisation of some objective functions requires additional exploration towards the end of the run in order to prevent premature convergence on local hills. Self-adaptation can deal with this problem e.g. [5], [9]. However, two main drawbacks of self-adaptive approaches are additional computational effort and the comparably long convergence time, since parameter values are gradually tuned towards optimal values [5].

In this paper, we present an approach which tries to overcome the classical problem of premature convergence on local hills by application of self-organised criticality (SOC) - a theory based on complexity research [1]. The key idea is to control the variance of Gaussian mutation and the extinction size during selection by a power law function, which reflects typical dynamics found in complex systems such as Darwinian evolution.

2 Self-Organised Criticality

Complex system behaviour is neither linear (stable systems) nor uncorrelated (chaos), but formed by self-organisation in a long transient period and found at the border of stability and chaos - a state known as self-organised criticality [1]. Typically, in complex systems the strength of an emergent property is inversely proportional to its frequency \( f \) (also known as \( 1/f \) noise) with the functional
relationship of a power law: \( x \rightarrow x^{-\tau} \). When plotted on a log/log scale, power law data can be described by a linear fit with a negative slope. Various complex systems follow these power laws such as light emitted from a quasar [13], Zipf’s law of population sizes in cities [16], or the Gutenberg-Richter law of earthquakes [7]. A model that can sufficiently explain this behaviour is based on the paradigm of a sandpile (the so-called Sandpile Model) [1]. The main idea in SOC is that most state transitions in a component of a complex system only affect its neighbourhood, but once in a while entire avalanches of propagating state transitions can lead to a major reconfiguration of the system. Our main motivation for exploiting the theory of self-organised criticality in the context of evolutionary algorithms was that extinction rates in evolution can also be described by power laws [1],[15]. Here avalanches correspond to mass extinction.

### 3 The SOC Models

#### 3.1 The Sandpile Model

Both SOC EA models that we present in this paper require power law distributed numbers, which can easily be obtained from a Sandpile Model and stored for further use in an array. In the Sandpile Model grains of sand are continuously dropped on top of a two dimensional grid lattice. The number of grains within a square at position \((x,y)\) is described by a variable \(Z(x,y)\) and cannot exceed three grains. The pseudo code of the Sandpile Model is shown in figure 1(a). At initialisation the grid lattice is empty.

```plaintext
procedure Sandpile Model
repeat
    Select a random \(x\) and \(y\)
    \(Z(x,y) := Z(x,y) + 1\)
    if \(Z(x,y) = 4\) then
        \(Z(x \pm 1, y) := Z(x \pm 1, y) + 1\)
        \(Z(x, y \pm 1) := Z(x, y \pm 1) + 1\)
        \(Z(x,y) := Z(x,y) - 4\)
        Update all affected cells recursively
```

At each time step, a random cell in the lattice is chosen and its number of grains is increased by one. If the selected cell has exceeded the critical value
of three then each of the four neighbouring cells receives one grain, and the number of grains of the cell itself is decreased by four. If the selected cell is at the boundary of the lattice its grains of sand leave the system.

In the beginning of the process, when most cells are nearly empty, there are only few cells that can propagate grains to neighboured cells, and thus there are no large avalanches of toppling sand grains. Figure 1(b) shows a distribution of avalanches created by a Sandpile Model with a dimension of 9x9 which has been running for 1000 steps. Both models described in the next two sections used power law distributed numbers, which we created with the Sandpile Model. Note that this row of numbers was only created and stored once and not before each run of the EAs. However, the computational effort to calculate these numbers is so marginal that even a calculation during run-time would not have caused significant additional computational costs compared to the overall run-time.

3.2 The SOC EA Mutation Model

In this study we designed a new mutation operator based on a power law distribution, which is generated by the Sandpile Model described in section 3.1 with a dimension of 9x9. The first 500 numbers of the distribution are not used in order to exclude the initialisation phase. The algorithm of the mutation operator is outlined in figure 2.

procedure SOC EA mutation
begin
if mutation rate > uniform random number then
for each gene in the chromosome do
  Get variance modifier of the power law distribution
  variance := variance modifier * parameter interval size
  modifier := N(0, variance)
  Add the modifier to the gene
  Validate that the gene's value is inside the parameter range
end

Fig. 2. The structure of the SOC EA mutation model.

In the beginning, the algorithm determines whether the genes of a chromosome should be mutated. If this is the case then each gene is treated as follows: First, the variance of the Gaussian mutation operator is calculated as the product of a value from the power law distribution\(^1\) and the parameter interval size, which represents the range of the gene. Second, the gene is mutated by adding the modifier, which is a number of normal distributed random numbers with a mean of zero and the previously calculated variance. Finally, the new gene value is truncated at the bounds of the parameter range if necessary. The difference

\(^1\) Scaling the value with 1/100 turned out to be beneficial.
in computational costs between this operator and the standard Gaussian mutation operator is a simple lookup in the power law distribution array. By using this distribution most mutations will be small, some will be medium, and very few will be large. This approach allows a fine scale numerical search as well as occasional exploratory leaps away from local optima.

3.3 The SOC EA Extinction Model

Apart from the SOC EA Mutation model we propose a new selection operator, which is inspired by the notion of mass extinction on a macro-evolutionary level. However, we applied the mass extinction at the population level instead of the species level. We used the Sandpile Model to create power law distributed numbers, which set the mortality in the extinction selection. These numbers were stored in an array PLDistribution. In the experiments we used the PLDistribution shown in figure 1(b). The ten first initial values are (1,0,0,0,1,0,0,0,0,0) while the values increases after the initialisation phase i.e. the PLDistribution after 500 steps are (0,15,0,1,0,0,36,0,0). The outline of our SOC EA extinction model is shown in figure 3.

\[
\text{procedure } \text{SOC EA extinction} \\
\begin{align*}
\text{begin} \\
\text{kill percent} & := \text{PLDistribution}[\text{generation}] \\
\text{if } & \text{kill percent} \geq 1 \text{ then} \\
\text{kill rate} & := (\text{kill percent} \times \text{population size})/100 \\
\text{Choose kill rate amount of individuals randomly} \\
\text{Replace the chosen individuals with randomly} \\
\text{chosen individuals from the surviving population,} \\
\text{and apply fixed Gaussian mutation once.} \\
\text{else} \text{ apply tournament selection} \\
\text{end}
\end{align*}
\]

Fig. 3. The structure of the SOC EA extinction model.

The extinction selection works as follows: First the algorithm sets the extinction rate (kill percent) based on a value from the Sandpile Model power law distribution. In the current implementation, we used the generation number as an array index. The value of this index represents the percentage of the population that should get extinct. If this value is below one, normal tournament selection is applied. Otherwise, all individuals that should be removed are randomly selected and replaced with randomly chosen individuals from the surviving part of the population. In order to provide diversity between the individuals, the newly introduced individuals are mutated once by fixed Gaussian mutation, before they are joined with the population. Afterwards, standard crossover and variable Gaussian mutation is applied to the new population.
4 Experiments

4.1 Test Functions

In our experiments we used the following six test functions:

De Jong F2: \( f(x, y) = 100(x^2 - y)^2 + (1 - x)^2 \) where \(-2.048 \leq x \leq 2.048 \) and \(-2.048 \leq y \leq 2.048 \)

De Jong F4: \( f(x) = \sum_{i=1}^{30} x_i^4 \) where \(-1.28 \leq x_i \leq 1.28 \)

Schaffer F6: \( f(x, y) = 0.5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0.5}{(1+0.001(x^2+y^2))^2} \) where \(-100 \leq x \leq 100 \) and \(-100 \leq y \leq 100 \)

Rastrigin F1 (20 dimensions): \( f(x) = 200 + \sum_{i=1}^{20} x_i^2 - 10 \cdot \cos(2\pi x_i) \) where \(-5.12 \leq x_i \leq 5.12 \)

Michalewicz's function: \( f(x) = \sum_{i=1}^{10} \sin(x_i) \cdot \sin^2\left(\frac{i \cdot x_i}{\pi}\right) \) where \(0 \leq x_i \leq \pi\)

Griewank's function: \( f(x) = \frac{1}{4000} \cdot \sum_{i=1}^{10} (x_i - 100)^2 - \prod_{i=1}^{10} \cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1 \) where \(-600 \leq x_i \leq 600 \)

Except for the Michalewicz function, all test functions are minimisation tasks. The Michalewicz test function has its global optimum at 9.66; all other functions have their global optimum at 0.

4.2 Settings for the Algorithm & Data Sampling

Each genome was represented by floating point encoding. In all experiments, the population size was set to 400 individuals, the mutation rate was \( p_m = 0.75 \) and the crossover rate was \( p_c = 0.90 \). Both algorithms, the standard EA and the SOC EA used elitism. The standard EA used tournament selection (with a tournament size of 2) and Gaussian mutation with variable variance\(^2\) as operators. The extinction version of our SOC EA used our new selection methods and the same mutation operator. The mutation version of our SOC EA used tournament selection and our new SOC EA mutation operator. To compare our models with the standard EA we used the average best fitness of 30 runs.

5 Results

5.1 The SOC EA Mutation Model

For the comparison between the Gaussian mutation operator and the SOC EA mutation operator we made 18 experiments with the six test functions mentioned in section 4.1. For each test function we compared the SOC EA mutation operator with Gaussian operators using (i) fixed and (ii) variable variance. In

\[^2\] The variable Gaussian mutation operator uses \( \sigma^2(t) = \frac{1}{t+1} \) where \( t \) is the current generation.
both EAs we used tournament selection. Table 1 summarises the results of the various experiments. The mean column represents the mean best fitness value found after 1200 generations.

Table 1. Results obtained from the experiments: Best fitness after 1200 generations. SE = Standard Error. (* indicates a maximisation task.)

<table>
<thead>
<tr>
<th>Test function</th>
<th>Variable Gaussian</th>
<th>Fixed Gaussian</th>
<th>SOC Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Jong F2</td>
<td>0.01345±1.926E-4</td>
<td>1.197E-4±1.286E-6</td>
<td>4.167E-5±1.152E-6</td>
</tr>
<tr>
<td>De Jong F4</td>
<td>3.830E-3±1.235E-4</td>
<td>3.917±0.01227</td>
<td>2.806E-7±6.224E-9</td>
</tr>
<tr>
<td>Schaffer F6</td>
<td>6.112E-3±1.221E-4</td>
<td>4.776E-3±1.675E-4</td>
<td>6.534E-6±9.282E-7</td>
</tr>
<tr>
<td>Rastrigen F1 (20D)</td>
<td>3.332±0.1122</td>
<td>102.042±0.3243</td>
<td>2.517±9.651E-3</td>
</tr>
<tr>
<td>Michalewicz'</td>
<td>9.608±0.01551</td>
<td>7.343±2.179E-3</td>
<td>9.633±5.101E-4</td>
</tr>
<tr>
<td>Griewank (10D)</td>
<td>0.07443±9.093E-4</td>
<td>3.115±2.069E-3</td>
<td>0.09633±4.582E-4</td>
</tr>
</tbody>
</table>

In all experiments, we observed characteristic differences in the performance of the SOC EA mutation model and the standard EAs with fixed and variable variance in the Gaussian mutation. Except for the Griewank-function, the SOC EA mutation model found much better solutions after 1200 generations. Further, the performance of both Gaussian mutation operators was highly problem dependent (see table 1) in contrast to the SOC EA mutation model. Figure 4(a) and 4(b) show the results for the Michalewicz- and De Jong F4-function. Further, we found that the computation of the SOC EA mutation model took 3% more time than the standard EA (0.3 seconds difference after 30 runs on a Pentium II-466MHz PC).
5.2 The SOC EA Extinction Model

In twelve experiments we compared the standard EA with the SOC EA extinction model regarding the six test functions mentioned in 4.1. In both EAs we used Gaussian mutation with variable variance. Table 2 summarises the results of the various experiments. The mean column represents the mean fitness value found after 800 generations. In all experiments, we observed characteristic performance differences between the standard EA and the SOC EA extinction model. In the beginning of the runs, the convergence speed of the SOC EA and the standard EA was similar. However, in all experiments the performance of the standard EA stagnated after about 50-200 generations while the SOC EA continued to produce better solutions. Figure 5(a) and 5(b) illustrate these characteristics for the Michalewicz- and the Griewank-function. The computation time of the SOC EA extinction model was exactly the same as in the standard EA.

Table 2. Results obtained from the experiments: Best fitness after 800 generations. SE = Standard Error. (* indicates a maximisation task.)

<table>
<thead>
<tr>
<th>Test function</th>
<th>Std. EA Mean ± SE</th>
<th>SOC EA Mean ± SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Jong F2</td>
<td>9.238E-3±2.841E-4</td>
<td>5.128E-3±1.302E-4</td>
</tr>
<tr>
<td>De Jong F4</td>
<td>1.990E-3±6.309E-5</td>
<td>6.063E-10±2.049E-11</td>
</tr>
<tr>
<td>Rastrigen F1 (20D)</td>
<td>3.651±6.023E-2</td>
<td>3.324E-3±1.126E-3</td>
</tr>
<tr>
<td>Michalewicz*</td>
<td>9.599±1.337E-3</td>
<td>9.635±5.799E-4</td>
</tr>
<tr>
<td>Griewank (10D)</td>
<td>6.985E-2±1.205E-3</td>
<td>3.038E-3±1.029E-4</td>
</tr>
</tbody>
</table>

Fig. 5. Average best fitness (30 runs).
6 Discussion

In this paper we have introduced the SOC EA mutation model and the SOC EA extinction model, which use the concept of self-organised criticality (SOC) to control mutation on the individual level and extinction on the population level. Our results show that the SOC EA mutation was always better than fixed Gaussian mutation, and in 5 out of 6 test cases better than variable Gaussian mutation. In most test cases the SOC EA mutation converged slower to the optimum than the standard EA in the very beginning of the runs. However, after a short while the SOC EA mutation always found very exact solutions, whereas the standard EA got stuck on local optima. Further, our results show that our SOC EA extinction model found better solutions for all test functions compared to a standard EA. In contrast, the MA extinction model by Marín and Solé performs much worse compared to a standard EA for certain functions such as the Michalewicz's function [10]. Further, the SOC EA extinction model is easy to implement and computational cheap. In the initial phase of this study we investigated different variants of our SOC EA extinction model such as worst fit/best fit removal strategies, but they did not produce promising results and turned out to be computational expensive. However, one strategy slightly improved the results on the De Jong F2 test function. If random replacement of the individuals was used instead of mutation replacement by existing individuals, the mean fitness value found after 800 generations was 3.5 times better. Here, a search strategy that tries out more diverse ideas has an advantage due to the plateau of the De Jong F2 function. This idea is related to Cobbs's "random immigrants model" [2]. Our preliminary experiments with a combined SOC EA model involving both operators did not produce very promising results. However, a more thorough investigation would be necessary to draw firm conclusions.

Why did the SOC approach succeed? The advantage of the SOC EA is that it provides continuous exploration combined with a focus on exploitation, which keeps the search active and avoids the loss of genetic variance towards the end of a run. Regarding the extinction model, elitism and Gaussian mutation with variable variance turned out to be essential. Elitism prevented that the best solution was accidentally removed by a mass extinction event. The Gaussian mutation with variable variance had a similar effect, because the population tended to converge genetically during the run. Both mechanisms provided the necessary fidelity for a numerical optimisation task. Correspondingly, elitism was also an advantage concerning the SOC EA mutation.

In this investigation we focused on common benchmark problems in order to allow a comparison of our approach with evolutionary algorithms proposed in other studies. Interestingly, the SOC EA could be beneficial in conjunction with dynamic environments due to its ability to explore new solutions at all times. Unfortunately, there are no commonly accepted benchmark tests for dynamic problems yet. To our knowledge, this study is the first investigation that shows that self-organised criticality can be a used in context of Evolutionary Algorithms.
7 Acknowledgements

The authors would like to thank Brian H. Mayoh, Rasmus K. Ursem and Lars Bach of the EvAlife group for useful comments and suggestions.

References

Genetic Algorithms, Clustering, and the Breaking of Symmetry

Martin Pelikan and David E. Goldberg

Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
{pelikan, deg}@illigal.ge.uiuc.edu

Abstract. This paper introduces clustering as a tool to improve the effects of recombination and incorporate niching in evolutionary algorithms. Instead of processing the entire set of parent solutions, the set is first clustered and the solutions in each of the clusters are processed separately. This alleviates the problem of symmetry which is often a major difficulty of many evolutionary algorithms in combinatorial optimization. Furthermore, it incorporates niching into genetic algorithms and, for the first time, the probabilistic model-building genetic algorithms. The dynamics and performance of the proposed method are illustrated on example problems.

1 Introduction

Symmetry is one of the major difficulties of genetic algorithms on combinatorial problems. In spite of that the nature of the problem encourages the use of some kind of recombination, due to the symmetry of the problem the recombination often slows down the convergence by disrupting good solutions. Moreover, the use of niching in the probabilistic-model building genetic algorithms (PM-BGAs) (Pelikan, Goldberg, & Lobo, 1999) was identified as an important issue when extending the algorithms to solve difficult hierarchically decomposable problems (Pelikan, Goldberg, & Cantú-Paz, 2000).

In this paper we propose a method to both deal with the symmetry in a problem as well as incorporate niching. The method is not intended for any particular recombination operator and can be used to extend and improve the recombination in both the simple genetic algorithms as well as the probabilistic model-building genetic algorithms. Each generation the set of selected parents is clustered according to their genotypes. Each cluster is then processed separately, yielding some offspring. The number of offspring produced by each cluster can be either proportional to its size or its average fitness. By avoiding juxtaposition of the solutions from different clusters, the negative effect of symmetry in a problem is alleviated. By assigning each cluster resources proportional to its average fitness, niching is incorporated to allow the algorithms to thoroughly search the solution space for multiple optima. Even though niching is a widely studied concept in genetic algorithms, this is the first attempt to use it in the probabilistic model-building genetic algorithms.
The paper starts by a brief introduction into genetic algorithms and the problem of symmetry. Section 3 introduces the class of clustering evolutionary algorithms. The k-means clustering method that was used to cluster the parent population in our experiments is introduced in Section 4. Section 5 presents the results of our experiments. The paper is concluded in Section 6.

2 Symmetry

For many combinatorial problems there are a number of different solutions to a problem. Moreover, many regularities in the entire landscape can often be observed. In a graph bisection, for instance, the goal is to partition the nodes of a given graph into two equally sized groups so that the number of edges between the groups is minimized. Each bit in the solution string corresponds to one node in the graph and its value determines the group to which this node is assigned. It is easy to see that in this problem, there are at least two optima which are complementary. Moreover, the average fitness of any schema is equal to the average fitness of the complement of the schema which is fixed in the same positions as the original schema but to the exactly opposite values, e.g. $f(\star \star \star 00 \star \star \star ) = f(\star \star \star 11 \star \star \star )$. This implies that the fitness of each solution does not depend on the value of a particular bit or a set of bits but on the overall combination which can be often difficult to obtain. Each schema and its complement have the same fitness on average and unless the population drifts to either side, the GAs have no mechanisms to decide which way to go from a uniformly distributed population. This problem is aggravated in the algorithms which guide the search by building and using a probabilistic model of promising solutions that process schemata more explicitly.

Both the GAs as well as the PMBGAs guide the exploration of the search space to regions that can be reached by combining important parts of promising solutions found so far. However, in case of symmetric problems, this often yields to a decrease in the solution quality. In the simplest case (e.g., the graph partitioning mentioned above), there are two complementary parts of the search space that are to be explored. However, combining high-quality solutions and their complements which are equally good often results in poor solutions. Furthermore, as it was pointed out above, the algorithm has no means of deciding between complementary partial solutions since both seem to be of the same quality on average. If the niching were incorporated to eliminate genetic drift, the GAs would either converge very slowly or would never reach the optimum. This becomes a crucial problem for the PMBGAs that use only macroscopic information about the partial solutions in the population of parents to generate new offspring. The problem can be eliminated only by using more complex models that would take into account higher order dependencies. With a more complex model, traditional niching methods as tournament selection with continuous sharing could be used. However, using more complex models results in extra computational resources required to find such model and we would prefer to use the simplest model we can.
Similar property can be observed in a simple symmetrical two-max function with equally sized peaks which is defined as

\[ f_{\text{two-max}}(X) = \left| \frac{n}{2} - u \right|, \tag{1} \]

where \( u \) is the sum of bits in the input string, \( n \) is the length of the input string, and \( | \cdot | \) denotes absolute value. This function has two global maxima in \((0, 0, \ldots, 0)\) and \((1, 1, \ldots, 1)\), and the fitness of each solution is equal to the fitness of its complement. Even though the two-max is composed of two simple linear functions which are very easy to optimize by most evolutionary methods, their convergence on the two-max can get very slow.

The problem of symmetry was also studied on spin-glass systems in Naudts and Naudts (1998) and other problems, in all of which the simple genetic algorithms and other algorithms with the same bias experience great difficulties. However, the class of problems where we encounter symmetries in some form (not necessarily "clean" and "perfect" symmetry) contains many interesting and important real-world problems that could benefit from recombination. The question is how to resolve this problem without having to design special coding, heuristics, or other problem-specific operators. In the next section we propose a method that can be used in many evolutionary algorithms to alleviate the problem of symmetry and consequently improve their performance. Additionally, the method eliminates the source of difficulties of using niching in the PMBGAs and is the first attempt to address niching in the PMBGAs. Other niching methods can be used in concert with our method, and the method can be extended to also affect the selection by assigning each center the same amount of resources.

3 Improving Recombination by Clustering

In all problems mentioned above there are two complementary parts of the search space, each with the same structure. This structure can be very simple as in the two-max function where both parts are simple linear unimodal functions or more complex as in the graph partitioning where in most cases each part contains a large number of local optima. However, there exist algorithms that are able to deal with a wide range of problems and if they were able to distinguish between the two parts of the solution space, they would be able to optimize the problem very efficiently. The motivation to introduce clustering in evolutionary algorithms is that by helping the algorithm to separate the two or more complementary parts of the solution space, the problem of symmetry would be eliminated and the algorithms would simply not have to deal with it. By using algorithms which can solve the problem if the symmetry is not present in a problem (as a linear problem in case of two-max), the problems could be solved very efficiently, accurately, and reliably.

Recently, clustering was used in a businessman-customer scheme for an effective niching in genetic algorithms (Goldberg & Wang, 1997). It has proven to be a very powerful method for discovering and maintaining solutions close to a number of different optima. We use a similar scheme; however, clustering is
not used only to improve niching while selecting better solutions from the entire population, but also to separate unlike parts of the search space and process each part separately. Furthermore, clustering is not directly driven by fitness but the genotype itself.

A similar concept of using multiple populations, each corresponding to one optimum (in ideal case), was introduced by Hocaoglu and Sanderson (1995) who proposed the minimal representation size cluster genetic algorithm (MRSCGA). The MRSCGA starts with one intact population. Then, the population is clustered into a number of sub-populations by using the minimal representation criterion (Segen & Sanderson, 1981). Each sub-population is then processed separately for a number of generations. Subsequently, the statistics over the entire population is computed to get a new partitioning of the population, and the process is repeated, allowing each sub-population more and more generations as the time goes on. The MRSCGA was designed for dealing with multimodal functions and not to directly address the problem of symmetry. The issues of recombination and the effect of clustering to improve its power were not addressed. Moreover, the use of niching within this framework was not discussed and it was assumed that the niching comes up naturally by using multiple separated sub-populations.

Melhuish and Fogarty (1994) use restricted mating to search the state space in separate regions for an efficient learning of classifiers. Restricted mating selects and processes classifiers that best agree with the underlying state space model by picking a random region of the search space. Similarities among parents were also used to improve various selection and niching schemes (Deb, 1991; Horton, 1995; Harik, 1995).

In our framework the general algorithm proceeds as follows. The initial population is generated at random just like in the simple GAs. Each iteration of the main loop, the better solutions are selected from the current population. Before the recombination proceeds, the set of selected solutions is partitioned into a number of clusters. The number of clusters can be either given on input or determined by using hierarchical clustering methods or the minimal representation criterion (Segen & Sanderson, 1981). Recombination proceeds in each cluster separately and produces a number of new individuals, the offspring. Any recombination can be used, e.g. two-parent crossover of simple genetic algorithms, forward simulation of a probabilistic model learned by the Bayesian optimization algorithm, etc. The number of offspring produced by each cluster can be either proportional to its size or its average fitness which introduces niching and assigns each cluster resources proportional to its overall quality. The offspring are then incorporated into the original population, possibly replacing the entire population. The loop finishes when the termination criteria given by the user (e.g., convergence, maximum number of generations, etc.) are reached. The pseudo-code of the algorithm follows:

(1) \( t \leftarrow 0 \)
(2) Randomly generate initial population \( P(0) \).
(3) Select a set of promising strings \( S(t) \) from \( P(t) \).
(4) Cluster \( S(t) \) into \( k \) clusters \( C_i(t) \).
(5) Process each cluster \( C_i(t) \) separately to generate its offspring \( O_i(t) \).
(6) Create a new population \( P(t+1) \) by replacing a part of \( P(t) \) with \( O_i(t) \).
(7) Set \( t \leftarrow t + 1 \).
(8) If the termination criteria are not met, go to 3.

As it was pointed out above, the contribution of the proposed framework is twofold: (1) the negative effect of symmetry in a problem is alleviated, and (2) the use of effective niching in the PMBGAs is allowed. In the PMBGAs, the use of traditional niching methods often fails to achieve the goal and results in a very poor performance. Once niching can be incorporated into the PMBGAs, it can be used to improve their performance on difficult combinatorial problems, solve hierarchical problems (Pelikan, Goldberg, & Cantú-Paz, 2000), and tackle multi-objective problems by thoroughly searching the solution space for a diverse Pareto front. It is important to note that the effects of symmetry can be eliminated by using more complex models. However, the more complex the models get, the more resources it takes to learn them. Thus, avoiding using complex models whenever we can seems to be a step in the right direction.

4 k-Means Clustering

In k-means clustering, each cluster is specified by its center. Initially, \( k \) centers (where \( k \) is given) are generated at random. Each point is assigned to its nearest center. Subsequently, each center is recalculated to be the mean of the points assigned to this center. The points are then reassigned to the nearest center and the process of recalculating the centers and reassigning the points is repeated until no points change their location after updating the centers. The pseudo-code of the k-means clustering algorithm follows:

(1) Generate \( k \) centers \( C_i \) at random.
(2) Assign each point \( x_i \) to the nearest center.
(3) Assign each center to the centroid of the points assigned to it.
(4) If point locations have changed in step 2, go to 2.
(5) Return the cluster centers and point locations.

To cluster binary strings, we can simply use real vectors of the same length to represent the center of each cluster. Each variable in a vector can obtain values from \([0,1]\). Euclidean metric can be used to measure distance. Alternatively, phenotypic distance can be used to cluster the population which can be very useful on real-valued problems. In this case the centers can be also updated by computing frequency of each bit on each position and fixing each position of the genotype of the center to the most frequent value on this position. The value of the center would then be its phenotype.

The distance metric used in the clustering algorithm is also a very important issue and for very complex problems this may lead to anomalous results. In general, the more similar the genotype metric is to its phenotype equivalent, the better the clustering should work. Phenotype distance metric can also be used, if possible, as it was discussed above. More sophisticated clustering methods can
be used. We have chosen k-means clustering for its simplicity. We believe that the use of better clustering methods will lead to better results. However, we must keep an eye on the complexity of the used clustering method not to lose more than we gain.

5 Experiments

We have performed various experiments to demonstrate the effect of clustering on recombination. We have implemented clustering in two algorithms—the simple genetic algorithm with one-point crossover and the univariate marginal distribution algorithm. This study considers three problems: (1) two-max with equal peaks, (2) two-max with unequal peaks, and (3) graph-bisection. The two-max problem is very simple and was chosen to demonstrate ideal behavior of clustering. A similar behavior should be observed on more difficult problems by using more sophisticated operators together with clustering. Two cases were chosen in order to show the effects of niching. The remaining problems were chosen to demonstrate the power of clustering on highly multimodal problems with strong symmetries that are known to be very difficult for the simple genetic algorithms without using additional problem-specific operators dealing with symmetry.

The first problem is a simple two-max function which is a very good example to show the expected behavior of the extended algorithms. Even though most interesting problems are not as simple as the two-max, by using more sophisticated recombination operators similar behavior as the one of the simple univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paß, 1996) on the two-max can be observed. We used two two-max functions, one with equal peaks (see Equation 1), and one with one peak higher than the other by a factor of 1.5, defined as

\[
f_{\text{two-max}}^*(X) = \begin{cases} 
\frac{n}{2} - u & \text{if } u \leq \frac{n}{2} \\
1.5 \left( u - \frac{n}{2} \right) & \text{otherwise} 
\end{cases}
\]

The dynamics of the UMDA with two clusters on the two-max problems with equal peaks of size \( n = 50 \) is shown in Figure 1. Histograms with the number of ones in a solution on the horizontal axis and the number of corresponding solutions on the vertical axis are shown in a number of generations. The optima are on the left-most and right-most sides of the histograms. The size of the population was \( N = 500 \), binary tournament selection was used, and the offspring replaced entire original population. Initially, most of the mass is centered around the middle with the binomial distribution of mean \( \mu = \frac{n}{2} \) which follows from that the initial population is generated at random with a uniform distribution and the number of solutions with \( l \) ones is equal to \( \binom{n}{l} \). As the selection proceeds, it's pushing the population to both left and right sides. The clusters capture the division and each of them follows its optimum. The size of offspring generated by each cluster is proportional to its fitness, and thus the solutions are equally distributed between the two optima.

If the population were not separated into clusters each time we select parents, pressure from both sides would make it harder for the algorithm to converge
Fig. 1. Dynamics of the UMDA with two clusters on two-max with equal peaks of size $n = 50$. Generations 0, 5, 10, 15, and 20 are displayed from left to right.

to either side. Eventually, the population would drift and the algorithm would converge to one optimum. However, only one optimum would be found. If niching were used to eliminate drift and search for multiple solutions, the UMDA would require exponential time to converge to the optimum. Clustering allows the population to converge to the both optima very quickly and reliably. Another important point is that if we allowed generation gap and would keep best individuals of the original population, the lower optimum would be eliminated and all individuals would concentrate around the higher optimum. The effect of symmetry could also be alleviated by using more complex models; however, there is no reason to use complex models where simple models are sufficient.

Analogous experiment with the two-max with unequally sized peaks of size $n = 50$ is shown in Figure 2. As we can see, the algorithm finally discovers both optima and distributes individuals to each of these proportionally to their fitness. An interesting behavior can be observed in generation 20 (middle part of Figure 2). Even though more individuals are distributed in the neighborhood of the higher optimum, the algorithm converges faster to the lower one. This seems to be an effect of the tournament selection. At certain generation, the fitness of any individual close to the higher optimum is higher than the fitness of any individual close to the lower optimum. Then, the only chance to win tournament for the solutions close to the lower optimum is to compete with a worse solution from the same part. Thus, all winners from the lower basin of attraction are selected in competition with other members of this basin, and the pressure on their fitness is as if the sub-population on this basin was processed separately. However, the winners on the opposite basin often win only because they were competing with someone from the lower basin and only part of the time they compete with each other. This decreases the selection pressure on their fitness values, and the corresponding cluster converges slower.
It is important to note that if we used too strong selection, e.g. truncation selection with $\tau = 50\%$, we would eliminate the lower optimum before the population would converge to the both optima because even if we discovered both optima, the proportion of the higher optimum would be about 60\% of the population (proportional to its fitness). Truncation selection would then select only solutions at the higher peak. Thus, to have an effective niching, one must assure that the selection pressure is not too strong. This is an important issue of niching in general. Another important note is that the two-max problem with unequal peaks is not strongly affected by the problem of symmetry and the UMDA without clustering would be able to find the global optimum (and loose the local one).

We also performed some experiments on the graph bisection problem of a two-dimensional grid cut in halves which are connected by only two edges (Schwarz & Ocenasek, 1999). Each position in a string corresponds to one node in the graph and its value determines the classification of the corresponding node to either of the two classes. This problem requires a repair operator since only solutions with equal numbers of zeroes and ones are allowed. To repair the solutions, various operators can be used. In order to make our results independent of the ordering of the nodes in the solution strings, we use a randomized repair operator which turns on/off a required number of bits by randomly picking positions and changing the value of a particular bit if it moves the solution closer to a legal one. More sophisticated heuristic operators would improve the convergence significantly; however, this was not the goal of our experiments which were designed to show how powerful clustering can be.

The fitness of each string is determined by the size of the graph minus the number of edges between the two partitions of nodes specified by the string. The goal is to maximize this value. Since there are two edges between the optimal partitions, the optimal fitness is $n - 2$, where $n$ is the size of a problem (number of nodes in the graph). We have tried problems of size $n = 16$, $n = 36$, and $n = 64$. In all tested algorithms truncation selection with $\tau = 50\%$ was used and the worst half of the original population was replaced by the offspring.

All problems have shown to be extremely difficult for the simple GA with both uniform and one-point crossover, with or without mutation. The UMDA with an intact population also could not solve the problem. On the other hand, the UMDA algorithm with $k = 2$ clusters converged very fast on problems with $n = 16$ and $n = 36$ even though its equivalent with only one cluster was not
able to get better solutions than the ones in the initial population even with huge populations. The average number of evaluations until the population in the UMDA with \( k = 2 \) clusters got overtaken by optima in 9 of 10 independent runs is shown in Table 1. For a problem of size \( n = 64 \) and \( k = 2 \) clusters, the algorithm was often deceived to one of the local optima. Introducing additional two clusters resolved this difficulty and improved the performance of the algorithm. In all problems, the algorithm discovered both global optima which were equally distributed in the final population.

The graph bisection is a very complex problem with many local optima that are very hard to get from once deceived. Our results suggest that using clustering for difficult problems with symmetries can alleviate some difficulty and make the problem easier to be solved by even a very simple algorithm. Using more sophisticated algorithms (particularly, the Bayesian optimization algorithm (Pelikan, Goldberg, & Lobo, 1999)) on this problem results in much better scale-up behavior (Schwarz & Ocenasek, 1999).

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Pop. size</th>
<th>Number of clusters</th>
<th>Avg. number of evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>50</td>
<td>2</td>
<td>406</td>
</tr>
<tr>
<td>32</td>
<td>600</td>
<td>2</td>
<td>11633</td>
</tr>
<tr>
<td>64</td>
<td>5500</td>
<td>4</td>
<td>140555</td>
</tr>
</tbody>
</table>

Table 1. Results of the UMDA with clustering on the regular graph bisection.

6 Conclusions

Clustering of the population of parents and processing each cluster separately alleviates the problem of symmetry and makes the job for recombination easier. It can automatically be used to incorporate niching into the algorithm. Simple recombination methods can get much further with than without clustering; however, clustering does not solve the entire problem by itself and more sophisticated operators together with clustering must be used for an efficient scale-up behavior on difficult problems. Assuming we have a powerful enough method for combining promising solutions that is capable of covering important interactions of the decision variables in a problem, clustering can be used very efficiently to thoroughly search the solution space and discover multiple optima. However, for simple problems the use of clustering results in unnecessary loss of diversity and leads to an increase in an adequate population size. When the model is sufficient to capture the structure of the problem, including its symmetry, clustering needs not be incorporated and other niching methods can be used instead. However, by using clustering on symmetrical problems, simpler models can be used with the same effect.

Acknowledgments

The authors would like to thank Martin Butz, Erick Cantú-Paz, Bart Naudts, and Dirk Thierens for useful discussions and insights.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-97-1-0050 and F49620-00-0163. Research funding for this work was also provided by a grant from the National Science
Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of the Slovak Grant Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References


Pruefer Numbers and Genetic Algorithms: A Lesson on How the Low Locality of an Encoding Can Harm the Performance of Gas

Franz Rothlauf\textsuperscript{1} and David E. Goldberg\textsuperscript{2}

\textsuperscript{1} Department of Information Systems (BWL VII)
University of Bayreuth / Germany
\textsuperscript{2} Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
rothlauf@uni-bayreuth.de, deg@illigal.ge.uiuc.edu

Abstract. When handling tree networks, researchers have sometimes tried using the Pruefer number representation for encoding networks, however GAs often degraded when used on this encoding. This paper investigates the locality of the Pruefer number and its affect on the performance of a Genetic Algorithm (GA). The locality describes how the neighborhood of the genotype is preserved when constructing the phenotype (the tree) from the genotype (the Pruefer number). It is shown that the locality of the Pruefer number is highly irregular on the entire solution space, and that the performance of a GA depends on the structure of the optimal solution. A GA is able to perform well only for networks that have a high locality (stars). For all other types of networks (lists, trees) the locality is low and a GA fails to find the best list or tree. Using a GA with the Pruefer number encoding can be useful, when the best solution tends to be a star.

The locality of an encoding could have a strong influence on the performance of a GA. When choosing encodings for optimization problems, researchers should be aware of this and be careful with low locality encodings. If the locality of the encoding is low, a failure of the GA is often unavoidable.

1 Introduction
When applying GAs to tree networks the Pruefer number is sometimes used for encoding the trees (Julstrom, 1993; Gen et al., 1998; Abuali et al., 1995). This type of encoding often results in poor GA performance, which has been reported to be due to the low locality of the encoding (Palmer & Kershenbaum, 1994).

In this paper we investigate the locality of the Pruefer number encoding and the performance of a GA for a simple One-Min problem. The locality describes how well the neighborhood of the genotype is preserved when constructing the phenotype from the genotype. The problems of a GA with low locality is illustrated for different network types (stars, lists, trees). We have gained a deeper
insight into why this encoding fails, we illustrate the problem of GAs with low local- 
ity encodings, and give hints to other researchers how they could use this represen-
tation, or define new ones more effectively.

The remainder of the paper is organized as follows. We start with a short 
overview of how the Pruefer number is constructed. Section 3 describes some 
properties of the Pruefer number encoding. A description of the computer ex-
periments is given in Section 4. The results for the locality and the performance 
of the GA are presented in section 5. The paper ends with a summary of results.

2 From the phenotype to the genotype of a tree and vice 
versa

This section gives a short review on how to get a Pruefer number (the genotype) 
from a tree (the phenotype) and how to get the tree back from the Pruefer 
number.

Tree networks are generally defined as an undirected and connected graph 
with no cycles. A tree with \( n \) nodes has exactly \( n - 1 \) links. For a graph with \( n \) 
nodes there are exactly \( n^{(n-2)} \) trees. The Pruefer number for a \( n \)-node tree is a 
number with \( n - 2 \) digits and each digit is of base \( n \) (compare fig. 1).

2.1 The construction of the Pruefer number

Every tree has at least two nodes which have only one connecting edge (the 
degree of the node is 1). All nodes are labelled with numbers from 1 to \( n \). The 
Pruefer number can be constructed using the following algorithm:

1. Let \( i \) be the lowest numbered node of degree 1 in the tree.
2. Let \( j \) be the one node, which is connected to \( i \) (there is exactly one!). The 
number of the \( j \)th node is added at the right side of the Pruefer number.
3. Remove node \( i \) and the edge \( (i,j) \) from the tree and from further consider-
ation.
4. Go to 1 until only two nodes (that means one link) are left.

After termination one has a Pruefer number with \( n - 2 \) digits which represents 
the tree. Illustrative examples for constructing the Pruefer number can be found 
in Palmer & Kershenbaum, 1994 or Gen et al., 1998.

2.2 The construction of the tree from the Pruefer number

The construction of the tree from the Pruefer number follows the construction 
of the Pruefer number.

1. Let \( P \) be a Pruefer number with \( n - 2 \) digits. All node numbers which are 
not in \( P \) can be used for the construction of the network (are eligible).
2. Let \( i \) the lowest numbered eligible node. Let \( j \) be the leftmost digit of \( P \).
3. Add the edge \( (i,j) \) to the tree.
4. Designate $i$ as no longer eligible and remove the leftmost digit $j$ from the Pruefer number.
5. If $j$ does not occur anywhere else in the remaining Pruefer number, designate $j$ as eligible.
6. Go to 2 until there are no remaining digits in the Pruefer number. If no digits are left then there are exactly two numbers $r$ and $s$ eligible. Finally add the edge $(r,s)$ to the tree.

3 Properties of the Pruefer number encoding

In this section we give a short overview on the properties of the Pruefer number particularly with regard to the use of genetic and evolutionary algorithms (Palmer, 1994).

The use of the Pruefer number has some remarkable benefits:
- every tree can be represented by a Pruefer number
- only trees are represented by Pruefer numbers
- every Pruefer number represents exactly one tree
- all trees are equally represented

A look at the construction rule of the Pruefer number shows that the Pruefer number is able to represent all possible trees. Each tree has at least two nodes with degree 1 so the construction rule can be applied to all possible trees. It was shown by Pruefer (1918) that the Pruefer number represents only trees. This means random numbers can be created and that they always represent a tree. In contrast to many other representations for trees, repairing of a randomly chosen representation is not necessary. In addition, all trees are represented equally. Each tree is represented by exactly one specific Pruefer number. The mapping from the Pruefer number to the tree is one to one. The number of different trees for a graph with $n$ nodes is $n^{n-2}$, which is equal to the number of possible different Pruefer numbers.

However the Pruefer number also has some disadvantages; complex calculation, and low locality. The calculation of the Pruefer number is a little complex, but it can be done in $O(n \log n)$ (Palmer & Kershenbaum, 1994). The real disadvantage of the Pruefer number is the low locality of the representation. The mapping from the genotype to the phenotype does not preserve the neighborhood. Small changes in the representation can lead to large changes in the represented network and vice versa. Most of the individuals that are created with basic operators such as mutation or crossover are not similar to their parents. The operators work more like a random than a guided search. This behavior is illustrated in section 5.4.

4 Experiments

This section describes the computer simulations for evaluating the locality of the Pruefer number encoding and the performance of GAs for different network structures like stars, lists and trees.

We present results for four different types of networks:
(i) Star: One node is of degree $n - 1$ and the rest of the nodes have degree 1.
(ii) Ordered list: The nodes are connected according to their number. Node $n$ is connected to $n + 1$, node $n + 1$ is connected to $n + 2$ and so on. Two nodes are of degree 1 (the first and the last of the list) and the rest of the nodes have degree 2.
(iii) Random list: Like the ordered list, but the order of the nodes in the list is random.
(iv) Tree: Every other tree network

We distinguish between ordered and random lists, because the locality of the Pruefer number and the performance of the GAs is different for ordered and random lists.

Fig. 1 shows the encoding of the network as a Pruefer number and the encoding of the Pruefer number as a bitstring. Finally, the GA works on the bitstring representation. Looking at the locality means having either a random walk through the search space, or exploring the complete neighborhood of one individual. We investigate the locality in six different ways:

(i) random walk through the bitstring (one step is a one bit change in the bitstring)
(ii) random walk through the Pruefer number (change of one digit in the Pruefer number)
(iii) random walk through the phenotype (change of one edge in the tree)
(iv) neighborhood of a bitstring (all neighbors that are different in one bit from the examined individual)
(v) neighborhood of a Pruefer number (all neighbors that differ in one digit)
(vi) neighborhood of a tree network (all neighbors are different in one edge)

A random walk through the bitstring/Pruefer number representation of a tree means changing one bit/digit and examining how many edges in the network are different. A random walk through a tree means, that one edge of the tree is replaced by a randomly chosen edge and the difference of bits/digits in the bitstring/Pruefer number is examined. The start individual for the bitstring, the Pruefer number or the tree is chosen randomly. To gain statistically significant information independently of the start individual, 400 steps were carried out in each of the 20 runs. Exploring the neighborhood of the Pruefer number/bitstring means having a look at all individuals, which are different in one bit/digit in comparison to the examined individual, and counting how many edges have changed in the tree. Investigating the neighborhood of a tree means looking at all individuals which are different in one edge from the examined network. For each problem 20 independent runs were performed.

For our investigation in the performance of a GA we defined a simple One-Min problem. The fitness function is defined as the distance to the optimal solution. The optimal solution has fitness 0. The fitness of all other networks is equal to the number of different edges in comparison to the optimal solution.

---

1 For each problem we perform 8000 steps in the search space
This means for a $n = 16$ node problem that a tree which has 12 links in common with the best solution has fitness 3. The fitness values of the tree range between 0 and $n - 1$.

For finding the optimal solution we use a traditional simple GA (Goldberg, 1989) with one-point crossover, no mutation and a $(\mu + \lambda)$ selection scheme (Bäck & Schwefel, 1995). For each generation we produce from $\mu$ parents $\lambda$ offspring and choose from all $(\mu + \lambda)$ individuals the $\mu$ best individuals. For the 16 node problems $\mu = \lambda = 400$ and the GA stops after 50 generations. With 32 nodes, $\mu$ and $\lambda$ were set to 1500, and the GA was stopped after 70 generations.

200 runs were performed for each problem. Results are presented for 16 and 32 node tree networks.

5 Results

This section presents results investigating the locality of the Pruefer number and the performance of a traditional GA. We start by looking at how many neighbors a bitstring, a Pruefer number or a tree has. This is followed by an investigation into the locality by performing a random walk through the phenotypes and genotypes of trees and looking at the neighborhood of different network types (stars, lists and trees). The section ends with results for the performance of the GA for different types of optimal solutions.

5.1 Number of neighbors

In table 1 we present the number of neighbors for different types of networks. Assume a tree network with $n$ nodes. Each star network has exactly $(n-1)(n-2)$ neighbors. For a list network each tree has exactly $\sum_{i=1}^{n-1} i(n - i) - 1 = n \sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 - (n-1) = \frac{1}{2} n^2 (n-1) - \frac{1}{6} n(n-1)(2n-1) - n + 1 = \frac{1}{6} n(n-1)(n+1) - n + 1$ neighbors. The number of neighbors for tree networks is between the number for star and list networks and must be calculated for each network separately.

We prefer the $\mu + \lambda$ selection scheme in comparison to traditional roulette wheel or tournament selection schemes with replacement, because some kind of elitism helps in finding good solutions for this simple One-Min problem. Our experiments have shown that without elitism a GA can not find its way to the optimal solution.
depends on the different degree of the nodes in the tree. Each Pruefer number with \( k = n - 2 \) digits has exactly \((n - 1) \cdot (n - 2)\) neighbors. Each Pruefer number with \( n - 2 \) digits is encoded as a bitstring of length \((n - 2) \cdot \lceil \log_2(n) \rceil\). So each bitstring has \((n - 2) \cdot \lceil \log_2(n) \rceil\) neighbors. A change of one digit in the Pruefer number can result in up to \( \lceil \log_2(n) \rceil \) different bits.

Comparing the number of neighbors of a Pruefer number to the numbers of neighbors of a tree shows a one to one relationship to star networks. A star has as many neighbors as a Pruefer number. It can be seen in subsection 5.3 that for star networks the locality is perfect, too. When modifying the tree toward a list network the number of neighbors for the Pruefer number and bitstring stays constant, whereas the number of neighbors for the tree increases. This results in different locality for different areas of the solution space.

<table>
<thead>
<tr>
<th>nodes</th>
<th>tree</th>
<th>Pruefer number</th>
<th>bitstring</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>star ( \frac{1}{2} n(n - 1)(n - 2) )</td>
<td>((n - 1)(n - 2))</td>
<td>((n - 2) \cdot \lceil \log_2(n) \rceil)</td>
</tr>
<tr>
<td>list</td>
<td>( \frac{1}{2} n(n - 1)(n + 1) - n + 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>star</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>list</td>
<td></td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>star</td>
<td>210</td>
<td>210</td>
</tr>
<tr>
<td>list</td>
<td></td>
<td>665</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>star</td>
<td>930</td>
<td>930</td>
</tr>
<tr>
<td>list</td>
<td></td>
<td>5425</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Number of neighbors for tree, Pruefer number and bitstring

5.2 Random walks

A look at the results for a random walk through the bitstring (fig. 2) and the Pruefer number (fig. 3) shows that only about 40% of the one bit/digit changes lead to a change of one edge in the tree. More than 35% (16 nodes) or 50% (32 nodes) of all one bit/digit changes result in networks with at least four different edges. The locality of the genotype (bitstring as well as Pruefer number) is low.

When walking through the solution space of the phenotype (tree networks) the plots in figure 4 show that only about 50% of all one link changes result in a change of less than eight bits (16 nodes), resp. 20 (32 nodes) bits in the bitstring. For the Pruefer number (fig. 5) about 75% of the neighboring individuals are different in more than one digit. The locality of the phenotype is low.

5.3 A look at the neighborhood

The neighborhood of a bitstring (fig. 2) and a Pruefer number (fig. 3) is different for different network types. An investigation in the neighborhood of a bitstring/Pruefer number that encodes a star network shows that a change of one bit/digit results in a change of exactly one edge in the encoded star network. The locality of the bitstring/Pruefer number is perfect for star networks. A look at the neighbors of bitstrings and Pruefer numbers that encode ordered list networks shows that the change of one bit/digit results in a maximum change of 4
edges (fig. 2 and fig. 3) in the tree independently of the number of nodes. The locality of the Pruefer number for ordered list networks is worse than for stars, but better as for tree or random list networks. Bitstrings and Pruefer numbers that encode random lists or random tree networks show very low locality. The neighborhood of tree networks shows the same behavior as a random search. All neighbors of a bitstring (fig. 2) and a Pruefer number (fig. 3) have phenotypically not much in common with the considered network.

If the neighborhood of a star network with \( n \) nodes is investigated, the change of one edge results in the change of one digit in the Pruefer number (fig. 5), but up to \( \lceil \log_2(n) \rceil \) bits in the bitstring that represents the Pruefer number (fig. 4). The locality of the phenotype is perfect for star networks. For tree and list networks the change of one edge results most of the time in a completely different bitstring/Pruefer number (fig. 4 and fig. 5). The locality of the phenotype is very low for tree and list networks.

5.4 The performance of a GA

In fig. 6 we show the performance\(^3\) of the GA from section 4 for different optimal solutions. All parameters of the encoding and the GA remain the same for the different problems. Only the type of the best solution is different. For star networks, the GA is able to find the optimal solution without problems. In all 200 runs, the GA finds the optimal solution in less than 24 (16 nodes) or 43 (32 nodes) generations. However, if the optimal solution is a random list, or a random tree, the GA can never find the optimal solution and is completely misled because of the low locality. For ordered lists, the GA performs slightly better, but nevertheless the locality is so low that the GA fails.

We see that when a GA explores the neighborhood of a star network, it is really a search around one individual. The GA is able to find its way to the optimum. When it comes to searching the solution space around an ordered list, or even worse a random list or a tree network however, the search is not more than a blind and random search.

6 Summary and Conclusions

This paper presents an investigation into the locality of the Pruefer number and the performance of GAs using this encoding.

We illustrate that the locality of the phenotype (the tree) and the genotype (either Pruefer number or bitstring) is not homogenous. When looking at star networks the Pruefer number encoding shows a perfect locality. A change of one edge leads to the change of one digit of the Pruefer number and vice versa. A GA performs pretty well for the One-Min problem, if the optimal solution is star-like. Even though the locality is only slightly worse for ordered list networks, a GA is not able to find the optimal ordered list. For random lists and tree networks the locality is low. Every small change of the Pruefer number leads to a large change

---

\(^3\) Because of illustrative purposes the error bars are missing. The standard deviation is for all plots smaller than 1 (\( \sigma < 1 \))
Fig. 2. Cumulative frequency of how many edges are different in the tree, if one bit in the bitstring is changed for 16 (left) and 32 (right) node networks.

Fig. 3. Cumulative frequency of how many edges are different in the tree, if one digit of the Pruefer number is changed for 16 (left) and 32 (right) node networks.

Fig. 4. Cumulative frequency of how many bits in the bitstring are different, if one edge of the tree is changed for 16 (left) and 32 (right) node networks.
Fig. 5. Cumulative frequency of how many digits in the Pruefer number are different, if one edge of the tree is changed for 16 (left) and 32 (right) node networks.

Fig. 6. The performance of a Genetic Algorithm for 16 (left) and 32 (right) node networks. The structure of the best solutions has a big influence on the performance of the GA. If the best solution is a star, the GA performs well. If the GA has to find a best solution that is a list of a tree, it degrades and cannot solve the easy One-Min problem. For each problem 200 runs were performed.

in the encoded tree and vice versa. A GA cannot find the optimal solution, or even come close to it. Unfortunately, every star network has only one edge in common with another star, so the areas of high locality are separated from each other by large areas of low locality.

We have seen that the locality of an encoding has a strong influence on the performance of a GA. When choosing encodings for problems, researchers should be aware of this and be careful with low locality encodings. If the locality of the encoding is low, a failure of the GA even for simple problems is often inescapable.
References


Abstract. We describe a new selection scheme for steady-state evolution strategies, median selection. In steady-state algorithms, only one individual is generated and evaluated at each step and is immediately integrated into the population. This is especially well suited for parallel fitness evaluation in a multiprocessor environment. Previous steady-state selection schemes resembled $(\mu + \lambda)$ selection, which has a disadvantage in self-adaptation of the mutation step length. Median selection is similar to $(\mu, \lambda)$ selection. Median selection is compared with other steady-state selection schemes and with $(\mu, \lambda)$ selection on a uniprocessor and on a multiprocessor. It achieves equally good or better results as the best other selection scheme for a number of benchmark functions.

1 Introduction

Evolution Strategies (ES) were developed by Ingo Rechenberg and Hans-Paul Schwefel (Rechenberg, 1973, Schwefel, 1977) for multi-dimensional, real-valued parameter optimization problems. The most important property of ES is the ability of self-adaptation. With self-adaptation the variance of mutations on the object variables are adapted at runtime and thus the optimization progress is improved. Methods for self-adaptation are for example the 1/5-success-rule, mutative step control (Rechenberg, 1994), derandomized step control (Ostermeier et al., 1993) or covariance matrix adaptation (CMA) (Hansen and Ostermeier, 1996). The selection method plays an important role for self-adaptation. In (Schwefel, 1992) it is shown, that regarding speed of self-adaptation, comma selection is superior to plus selection (see next section). The $(\mu + 1)$ ES was proposed early by (Rechenberg, 1994), but nowadays is no longer used because it is missing a scheme for realizing self-adaptation (Rudolph, 1997). Median selection, which is presented in this paper, eliminates this disadvantage.

This paper is organized as follows: in Sect. 2 existing selection methods are presented and in Sect. 3 the new median selection is described; then in Sect. 4 the optimization results on a number of benchmark functions and are presented, which are discussed in Sect. 5. Finally, there are the conclusions in Sect. 6.
2 Selection in Evolution Strategies

2.1 Comma and Plus Selection

In evolution strategies generally the comma or plus selection is used, denoted as \((\mu, \lambda)\) and \((\mu + \lambda)\). In plus selection, the \(\mu\) parent individuals plus the resulting \(\lambda\) offspring individuals form the selection pool for the parents of the next generation. This causes an elitist selection where the best individual is always contained in the next generation.

In the \((\mu, \lambda)\) selection, only the \(\lambda\) offspring individuals form the selection pool. Thus it is possible that the best offspring individual is worse than the best parent individual and hence a regression happens. Nevertheless, this selection is better suited for adaptation of the step-lengths of the individuals (Schwefel, 1992), because in every generation the possibility of changing the strategy parameters exists.

2.2 Selection in Steady-State Algorithms

In contrast to generational evolutionary algorithms, where a whole offspring population is created in every generation, in steady-state ES only one or a few individuals are created per step and immediately integrated back into the parent population. The term “steady-state” expresses that in one step only a small change takes place and not the whole population changes. The basic algorithm step of steady-state ES is the following:

1. create a new individual and evaluate it with the fitness function,
2. (a) select an old individual which may be replaced by the new one,
   (b) decide, if the old individual will be replaced.

In step 2a one can chose the replacement strategy, e. g. replacement of the worst, the oldest or a randomly chosen individual. In step 2b one can chose the replacement condition, e. g. replacement if the new individual is better, or unconditional replacement. A widely used combination is to replace the worst individual only if the new individual is better (Bäck et al., 1997, Glossary, Smith, 1998, p. 8). This is an elitist selection and corresponds to the \((\mu + 1)\) strategy. In our simulations, this is denoted as “standard steady-state” selection.

2.3 Steady-State Algorithm with Local Tournament-Selection

Another steady-state algorithm compared here, was inspired by (Smith and Fogarty, 1996). The idea is to generate only a small number \(\lambda\) of offspring individuals and select one of them (e.g. the best) to integrate it into the main parent population. This is a kind of local tournament selection: \((1, \lambda)\). It has a high selection pressure and is distinctive like the normal comma selection. Parallelization of this algorithm is easy: instead of immediately integrating an evaluated offspring individual, \(\lambda\) offspring individuals are collected in a buffer and only the best of them is then integrated in the parent population. In the experiments this algorithm is denoted as “steady-state with local tournament selection”.

3 Median Selection

The motivation for the design of the median selection was to get a selection scheme with the following properties:

- it should evaluate and integrate only one individual per step,
- it should be a non-elitist selection, which facilitates self-adaptation; a temporary worsening of the average fitness should be allowed, like in the \((\mu, \lambda)\) selection.

The idea behind the median selection is, that the decision, whether an individual is integrated into the population or not, is made by a decision function without the context of other individuals. This makes it easy to realize a steady-state selection. The function uses data about the fitness distribution of formerly generated individuals which have already passed the selection process. Using this data, the behavior of a \((\mu, \lambda)\) selection is modeled by determining the fitness limit, which separates the \(\mu\) best individuals from the \(\lambda - \mu\) remaining individuals. No further replacement condition is needed.

The model of Fig. 1 was assumed for the \((\mu, \lambda)\) selection. Out of \(\mu\) parent individuals \(\lambda\) offspring individuals are generated. Thereof the \(\mu\) best are selected as parents for the next generation. To do this, it is useful to sort the offspring individuals according to their fitness. If \(f_{lim}\) is the acceptance limit fitness value, a new offspring individual \(i\) is integrated if \(f_i \geq f_{lim}\). \(f_{lim}\) is the \(\mu\)-smallest value (for minimization) or the \(\mu\)-median of the fitness values of the offspring. Hence the name.

A model is used for the distribution of the fitness values and \(f_{lim}\) is determined from it. With every newly created and evaluated individual the model is updated. Because the average fitness of the population should be improving permanently, it is desired that relatively old fitness values are removed from the model. Instead of modeling a particular distribution, the distribution is represented by a sample of the last \(n_p\) fitness values of the created offspring individuals. For this a FIFO fitness buffer whose elements additionally are linked in sorting order, is used (Fig. 2). This is not a priority queue, as elements leave the FIFO after \(n_p\) steps, regardless of their fitness. This buffer can be accessed in two ways:

1. in FIFO organization, for insertion of a new fitness value. It remains \(n_p\) steps in the buffer and then drops out.
2. in sorted order according to the fitness value, for determining the $\mu$-median.

The operation of inserting an element into the FIFO buffer is denoted in the algorithm below by `fifo_insert()`. Access to the $k$-smallest element is realized by the function `fifo_getSorted(k)`. For determining the $\mu$-median, which represents the acceptance limit $\mu_{lim}$, the fitness value at index $n_p \cdot \frac{\mu}{\lambda}$ has to be accessed in the sorted buffer.

Additional parameters for the median selection are the length of the FIFO-buffer $n_p$ and the relative rate of acceptance $\rho = \frac{\mu}{\lambda}$ which determines the acceptance limit in this way: $\mu_{lim} = \text{fifo_getSorted}(r_p \cdot n_p)$. The buffer length $n_p$ corresponds to the number of offspring individuals of a corresponding $(\mu, \lambda)$ ES. But now we have the advantage to chose $n_p$ lower than we would chose $\lambda$, because this does not primarily affect the selection pressure like $\lambda$ and it speeds up the adaptation of the fitness acceptance limit. (Smith and Fogarty, 1996) use a ratio of $\frac{\mu}{\lambda} = \frac{1}{5} = 0.2$. (Bäck, 1992a) uses a ratio of $\frac{\mu}{\lambda} \approx \frac{1}{6} \approx 0.16667$. In evolution strategies the ratio $\frac{\mu}{\lambda} = \frac{15}{100} = 0.15$ is often used (Ostermeier et al., 1993).

The algorithm for the steady-state ES with median selection is:

```
t = 0;
initialize($P_\mu(0)$); evaluate($P_\mu(0)$);
fifo_init();
while (not termination) do
    $I = \text{recombine}(P_\mu(t))$;
    $I' = \text{mutate}(I)$;
    evaluate($I'$);
    $\mu_{lim} = \text{fifo_getSorted}(r_p \cdot n_p)$;
    if ($f(I')$ better than $\mu_{lim}$) then
        select Individual to replace ($I_{repl}$)
        replace $I_{repl}$ by $I'$;
    endif
    fifo_insert($f(I')$);
    $t = t + 1$;
endwhile
```
The selection of the individual to replace $I_{rep}$ can be performed by one of the replacement methods mentioned in Sect. 2.2: replacement of the worst, oldest or a randomly chosen individual.

4 Evaluation

The following functions numbered according to (Bäck, 1992b) were used as test functions (formulas are not presented here due to space limitations):

- $f_2$ Generalized Rosenbrock’s Function (unimodal, correlated variables),
- $f_6$ Schwefel’s Function 1.2 (unimodal),
- $f_9$ Ackley’s Function (multimodal)
- $f_{15}$ Weighted Sphere (unimodal, different weights for each variable),
- $f_{24}$ Kowalik (multimodal).

All simulations were done with EvA (Evolutionary Algorithms) (Wakunda and Zell, 1997), our own system which contains a large number of variants of genetic algorithms and evolution strategies.

In the simulations a population size of $\mu = 20$ was used consistently to ensure comparability. This is especially necessary for the multimodal functions $f_9$ and $f_{24}$ in order not to converge into a local optimum.

The simulations on one processor were performed on PCs, the multiprocessor simulations were performed on a Hewlett Packard V2200, a 16 processor shared memory machine, using the MPI library. The parallel version of the program runs on 2 or more processors and consists of one master processor for the core ES algorithm and one or more worker processors for (asynchronous) fitness function evaluations. So, the lowest useful number of processors in the parallel version is 3.

For all simulations covariance matrix adaptation (CMA) was used for adaptation of the strategy parameters, because it is currently the most powerful adaptation method (Hansen and Ostermeier, 1996). The compared strategies are:

1. $(20, \lambda)$ Evolution Strategy (comma) (only sequential simulations),
2. $(20 + 1)$ steady-state ES with replace worst, if better; the “standard steady-state”-algorithm,
3. $(20 + (1, \lambda))$ steady-state ES with local tournament selection, replacement strategy replace oldest and replacement condition always (selection takes already place in local tournament),
4. $(20 + 1)$ steady-state ES with median selection, also with replacement strategy replace oldest and replacement condition always.

In simulations prior to the tests listed here, it was shown that the replacement strategy replace oldest is advantageous in evolution strategies: it causes a non-elitist selection (in contrast to replace worst), which is also the case in $(\mu, \lambda)$ selection.

For the different parameters to set for these strategies, no static standard values were used, but for every function the optimal values were determined separately by an extra experiment. These are the following parameters:
- \((20, \lambda)\) ES: optimal \(\lambda\),
- \((20 + (1, \lambda))\) ES with local tournament selection: optimal tournament size \(\lambda\),
- \((20 + 1)\) ES with median selection: optimal buffer size \(n_p\), the acceptance limit \(r_p = 0.15\) turned out to be good for all simulations.

These values were determined only for the sequential version and were then used also for the parallel version. The actual values are given with each function. For the \((\mu, \lambda)\) ES only sequential results are shown, because it is not very well suited for this kind of parallelization and the other algorithms were more promising.

In all experiments with the sequential algorithm 30 runs were evaluated for each strategy with different values for the random number generator. With the parallel algorithm, 20 runs were evaluated for each strategy and every number of processors. Function \(f_{24}\) demanded lower computation resources and had lower convergence rates, so 100 runs were made with all numbers of processors.

4.1 Generalized Rosenbrock's Function \(f_2\)

Function \(f_2\) was calculated with dimension \(n = 20\), termination criterion was reaching a fitness value less than \(\Delta = 10^{-20}\) with a maximum of \(t_{\text{max}} = 270.000\) function evaluations. For the comma-ES \(\lambda = 80\) was chosen, for the steady-state ES with local tournament selection \(\lambda = 5\) was chosen and the buffer size of the median-ES was \(n_p = 40\) (the acceptance limit is \(r_p = 0.15\) for all tested functions).

The results are shown in Fig. 3(a). Standard Steady-State and Median selection need nearly the same number of function evaluations, at 9 processors Median needs about 8% more, this is the maximum difference. The difference to local tournament selection is significantly greater: about 10% to 35% in relation to standard steady-state. For local tournament with 7 and 15 processors none of the 20 runs did converge to \(10^{-20}\). In general the convergence ratio was significantly worse than for the other algorithms.

The comma strategy needs nearly twice as much function evaluations as the steady-state algorithms on one processor. This is similar for the other functions and will be discussed in more detail in Sect. 5.

4.2 Schwefel's Function 1.2 \(f_6\)

Function \(f_6\) was calculated with dimension \(n = 20\), \(\Delta = 10^{-20}\), \(t_{\text{max}} = 100.000\). The free parameters optimized were \(\lambda = 70\) (comma-ES); \(\lambda = 5\) (local tournament); \(n_p = 70\) (median).

The results are shown in Fig. 3(b). Here Median selection needs only between 80% and 87% of the function evaluations of the standard steady-state algorithm. This means that Median selection is able to adapt the step sizes better. Local tournament selection is here for one processor as good as standard steady-state, but for other number of processors, it needs clearly more function evaluations. The convergence rates for local tournament are quite surprising. With one or three processors, all 20 runs converged. This is the same for the other methods,
Fig. 3. Line graphs: comparison of numbers of function evaluations until the termination criterion is reached. Bar graphs: comparison of the number of not converged runs. Details see text.
4.3 Ackley's Function $f_9$

Function $f_9$ was calculated with dimension $n = 20$, $\Delta = 10^{-10}$ (due to limited computing precision), $t_{\text{max}} = 150.000$. The free parameters were optimized to: $\lambda = 60$ (comma-ES); $\lambda = 5$ (local tournament); $n_p = 40$ (median).

The results are shown in Fig. 3(c). The results are similar to function $f_6$: Median selection constantly needs less evaluations than standard steady-state. Local tournament is here a little better for one processor, but this is not the case for more than one processors, where it needs clearly more evaluations than the other two parallel methods. The convergence rate for Median were always a little worse than for standard steady-state. Local tournament selection failed to adapt in many runs.

4.4 Weighted Sphere Model $f_{15}$

Function $f_{15}$ was calculated with dimension $n = 20$, $\Delta = 10^{-20}$, $t_{\text{max}} = 160.000$. The free parameters were optimized to: $\lambda = 65$ (comma-ES); $\lambda = 4$ (local tournament); $n_p = 40$ (median).

The results are shown in Fig. 3(d). Standard steady-state and the Median method show almost equal behavior, the convergence rates are 100% for both methods. The local tournament method needs up to two times more function evaluations than the other two, convergence rates are between 0% and 100%.

4.5 Kowalik $f_{24}$

The dimension of function $f_{24}$ is fixed at $n = 4$. The optimum of $f_{24}$ is given in literature (Bäck, 1992b) with $\min(f_{24}) \approx f_{24}(0.1928, 0.1908, 0.1231, 0.1358) \approx 3.07485988 \cdot 10^{-4}$. Termination criterion was reaching a fitness value less than
3.07486 \cdot 10^{-4} \text{ with a maximum of } 200,000 \text{ function evaluations. The following free parameters were chosen: } \lambda = 100 \text{ (comma-ES); } \lambda = 6 \text{ (local tournament); } n_p = 40 \text{ (median).}

Because all compared strategies reached the global optimum at maximum only in half of the runs and this function needed clearly less evaluations than the others, we used 100 runs per strategy to obtain more significant results. The results are shown in Fig. 4(a). The best results are achieved by the standard steady-state algorithm, which needs about 10,000 function evaluations for all numbers of processors. For some numbers of processors, Median selection achieves roughly the same results, but for others, it needs up to 1.5 times more evaluations. Local tournament constantly needs more evaluations than the other two steady-state methods, the convergence rate is worse for some number of processors.

5 Discussion of the Results

The comparisons were all performed with the same number of parent individuals \( \mu = 20 \). Thereby the \((20, \lambda)\)-ES needs more function evaluations than the steady-state algorithms. The reason for this probably lies in the interdependence of the population size and the selection pressure of the comma strategy, which is given by \( \mu/\lambda \). For a fixed \( \mu \) and fixed selection pressure, a large \( \lambda \) has to be chosen. But above a certain value, an increase of \( \lambda \) has no significant effect towards progress. In this case, it is more efficient to take several smaller steps with a reduced size offspring population.

The strategy with local tournament selection seems to be not so suitable for evolution strategies. It fails to adapt step sizes correctly in many cases and needs more function evaluations than the standard steady-state methods or even is not able to make significant progress at all and the optimization stagnates.

The new method Median selection shows a better performance than the other tested methods for the functions \( f_6 \) and \( f_9 \). For the other three test functions, Median selection shows similar performance as the standard steady-state method or is only slightly worse for some numbers of processors. The high computing resources needed for the parallel measurements together prevented a still higher number of runs per data point.

Median selection introduces the two new parameters \( n_p \) and \( r_p \), but makes the parameter \( \lambda \) obsolete. It seems to be very robust for a fixed setting of these parameters for all numbers of processors.

Regarding the number of function evaluations needed with increasing number of processors, there is only a relatively small increase. This is due to the overlapping of the asynchronous handled fitness evaluations. This promises a near linear speedup and is very good to reduce the computation time for real applications, which need a high amount of computing power.
6 Conclusions

The new selection method median selection for steady-state evolution strategies was presented and compared for a number of test functions with other steady-state selection methods and the generational \((\mu, \lambda)\) ES. It indicated that median selection enables self-adaptation as well as or even better than all other selection methods. The algorithm is very well suited for asynchronous, parallel fitness evaluation, which is the preferred parallelization method for optimization problems with the need for high computing resources. Furthermore it turned out that the use of a steady-state evolution strategy is valuable even on a single processor computer without parallel evaluation of the individuals. This is true especially for multimodal functions.

References


The Origination of Diversity by Adaptive Clustering

N Walton and G D Smith

University of East Anglia
Norwich
England

Abstract. We propose to show from a study of population genetics that convergence in the simple genetic algorithm is due to the homogeneous nature of its population. By applying an adaptive clustering algorithm we demonstrate that highly fit yet diverse populations result. Heterogeneity is established using both genotypic and phenotypic measures, and we show that genetic algorithms using genotypic measures out perform both the simple genetic algorithm and ones using a phenotypic measure.

1 Introduction

Genetic Algorithms have proved useful in a variety of search and optimisation problems. As their use advanced, certain operational problems were raised and addressed. Dominance and diploidy solved the problem of a constantly changing fitness function [GS87], whilst, to overcome limitations in fixed codings, inversion and reordering operators were introduced [Gol89].

All such approaches rely on one condition which, while self evident in nature, is generally not met within evolutionary computation. Efficient search requires a solution set, or population, sufficiently diverse to explore the space while benefitting from the information formed by the convergence of solutions. We do not accept the conventional view that stochastic errors in the genetic operators cause premature convergence [Bak87]. Instead, we consider the mechanisms found in nature, where such populations are the norm.

In nature, stable subpopulations, or demes, emerge around niches. Demes appear to grant evolutionary advantage. The Eurasian Crow subspecies (Corvus Corone Corone and C. c. comix) freely hybridise, but maintain distinct genetic identities [May70].

Darwin [Dar59] alluded to mechanisms by which species originate, but his thesis, The Origin of Species by Means of Natural Selection, might be better titled The Adaptation of Species. It proposed how diversity would decline as fitter individuals survived to dominate. It does not accommodate the effect that a population’s genetic structure has upon evolution.

"...the classical theory of genetics could find no reason for so much genetic variability: superior mutations would be incorporated into the
genotype of the species while inferior ones would be eliminated. As a result the uniformity of the species would be maintained and genetic variation kept at a minimum. But since the high variability of natural populations is an observed fact, it is evident that there must be some basic flaws in the assumptions of classical theory” [May70]

Fisher's fundamental theory of natural selection [Fis58] quantified the evolutionary process: “the rate of increase in fitness of any organism at any time is equal to its genetic variance in fitness at that time”. This exposed Darwinism’s major problem,

“If we believe that species arise by the transformation of variation within populations into variations between populations, we must explain the renewal of variation” [Lew71]

Fisher and his contemporary Wright [Wri78] held mutation responsible for introducing variation. Selection removed deleterious mutations [MS96]. They differed over how beneficial mutations fixated. Fisher [She58] postulated evolution to occur in randomly mating, panmictic populations, with the fitness landscape in constant flux so ensuring the population never fixated on local optima. Against this, Wright saw many small, differentiated populations. In small populations beneficial mutations fixate easier. In large ones selection tends to remove them. Paradoxically, small populations benefit deleterious mutants. To cross valleys on the fitness surface, occasionally a deleterious mutant must approach fixation to form a route between optima. Smaller populations favour their survival, since large populations will select against them. So, small populations favour faster evolution rates.

2 Multiple Deme GA Models

Crow [Cro86] emphasises population structure by introducing the neighborhood [MS89, Muh89] and island models [LTTP95, CHMR87, Tan87, SWM91]. In the island model, selection occurs within the deme. Migration moves individuals between demes. With the neighborhood model, selection occurs within demes formed from geographically close neighbors. Migration occurs by overlapping the neighborhoods. Both have shown advantages over the panmictic simple genetic algorithm (SGA) of Holland [Hol92], beyond those expected from a mere parallelisation of the SGA.

Mühlenbein’s [Muh89] grid model, based on Crow’s neighborhood model, extends the SGA in two ways. Individuals are spatially separated on a grid, and hill climb to their local optima. This ensures breeding between locally optimal individuals. Secondly, selection is carried out within the local neighborhood. A distance metric defines locality, allowing neighborhoods to overlap. Information may therefore flow between neighborhoods, over time.

The island model, as described by Loraschi [LTTP95], divides the population into demes. Each deme evolves separately using a SGA. Fixed connections
between demes allow individuals, and hence information, to explicitly migrate between demes. In the grid model this is implicit.

These models improve the SGA, but both are artificially constrained. They enforce properties which, in nature, evolve, eg migration rates, deme size and number. Neither model clusters similar phenotypes, and whether diverse phenotypes come to dominate each deme depends on genetic drift [Mah94].

3 Adaptive Multiple Deme Models

This lack of plasticity in the island and grid models yields rapid evolution, but to suboptimal solutions [GS89]. A "softer" approach is necessary.

Goldberg’s sharing function [DG89] recognises that like individuals compete for resources. The probability of selection becomes a function of the average fitness and number of similar individuals. Similarity depends upon either the Hamming distance in genotypic space, or Euclidean distance in phenotypic space. However, the use of the Hamming distance is debatable. To correlate it to the difference in phenotypic fitness, it must measure nonlinearities between chromosomes. This it cannot do. Chromosomes equally separated may have grossly different phenotypes.

Goldberg showed that phenotypic sharing demonstrated better multimodal optimisation than did genotypic.

Sharing still enforces constraints. The number of clusters is set a priori so that the similarity criteria do not emerge from the interaction of selection and reproduction [Dav91].

Yin [YG93] addresses this need for a priori knowledge, by adapting MacQueen’s KMEAN algorithm [Web99] to evolve the number of clusters. Each individual has a vector of numeric properties. In addition, a centroid $G(C_c)$ typifies each cluster $C_c$ as

$$G(C_c) = (\bar{x}_{c1}, \bar{x}_{c2}, \ldots, \bar{x}_{cp}) \quad c = 1 \ldots k$$

with

$$\bar{x}_{cj} = \frac{1}{n_c} \sum_{i=1}^{n_c} x_{ij}$$

where

$p = $ number of properties in a vector

$k = $ the number of clusters

$n_c = $ number of individuals in cluster $C_c$

$\bar{x}_{cj} = $ mean of property $j$ for cluster $c$

$x_{ij} = $ value of property $j$ for individual $i$

$x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is the vector for individual $i$

The algorithm sorts $N$ individuals into $k$ clusters as follows

1. Take $k$ individuals as initial seed points
2. Assign each other individual to the cluster with the nearest centroid, recomputing the centroid after each assignment
3. Take the computed centroids as new seed points and reassign the individuals to their now nearest centroid
Yin adapts this using two parameters, \( d_{\text{min}} \) and \( d_{\text{max}} \). If an individual's nearest centroid is further than \( d_{\text{max}} \) it forms a new cluster. Clusters then merge until no clusters are closer than \( d_{\text{min}} \).

Each individual's fitness reduces by a factor inversely proportional to the cluster's size and the distance to its centroid. The effect, in theory, is to reduce the pressure on a population to converge by reducing fitness in heavily populated areas of the search space.

Yin demonstrates multimodal optimisation with a set of one dimensional, sinusoidal functions in both phenotypic and genotypic space. However, they and the chromosomal coding used have a low degree of non-linearity. Yin [YG91] reports limited success with a real-life example.

Yin's conclusions present a number of questions. Yin used a zero mutation rate. The effect of a non zero rate is unknown. One of the effects of clustering is to allow mutants outside a cluster to achieve a higher fitness than clustered individuals. In nature, hybridisation and mutation play a major part in evolution, but here they are absent. Why are clusters around separate peaks not hybridising and yielding poor unfit siblings so destabilising the clusters? The answer appears to be that the chromosomal coding is biased towards stability. Since Yin uses a binary encoding the early genes are most significant in determining the value of \( x \), where \( F(x) \) is the fitness. Crossover will replace later order bits of low value with a sequence of similar value. Hence, crossover makes little difference.

In the following sections we examine the dynamic behavior of clustering genetic algorithms including mutation, on a real-world problem in which the encoding gives greater significance to crossover.

4 A Real Life Example

An analysis of Yin's test functions shows that to examine the dynamics of clustering any example must have certain properties, so as not to bias clustering. The fitness function should be nonlinear. The chromosomal encoding should enhance this, so that crossover may move siblings between clusters.

Our chosen testbed is Modern Portfolio Optimisation [Mar59]. The intention is to maximise the return for a portfolio of financial stocks having a set risk. We modify Yin's algorithm to scale fitness down by the cluster size alone.

We model the portfolio as \( N \) integers, each encoded using 10 bits. If the required risk is \( \sigma_r \), the expected return and risk \( R_p \) and \( \sigma_p \) we define fitness as

\[
F(p) = R_p - \alpha(\sigma_r - \sigma_p)^2
\]

The factor \( \alpha \) is an arbitrary constant biasing the fitness function towards high \( R_p \) or small \((\sigma_r - \sigma_p)\).

Phenotypically each chromosome expresses a ratio. There then exist portfolios with equal expressions, e.g. 1,2,3 and 4,8,12 are equivalent. To separate such portfolios we add a factor \( L \), the longest sequence of leading zeroes in any of the binary integers, so that

\[
F(p) = (R_p - \alpha(\sigma_r - \sigma_p)^2)/(L + 1)
\]
Any chromosome with $L \neq 0$, has an equivalent with $L = 0$. This biases selection towards a single model of each portfolio.

This example shows the necessary properties of non-linearity and unbiased encoding.

4.1 Experimental Results

We now consider the dynamics of clustering as $d_{\text{max}}$ and $d_{\text{min}}$ vary, where $d_{\text{max}}$ and $d_{\text{min}}$ are as defined above.

Both $d_{\text{max}}$ and $d_{\text{min}}$ have a lower limit of 0. In genotypic space, the maximum possible limit $\delta$ is that between a string of $n$ zeroes and a string of $n$ ones, so $\delta = \sqrt{n}$.

The phenotypic space, of normalised vectors $V$ such that

$$\sum_{i=1}^{n} V_i = 1.0, \text{ with } 0.0 \leq V_i \leq 1.0$$

defines a section of a hyperplane of dimension $n$. The maximum distance between two points on that surface is then from one corner to another, so $\delta = \sqrt{2}$.

If $d_{\text{max}} = d_{\text{min}} = 0$ only identical individuals will cluster, and initially the clusters are likely to contain single individuals. This approximates an SGA. At the alternative extreme, $d_{\text{max}} = d_{\text{min}} = \delta$, all individuals belong to one cluster, and again approximates an SGA.

As our example we take a portfolio of six stocks, yielding chromosomes of 60 bits. The required risk $\sigma_r$ is fixed at 25%. The population size is 100, with crossover probability set to 0.6 and the mutation probability to 0.001. Each generation replaces 60% of the previous.

We use two fitness measures in our analysis. The basic fitness of the population is the highest individual fitness $F(p)$, not scaled down by the cluster size. The alternative scaled fitness of the population selects the highest scaled fitness and then scales it back up by its associated cluster size. Individuals are ranked by scaled down fitness for both selection and replacement. This has the corollary that if no new individuals with higher $F(p)$ occur in a generation, it is possible for the ranking to change due to changes in relative frequency. We therefore expect to see cyclical behavior. A highly fit chromosome, of which only one instance exists, will breed rapidly in one generation. In the next generation its scaled fitness will drop drastically. So by the third generation it will again be in the minority, and the cycle will restart.

We present the results of each run at termination as; number of clusters, basic fitness, and scaled fitness. One run is made for each combination of $d_{\text{max}}$ and $d_{\text{min}}$ varying in steps of $\delta/10$ from 0 to $\delta$, using phenotypic and genotypic clustering. Each run terminates after one hundred generations, or after four successive single cluster generations which we consider to indicate convergence. For reasons of space, we tabulate the fitness values in intervals of $\delta/5$.

Phenotypic clustering (Table 1) shows that for values of $d_{\text{min}}$ and $d_{\text{max}}$ up to 20% of $\delta$, diversity is maintained. The number of clusters degrades as the gaps
increase, since the clusters can cover a larger area. The effect is most significant for $d_{\min} = d_{\max} = 0$. Here, only identical individuals cluster. For selection, such a population is identical to one created by taking one individual from each cluster. This removes imbalances in the population. If two individuals have fitness in ratio $\alpha$ to $\beta$ then, regardless of their proportions in the population, selection will occur as if the population was in perfect balance of $\alpha$ to $\beta$. In such populations mutants offering only a minute advantage may spread with much greater ease.

The effect of clustering on the two fitness measures is shown in Table 2 by the cluster of low scaled values, at $d_{\min}, d_{\max} = \{0, 1\}, \{1, 0\}$ and $\{1, 1\}$.

<table>
<thead>
<tr>
<th>$d_{\min}$</th>
<th>$d_{\max}$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35.68</td>
<td>29.31</td>
</tr>
<tr>
<td>1</td>
<td>32.38</td>
<td>28.76</td>
</tr>
<tr>
<td>2</td>
<td>30.37</td>
<td>30.15</td>
</tr>
<tr>
<td>3</td>
<td>30.53</td>
<td>30.04</td>
</tr>
<tr>
<td>4</td>
<td>31.36</td>
<td>30.57</td>
</tr>
</tbody>
</table>

The presence of such low values indicates the presence of relatively large clusters of much fitter individuals.

For both scaled and basic measures in Table 2, fitness generally declines with increased gap. This effect is due to the decrease in the number of generations before the run terminated.

Yin [YG93] and Goldberg [DG89] claim phenotypic clustering produces better results. We have alluded to reasons for this, and our results show genotypic clustering is more able to maintain diversity. Table 3 shows diversity is main-
tained with gaps up to 40% of δ. The decline in the number of clusters is slower than in phenotypic, climbing from 81 to 92 before declining.

Table 3. Genotypic Clustering: # Clusters at Termination

<table>
<thead>
<tr>
<th>0</th>
<th>← d_{min} →</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81 90 60 65 41 8 2 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>92 90 80 73 48 11 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑</td>
<td>70 65 66 54 29 8 2 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>59 71 68 75 38 9 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42 34 27 33 36 1 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d_{max}</td>
<td>1 3 1 3 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>↓</td>
<td>1 1 1 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>δ</td>
<td>1 1 1 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Genotypic Fitness at Termination

<table>
<thead>
<tr>
<th>0</th>
<th>← d_{min} →</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34.60 29.85 28.99 33.02 30.00 31.13</td>
<td></td>
</tr>
<tr>
<td>↑</td>
<td>33.90 34.67 32.04 34.02 32.31 31.24</td>
<td></td>
</tr>
<tr>
<td>25.96 30.77 30.29 33.06 28.90 30.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d_{max}</td>
<td>34.05 32.17 31.47 30.12 31.69 31.34</td>
<td></td>
</tr>
<tr>
<td>↓</td>
<td>28.40 28.52 30.77 28.26 27.53 31.15</td>
<td></td>
</tr>
<tr>
<td>δ</td>
<td>32.27 27.82 31.18 30.21 30.55 28.33</td>
<td></td>
</tr>
</tbody>
</table>

(a) Scaled
(b) Basic

Tables 4(a) and 2(a) show similar behaviour: clusters of around 40 individuals at termination correlate with low scaled fitness values. Table 4(b) show little difference from Table 2(b). Without much more experimental data we consider that the differences are statistically insignificant.

We now consider an SGA by counting how many clusters occur in each generation, but without scaling the fitness. We contrast this to the clustered GA using identical d_{max} and d_{min}.

The simple genetic algorithm (SGA) is known to lose diversity rapidly. Here we measure diversity as the number of clusters present in the population, using d_{min} = d_{max} = δ/10. These values demonstrated clustering in both the genotypic and phenotypic examples previously given.

Figure 1 shows that clustering, of either type, maintains diversity. The SGA decreases almost logarithmically in diversity. The SGA shows the most rapid growth in fitness, see Fig. 2. Genotypic clustering behaves in a similar manner
but takes longer to reach a near optimal value. However, it does briefly find a higher fitness than the SGA, possibly indicating the SGA had insufficient diversity to continue the search. Certainly the population was more diverse with genotypic clustering once near optimal values had been reached.

With phenotypic clustering, fitness appears cyclic and chaotic. Fitness grows slower than with the SGA or genotypic clustering. This accords with our previous prediction. However, we note that genotypic clustering shows no evidence of cyclic variation. The large number of clusters with genotypic clustering implies that most clusters are unitary, so very few duplicate individuals exist, in which case we do not expect to see cyclical behavior.

If we move to consider basic fitness (see Fig. 3) we note a change. The growth of clustering with the SGA and genotypic forms is similar. Phenotypic clustering now shows a much more stepped growth than before. We hypothesise that this is due to the constrained nature of the search space, where each vector must sum to unity. If in the genotypic space individuals are randomly distributed, then projecting the same space into the phenotypic space causes individuals to conglomerate. This means that a phenotypic space is inherently less diverse than a genotypic one.
Fig. 3. Basic Fitness Comparison

5 Conclusions

By the use of adaptive clustering to moderate fitness in a single homogeneous population, we have combined the islands of Wright’s model with Fisher’s constantly varying fitness landscape.

Previous test cases implied phenotypic clustering produced better results, compared to genotypic clustering, as measured by the number of optima. We have shown that this may be due to bias in their chromosomal encoding. By reducing such bias, and applying clustering to a larger more complex encoding, we have shown both genotypic and phenotypic clustering can maintain a highly diverse population. Compared to the SGA a genotypic algorithm closely matched its growth in fitness. Yet unlike the SGA it maintained high diversity.

References


Symbiotic Combination as an Alternative to Sexual Recombination in Genetic Algorithms

Richard A. Watson  Jordan B. Pollack

Dynamical and Evolutionary Machine Organization
Volcan Center for Complex Systems – Brandeis University – Waltham, MA – USA
richardw@cs.brandeis.edu

Abstract. Recombination in the Genetic Algorithm (GA) is supposed to enable the component characteristics from two parents to be extracted and then reassembled in different combinations – hopefully producing an offspring that has the good characteristics of both parents. However, this can only work if it is possible to identify which parts of each parent should be extracted. Crossover in the standard GA takes subsets of genes that are adjacent on the genome. Other variations of the GA propose more sophisticated methods for identifying good subsets of genes within an individual. Our approach is different; rather than devising methods to enable successful extraction of gene-subsets from parents, we utilize variable-size individuals which represent subsets of genes from the outset. Joining together two individuals, creating an ‘offspring’ that is twice the size, straightforwardly produces the sum of the parents’ characteristics. This form of component assembly is more closely analogous to combination of symbiotic organisms than it is to sexual recombination. Whereas sexual recombination, modeled by crossover, occurs between similar individuals and exchanges subsets of genes, symbiotic combination, as modeled in our operator, can occur between entirely unrelated species and combines together whole organisms. This paper summarizes our research on this approach to recombination in GAs and describes new methods that illustrate its potential.

1 Introduction

The natural process of symbiogenesis [Merezhkovsky 1909] is the creation of new species from the genetic integration of symbionts, organisms in symbiotic relationship. Symbiogenesis has enabled some of the major transitions in evolution [Maynard-Smith and Szathmary 1995], including the origin of eukaryotes which include all plants and animals. This kind of genetic integration is quite different from the transfer of genetic information in sexual reproduction. Sexual recombination occurs between similar organisms (i.e. of the same species) and involves the exchange of parts of the genome in a mutually exclusive manner. That is, every gene acquired from one parent is a gene that cannot be acquired from the other parent. In contrast, symbiotic combination can occur between genetically unrelated organisms (i.e. different species) and can involve the integration of whole genomes. The resultant composite may have all the genes from one organism and at the same time acquire any number of genes from the second organism.

Our research has been investigating whether this form of genetic combination has any potential to inspire an alternate form of evolutionary algorithm. This has first required us to address some old arguments about the utility of ordinary crossover in the standard GA
To this end, we developed a hierarchical building-block problem that exemplifies the utility of sexual recombination as demonstrated by crossover [Watson et al 1998, Watson & Pollack 1999b, Watson 2000]. Moreover, our experiments also illustrated a well-known limitation of crossover — that is, it relies on the heuristic of bit adjacency to identify appropriate subsets of genes during recombination. Clearly, we cannot in general rely on this heuristic, and when the adjacency of the genes is not correlated with gene interdependency, the subset of genes extracted from a parent by crossover is unlikely to contain a meaningful component of the parents' abilities [Altenberg 1995]. Figure 1 illustrates the problem of crossover in a poor-linkage problem — i.e. where the bits of a block (a set of interdependent genes) are distributed.

\[
\begin{array}{c}
A: 01010011 \\
B: 10000110 \\
C: 11010011
\end{array}
\]

**Figure 1:** Sexual recombination (crossover). Parents, A and B, each contain a useful subset of genes that are distributed on the genome (three genes per parent, shown in bold). The desired offspring, C, should take the good genes from both parents as shown. One-point or n-point crossover cannot do this. In principle, uniform crossover could provide the crossover that we need but the probability of this is equivalent to random guessing at each locus where parents disagree on allele values [Watson & Pollack 1999a].

Thus we see that building-blocks in the standard GA are, at best, only represented implicitly by virtue of the proximity of genes on the genome. In poor-linkage problems we need some explicit mechanism to represent the unity of bits comprising a block. This is where a mechanism based on symbiotic combination might be useful. Put simply, if each organism represents just one building-block, then a combination operator can put different organisms together straightforwardly producing an organism twice the size with both building-blocks intact. A simple representation for an individual in this scheme utilizes 'unspecified' genes — then symbiotic combination can be described as a genetic operator acting on partially-specified individuals (Figure 2).

\[
\begin{array}{c}
A: \text{---0-1} \\
B: 1\text{-0-0-} \\
C: 11\text{0-00-1}
\end{array} \quad \begin{array}{c}
A: \text{-1--0-1} \\
B: \text{100-0--0} \\
C: \text{110-00-1}
\end{array}
\]

**Figure 2:** 'Symbiotic combination'. Left) Combination of partially-specified individuals produces an offspring that is twice the size of the parents with the sum of their characteristics. Here we represent unspecified genes, or don't cares, by "-". Each parent represents a single building-block explicitly and the offspring is created by taking specified genes from either parent where available. Right) Where conflicts in specified genes occur we resolve all conflicts in favour of one parent, e.g. the first parent.

Previous work [Watson & Pollack 1999a, 2000a], implemented the 'Incremental Commitment GA', ICGA, using this operator. The ICGA can be described as a simplified and generalized version of the Messy GA [Goldberg et al 1989], which also uses combination of partially specified individuals. But our implementation of the ICGA had important limitations, as we shall discuss. In this paper we introduce a much richer model of symbiotic combination: The Symbiogenic Evolutionary Adaptation Model, SEAM. In addition to symbiotic combination, as used in the ICGA, the key features of SEAM are group evaluation of individuals which removes the need for partial evaluation, and a
'Pareto coevolution' method that segregates competition to maintain diversity and prevent large sub-optimal individuals from replacing small optimal individuals. These new features overcome the limitations of the earlier ICGA and facilitate the application of symbiotic combination more effectively.

The remainder of this paper is organized as follows: Section 2 summarizes relevant background work. Section 3 details our Symbiogenic Evolutionary Adaptation Model. Section 4 gives results for the standard GA, ICGA and SEAM on the hierarchical building-block problem with random linkage. Section 5 concludes.

2 Background: Approaches to Building-Block Combination

Our previous work describes a formal building-block problem that exemplifies the class of problems for which recombinative algorithms like the GA are well-suited. This problem, which we call Hierarchical If-and-Only-If (HIFF), was first introduced in [Watson et al 1998] (Equation 1). This function interprets a string as a binary tree and recursively decomposes the string into left and right halves. Each block at each level in the hierarchy has two solutions – all ones and all zeros – and the function has two corresponding global optima. Although each building-block is identifiable via its fitness contribution, a successful algorithm must maintain competing solutions for each block and search combinations of blocks to find complete solutions. This discovery and combination process continues through a hierarchical structure which is consistent in the nature of the problem at each level [Watson and Pollack 1999b]. On this class of problem, mutation based algorithms cannot be guaranteed to succeed in less than time exponential in N (the size of the problem in bits), whereas an idealized recombinative algorithm has an upper bound in time to solution of order $N \lg^2 N$ [Watson 2000]. However, Equation 1 states the fitness of a string assuming tight linkage i.e. two bits that form a block at the lowest level are adjacent on the genome, and two size-2 blocks that form a block at the next level are also adjacent, and so on. The success of the regular GA is dependent on tight genetic linkage – and the focus of our ongoing research is on solving the problem with randomized linkage.

$$F(B) = \begin{cases} 
1, & \text{if } |B|=1 \text{ and } (b_i=0 \text{ or } b_i=1) \\
|B| + F(B_L) + F(B_R), & \text{if } (|B|>1) \text{ and } (\forall i(b_i=0) \text{ or } \forall i(b_i=1)) \\
F(B_L) + F(B_R), & \text{otherwise.} 
\end{cases}$$

Eq.1

where $B$ is a block of bits, $\{b_1, b_2, ..., b_k\}$, $|B|$ is the size of the block=$k$, $b_i$ is the ith element of $B$, $B_L$ and $B_R$ are the left and right halves of $B$ (i.e. $B_L=\{b_1, ..., b_{|B|/2}\}$, $B_R=\{b_{|B|/2+1}, ..., b_k\}$). The length of the string evaluated must equal $2^p$ where $p$ is an integer (the number of hierarchical levels). Notice that this function gives no reward to nulls and therefore naturally evaluates partially specified strings.

In [Watson & Pollack 1999a] we started investigation into solving Shuffled HIFF, (SHIFF) where the position of bits in the problem is randomly re-ordered. One approach to address problems of poor linkage is seen in the Messy GA [Goldberg et al 1989] and the Linkage Learning GA [Harik & Goldberg 1996] which use a moving locus representation of genes – each gene is represented by a locus/allele pair. This enables genes to be re-ordered on the genome and potentially allows interdependent genes to collect together. However, there is one feature of the Messy GA that is quite simple, and
potentially effective, yet is generally under-emphasized. This is the feature of 
underspecification -- individuals that specify only a subset of genes.

Fully-specified individuals, as used in the standard GA, may contain good building-
blocks but selection acting on these individuals will also promote garbage genes riding on 
the same string (see “parasites” [Goldberg et al 1989], and “hitch-hikers” [Forrest & 
Mitchell 1993], [Vekaria & Clack 1999]). Consequently, a crossover operator using fully-
specified individuals must, one way or another, express which subsets of genes represent 
good schemata to exclude garbage genes from recombination events. The interesting 
property of approaches using partially-specified individuals is that individuals represent 
schemata explicitly; and it is the normal operation of selection in the GA, operating on 
these sub-strings, that permits the successful identification of good building-blocks.

In the problem class we address, large building-blocks at higher levels of the hierarchy 
produce significant high-order interdependencies. For a multi-level building-block 
problem like this we must use a method of gradually increasing specification that allows 
blocks to be accumulated through many incremental stages (unlike the two-phase method 
of underspecification seen in the Messy GA). To enable this, previous work employed a 
size-penalty augmentation to the fitness function. This enables the size of strings to grow 
gradually, only committing to gene alleles when these genes return significant fitness 
contributions, hence, Incremental Commitment GA [Watson & Pollack 1999a]. Using the 
ICGA we showed that the feature of underspecification is sufficient to enable success on 
a poor-linkage problem and that the other features of the Messy GA, in particular the 
moving-locus features which are often the focus of related work, are not required.

However, our ICGA was not without its own complications. The first, is that the 
approach requires a diversity maintenance technique to keep the population from 
converging. As in earlier work with the standard GA, we used a resource-based fitness 
sharing method that utilized considerable knowledge of the problem structure. 
Specifically, it maintained a ‘resource level’ for each building-block in the problem. 
Since then we have found an off-the-shelf diversity maintenance technique, deterministic 
crowding [Mahfoud 1995], that works very well for our problem [Watson & Pollack 
2000a]. Deterministic crowding, DC, is naturally implemented in a steady state GA as 
described in Figure 3.

- Initialize population.
- Repeat until stopping condition:
  - Pick two parents at random from the population, p1 & p2.
  - Produce a pair of offspring using recombination, c1 & c2.
  - Pair-up each offspring with one parent according to the pairing rule below.
  - For each parent/offspring pair, if the offspring is fitter than the parent then 
    replace the parent with the offspring.

  **Pairing rule:** if H(p1,c1)+H(p2,c2) < H(p1,c2)+H(p2,c1) then pair p1 with c1, and 
p2 with c2, else pair p1 with c2, and p2 with c1, where H gives the genotypic 
Hamming distance between two individuals.

**Figure 3:** Pseudo-code for a simple form of a GA using deterministic crowding.

The most important feature of DC is that offspring only compete with their own 
parents. Additionally, offspring and potential replaces are paired to maximize their 
similarity. This segregated competition introduces tolerance for lower-fitness individuals 
in different niches and reduces the pressure for convergence. Note that DC does not
segregate mating like other diversity maintenance techniques, e.g. ‘thresholding’ [Goldberg et al 1989]. In DC, an individual may mate with any other regardless of their similarity or dissimilarity, but it only competes with similar individuals.

The advantage of DC with respect to our previous method of diversity maintenance is that it does not require any knowledge of the problem’s building-block structure. It also alleviates complications in the size-penalty that arose from the distorted fitness values given by the fitness sharing mechanism. However, the ICGA still needs a size-penalty that requires knowledge of how fitness is expected to grow with string length. Moreover, the ICGA also depends on being able to evaluate partially specified strings. These are handled quite naturally in our test problem but, in general, an objective function may not be able to evaluate a string that is not fully-specified.

Goldberg et al [1989] suggest that one way to overcome the need for partial evaluation is to use competitive templates, bit-strings that are used to fill-in the unspecified bits of an individual. The template provides a context in which the partial individual can be evaluated. However, Goldberg et al also show that the use of random strings as templates would produce too much ‘background noise’ to identify the relatively small fitness contribution of a low-order schema represented by an individual. And they also argue that the use of a single random template used for all evaluations (reducing the influence of random noise) cannot evaluate a schema in an appropriately diverse range of contexts to assess its proper value. They propose that one way forward is to use a ‘locally optimized’ template provided by some other search method. However, this approach assumes that the “highest order non-linearity expected in the problem” is bounded and, in fact, quite low-order, as Goldberg et al assume. In this case, an appropriately optimized template is quite easy to find. But, in problems with strong high-order interdependencies, like the class we address, the task of providing an appropriately optimized template is only one step easier than solving the whole problem. Nonetheless, a form of templating will be useful in solving the problem of partial evaluation in our new algorithm, SEAM.

3 The Symbiogenic Evolutionary Adaptation Model, SEAM

This section introduces the ‘Symbiogenic Evolutionary Adaptation Model’, SEAM, which utilizes three main ideas. First, as in the ICGA, SEAM uses symbiotic combination that combines whole small organisms, rather than sexual recombination that recombines parts of fully-specified organisms. Second, we use groups of other individuals from the population to provide the templates and preclude the need for partial evaluation. And the third new component of SEAM is to use what we call ‘Pareto coevolution’ which segregates competition to maintain diversity, and prevents large sub-optimal strings from replacing small optimal strings (removing the need for a size-penalty).

Group templating

The use of other organisms to provide templates is inspired by the co-adaptation of symbiotic organisms in an ecosystem. We think of the templates as different environmental contexts, or niches, provided by different combinations of neighboring organisms. Algorithmically, the templates test the performance of a given schema in the context of many other different schemata provided by other optimized individuals. In this way we do not need to use a different search technique to provide the templates as Goldberg et al propose – rather, the organisms that are used as templates, and the
organisms that use the templates, are all created by the same unified process. As more fit
large individuals are discovered by the algorithm, they provide better templates for
discovering individuals for the next hierarchical level in the problem. This use of
individuals that are co-adapted to fill-in for one-another provides effective templates and
precludes the need for partial evaluation in the algorithm. Group evaluation is used in a
couple of existing algorithms, [Moriarty 1997, Potter 1997], and we have also
investigated related effects of group evaluation ourselves [Watson & Pollack 1999c]. But
the technique has not been connected to the use of templates in the Messy GA, nor has it
been integrated with genetic operators that combine organisms together permanently.

Naturally, this method of templating will return different fitness scores depending on
which individuals are chosen for the template. As Goldberg et al caution, an accurate
measure of fitness for the individual in question might require prohibitively many trials.
To alleviate the background noise of a template, individuals in SEAM are assessed in
pair-wise competitions. That is, two individuals, A and B, are evaluated using the same
additional individuals to provide the template/context for evaluation (Figure 4).

\[
\begin{align*}
A: & \ 11----00 & B: & \ -1010-- \\
1: & \ -0--1-0- & 1: & \ -0--1-0- \\
2: & \ 10-0-11- & 2: & \ 10-0-11- \\
3: & \ 1-1--1- & 3: & \ 1-1--1- \\
4: & \ --0-01-1 & 4: & \ --0-01-1 \\
\end{align*}
\]

\[
A':11101100 B':10101001
\]

Figure 4: Left) A given partially-specified individual, A, is evaluated by building a template from
several other partially-specified individuals, 1 through 4. Specified genes are provided by A where
available, and unspecified positions are filled-in with genes from 1, and so on through 4, using
additional individuals until all genes are specified. The resultant string A', is the string evaluated
for A. Right) The same individuals 1 through 4 are used to evaluate a second individual B. The
difference in fitness between A' and B' indicates which is better in this context.

Note that any given context may favor A more than B, for example, depending on
whether A happens to be better adapted to that particular context or not. So we will still
need to perform evaluations in many different contexts to determine the superiority of A
and B. However, this will not require prohibitively many evaluations.

**Pareto coevolution**

The group evaluation used in templating makes SEAM a coevolutionary system – the
task of being a successful organism is dependent on the composition of the population.
The normal coevolutionary procedure is to average performance over many trials, in this
case, many different contexts. However, preliminary investigations using averaged scores
caused convergence and failure of the algorithm. This problem has prevented us from
progressing our model of symbiotic combination with group templating for some time.
Here we introduce a new method of coevolution that we can use in SEAM to overcome
this problem. This new method, which we call 'Pareto coevolution', incorporates ideas
from Pareto optimization methods that are well-established for optimization in problems
with multiple objectives [e.g. Horn 1997]. Pareto optimization recognizes that
performance over different objectives, say 'financial cost' and 'construction time', cannot
be combined to give an overall performance unless we know how to convert one
'dimension' to another – in this case, we need to know what our time is worth. Pareto optimization techniques may be used when the relative weighting of different dimensions is not known. Specifically, Pareto optimization is built on the principle of Pareto domination. A solution is said to Pareto dominate another solution if it is superior or equal in all dimensions (and superior in at least one dimension).

The idea behind Pareto coevolution is to use different coevolving opponents as the dimensions for determining dominance. Specifically, an individual dominates some other individual if it performs no worse than that individual against each and every opponent. In this manner, the performance of an individual will be assessed on the basis of which particular opponents it does well against and not just an average score. This allows individuals to adapt to different sets of opponents and promotes diversity.

Pareto coevolution is simply the application of this form of dominance in any Pareto optimization technique. We suggest that Pareto coevolution may be valuable in a variety of existing coevolutionary games and with a variety of Pareto optimization methods. But in the next subsection we show how to apply the technique to our specific domain of function optimization via group evaluation, and how to implement a very simple form of Pareto optimizer that is sufficient for our needs.

Integrating the features of SEAM

In SEAM the coevolution is subtle – the result of an evaluation depends on the other members of the population selected for the template – but there is no overt opponent in this setup. Nonetheless, we can use contextual groups as dimensions to determine dominance and apply Pareto coevolution. SEAM uses this principle directly in determining the outcome of the pair-wise competition illustrated in Figure 4. Specifically, $A$ dominates $B$, if it is superior or equal to $B$ in all groups tested. Figure 5 shows how to utilize this rule in a very simple Pareto optimizer.

- Initialize population to random single-bit individuals.
- Repeat until stopping condition:
  - Pick two parents at random from the population.
  - Produce an offspring using symbiotic combination.
  - If the offspring dominates both parents (see below) then replace the parents with the offspring.

To determine whether $A$ dominates $B$: Repeat for $t$ trials:
- Build a complete template from randomly selected individuals.
- If $B$ superimposed on this template receives a higher score than $A$ superimposed on this template then $A$ does not dominate $B$.

Figure 5: Pseudo-code for a simple implementation of SEAM.

Segregating competition by the use of contextual niches in SEAM maintains appropriate diversity without the need to use the genotypic similarity metric usually used in the pairing-rule of deterministic crowding. More importantly, these methods preclude the need to use a size-penalty function. The size-penalty was used in the ICGA to prevent organisms from prematurely filling-up with incompatible blocks. In SEAM, if just one of the contexts tested for the parent includes a compatible block, then an offspring formed by joining with an incompatible block in that position will be rejected. By insisting that a composite must perform as well as the parents in all contexts, we prevent incompatible blocks from being joined, and individuals being filled with sub-optimal schemata.
The algorithm in Figure 5 differs considerably from established Pareto optimization methods: specifically, like deterministic crowding, an offspring only survives through competition with its parents. This restriction is applied for efficiency – the offspring is intended to combine the characteristics of the parents so, if this combination is successful, the offspring should be at least as good as either parent in any context. This method proves sufficient for our problem class but preliminary investigations suggest that a more conventional Pareto optimization approach may broaden the applicability of SEAM. Additionally, our implementation of this algorithm also reduces computational expense by removing duplicates from the population. So, when we replace the parents we insert only one copy of the offspring, reducing the size of the population by one with each offspring that is successful. At present, duplicates in the initial population are identified by genetic comparison but, in principle, we can test for identity in their contextual performance.

4 Experiments

This section gives experimental results of the GA, ICGA, and SEAM applied to a 64-bit Shuffled HIFF. The GA is implemented using the deterministic crowding algorithm of Figure 3, and is tested using uniform crossover (GA-uniform) and one-point crossover (GA-onepoint). A population size of 1000 is used; crossover is applied with probability 0.7; and mutation is applied with 0.03 probability of assigning a new random allele (0 or 1, with equal probability). The ICGA differs from the GA in three respects: 1) it uses partially specified individuals, initialized to one random bit, and mutation assigns 0, 1, or null, with equal probability; 2) it uses the combination operator described in Figure 2, and; 3) it uses a size-penalty augmentation to the fitness function. In HIFF, the maximum fitness, MF, of a string of size $N$, is the product of $N$ and the number of hierarchical levels in the string, i.e. $MF(N)=N(\log_2N+1)$. Accordingly, individuals in the ICGA receive fitness $F'(B)=F(B)-MF(1|B|)$. SEAM uses the algorithm outlined in Figure 5. A population size of 1000 is used in initializing SEAM but the removal of duplicates quickly reduces this to (approximately) the 128 unique individuals (for a 64-bit problem). Symbiotic combination (Figure 2) is applied always, no mutation is required. The number of trials, $t$, used in testing the dominance of two individuals is at most 50, but most tests fail in less than 10 trials. Performance in Figure 6 is measured by the fitness of the best string evaluated (in the preceding 2000 evaluations) averaged over 30 runs for each algorithm. The problem size of 64 bits gives a maximum fitness of 448.

We see that the regular GA, using either crossover operator, tends to converge on sub-optimal solutions. The disruption caused by uniform crossover [Watson & Pollack 2000b] makes it worse than one-point crossover at first, but ultimately allows exploration that outperforms one-point. Actually, uniform crossover succeeds in 16 of the 30 runs, which is better than expected [Watson & Pollack 2000a], but those that do succeed take about 1,200,000 evaluations to do so. The ICGA is very slow to start because, unlike the GA, it must discover building-blocks explicitly – one per individual. But, eventually the ICGA shows that this method of partial-specification and symbiotic combination does allow successful combination of building-blocks in poor-linkage problems (about 1,700,000 evaluations permits 100% success). However, as noted, the ICGA uses a problem-specific size-penalty and partial-evaluation to achieve this. In contrast, SEAM performs very rapidly and successfully without using a size-penalty or partial evaluation. Group templating and Pareto coevolution introduced in SEAM prove to be very effective.
at enabling effective symbiotic combination. Control experiments, not shown, confirm that the use of either random templates instead of group evaluation, or replacement based on superior average performance instead of Pareto dominance, both cause the algorithm to fail. In either case, strings quickly fill with sub-optimal blocks and the combination operator is prevented from operating.

![Figure 6: Performance of regular GA (using one-point and uniform crossover), ICGA, and SEAM, on Shuffled HIFF.](image)

All algorithms except the GA-onepoint, perform the same on HIFF as on SHIFF, since they have no locus-dependent features. For reference, the GA-one point succeeds on regular HIFF, in all 30 runs, in less than 100,000 evaluations [Watson & Pollack 2000a]. Note that SEAM succeeds in solving SHIFF, a significantly harder problem, in a little over 200,000 evaluations.

5 Conclusions

To summarize, SEAM combines three new features with respect to a standard GA: 1) Partially-specified individuals and symbiotic combination (Figure 2) instead of sexual recombination. 2) Group evaluation (Figure 4) to provide contexts/templates that preclude the need to evaluate partially specified strings. 3) Pareto coevolution using different contexts to automatically define multiple dimensions for the problem space, thereby segregating competition to maintain diversity in the population, and prevent large sub-optimal individuals from replacing small optimal individuals. But although SEAM introduces several new concepts, it is algorithmically quite simple (Figure 5).

SEAM is the first known algorithm to solve Shuffled HIFF reliably. However, although SEAM is superior when applied to SHIFF, we have yet to compare SEAM's performance with that of other algorithms on different problem domains. In the meantime, our experiments illustrate some important principles in (re)combination methods. SEAM demonstrates that symbiotic combination of partially-specified individuals can provide a successful alternative to sexual recombination for building-block assembly in problems of poor genetic linkage.
Acknowledgments

The first author is indebted to Martin Oates for tireless investigations of diversity maintenance techniques, Sevan Ficici for stimulating discussion on coevolution and its connection to Pareto optimization, and other members of DEMO at Brandeis.

References


Moriarty, DE, 1997, Symbiotic Evolution of Neural Networks in Sequential Decision Tasks, PhD thesis, University of Texas at Austin, USA.


Co-evolution
Island Model Cooperating with Speciation for Multimodal Optimization

Mourad Bessaou\(^1\), Alain Pétrowski\(^2\), and Patrick Siarry\(^3\)

\(^1\) Université de Cergy-Pontoise, IUT Génie Electrique, Rue d’Eragny, Neuville-sur-Oise, 95031 Cergy-Pontoise, France

\texttt{Mourad.Bessaou@iutc.u-ergy.fr}

\(^2\) Institut National des Télécommunications, Département Informatique, 9 rue Charles Fourier, 91000 Evry, France

\texttt{Alain.Petrowski@int-evry.fr}

\(^3\) Université de Paris XII, Laboratoire d’Etude et de Recherche en Instrumentation Signaux et Systèmes, 61 avenue du Général de Gaulle, 94010 Créteil, France

\texttt{siarry@univ-paris12.fr}

Abstract. This paper considers a new method that enables a genetic algorithm (GA) to identify and maintain multiple optima of a multimodal function, by creating subpopulations within the niches defined by the multiple optima, thus warranting a good “diversity”. The algorithm is based on a splitting of the traditional GA into a sequence of two processes. Since the GA behavior is determined by the exploration / exploitation balance, during the first step (Exploration), the multipopulation genetic algorithm coupled with a speciation method detects the potential niches by classifying “similar” individuals in the same population. Once the niches are detected, the algorithm achieves an intensification (Exploitation), by allocating a separate portion of the search space to each population. These two steps are alternately performed at a given frequency. Empirical results obtained with F6 Schaffer’s function are then presented to show the reliability of the algorithm.

1 Introduction

Genetic Algorithms are stochastic optimization techniques that mimic the Darwinian evolution by modeling the natural selection process and the genetic modifications [1]. They act on a population of individuals that evolve under the effect of three basic operators: selection, crossover and mutation. The parents with high “fitness” survive and reproduce in order to create individuals again more adapted. In the case of standard unimodal GAs, the population quickly converges towards a promising zone of the search space.

However, this attractive feature can be not interesting for many problems. Indeed, in the case of multimodal functions, the algorithm risks to converge towards a sub-optimum : this feature is known as the premature convergence. It is mainly due to the loss of diversity in the population. In particular this issue is worrying in the case of some practical optimization problems, for which
one searches for a set of minima, instead of only one. This type of multimodal optimization especially occurs when the working out of the objective function requires a difficult modeling of the problem.

To overcome this problem, some diversity must be maintained during the generations in the population, by preventing its individuals of being "clones" of the current best one. The speciation/niching techniques aim at maintaining an appropriate diversity to find several optima. However, even with a speciation method, possible promising regions found late will not be able to be exploited because of the high selection pressure generated by already found peaks.

To avoid this harmful behavior, it would be desirable that the selection pressure in region of the search space far from the peaks already found does not depend of these peaks. This objective led us to consider a multi-population model, or island model, to improve the robustness of existing speciation methods. The use of island models to perform speciation has been studied by some authors with different motivations or techniques [2, 3, 4]. In this paper we propose a new algorithm aimed at multimodal optimization. It handles an island model cooperating with a classification tree speciation [5]. In Section 2, we describe in detail that algorithm. Our experiments are presented in Section 3 and some results relating to F6 Schaffer's function are discussed in Section 4. Section 5 makes up the conclusion.

2 A multipopulation genetic algorithm aimed at multimodal optimization

The algorithm that we propose identifies and maintains the best optima of an objective function, by generating subpopulations around these optima. In several recent "metaheuristics", the search for the optima of an objective function is performed as suggested by Glover [6], by using two processes, namely "exploration" and "exploitation". Roughly speaking, exploration aims at locating promising zones within the search domain and exploitation allows a descent at the bottom of detected valleys. The proposed algorithm tries to find a compromise between both processes. It comprises two steps alternating at a given frequency. During the first one, namely diversification (Exploration), a multipopulation algorithm associated with a speciation module detects the potential niches, first by combining the genetic material available, then by gathering the individuals in different subpopulations, in function of their similarities. In the second step, the algorithm performs an intensification (Exploitation), during a given number of generations, by allocating to each subpopulation a portion of the search space, the size of which is determined by a speciation module. This two-steps process is reiterated until a stopping condition is reached. The switching frequency between the two steps constitutes a parameter of the algorithm.

2.1 Diversification

The exploration is performed through the coupling between a multipopulation genetic algorithm and a speciation method (see on Figure 1). It aims at keep-
ing diversity among the subpopulations, by gathering individuals representing a same region inside one subpopulation.

**Multipopulation genetic algorithm.** Generally, GA's handle a single population, randomly generated within the search space of the problem at hand. However another model, called "island model", based on the use of several subpopulations, was subsequently proposed [8, 9]. To keep diversity in the course of the optimization process, a splitting of the population into separated subpopulations was performed. However to favour the proliferation of good individuals through the set of subpopulations, an exchange of individuals between subpopulations is periodically performed. In addition, that exchange of individuals coming from different regions allows the algorithm to generate, through the crossover operator, individuals belonging to regions not already explored. That exchange uses the "migration" operator.

![Fig. 1. Synoptic diagram of the diversification step](image)

In the case of the diversification operated in our algorithm the GA's shown on Figure 1 are operated with only one generation, for each subpopulation.

**Migration.** It consists in an exchange of individuals between subpopulations [10], allowing the population to better cover the search space and avoid a premature convergence. Migration of individuals between the different subpopulations, followed by application of genetic operators, achieves generation of new individuals representing still unexplored regions. The selection of individuals to be migrated and individuals to be replaced is performed at random, to the exclusion of the best individual of each subpopulation. The rate of migration (which determines, for a given subpopulation, the number of individuals to be replaced) allows the algorithm to control the level of diversity to be maintained inside the subpopulation.

**Speciation.** Speciation consists in the gathering of individuals according to their genetic likeness. With a multipopulation GA the request for that mech-
anism is fruitful to avoid the accumulation of several subpopulations around a same peak of the fitness function. In consequence, a high diversity will be maintained inside subpopulations. Several techniques have been proposed in the literature, such as clustering based methods [11], neighbouring techniques [10] and the speciation tree [5]. We used this last method, because its algorithmic complexity is $O(n \log n)$ and it does not require a lot of individuals (two for each peak) to achieve a good classification. In addition it needs no problem-dependent information, contrary to neighbouring methods, which use a problem-dependent parameter, namely the niche radius [5].

That speciation module is coupled with the multipopulation GA (see on Figure 1), so that at the end of the diversification each subpopulation gathers individuals both similar and exploiting the same peak of the fitness function. For that purpose, at each generation, all individuals from different subpopulations are gathered inside a common "speciation pool", which is subsequently processed by the speciation module. The number $k$ of subpopulations returned depends on the pool diversity and particularly on the landscape of the fitness. Therefore it is variable in function of the problem at hand. Practically, a maximal number, $P_{max}$, of subpopulations that can be returned is specified. It must be chosen so that the diversity is sufficient.

$$k \leq P_{max}$$ (1)

In case of multimodal optimization, $P_{max}$ must be higher than the desirable number of peaks to be determined. At the end of the speciation step, one keeps only the subpopulations for which the best individual is better than the average one.

**Distance.** Speciation uses a distance to evaluate the similarity between the individuals, thus clustering them into subpopulations formed with individuals having the same genotypic features. The method implemented is the normalized Euclidean distance: let be two individuals $X$ and $Y$ of dimension $n$; each of the $n$ parameters has max and min values (the search space being bounded). The distance $\text{dist}(X, Y)$ between $X$ and $Y$ is:

$$\text{dist}(X, Y) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \frac{(X_i - Y_i)^2}{(\text{max}_i - \text{min}_i)^2}}$$ (2)

### 2.2 Intensification

The intensification is used in Tabu Search [6], in conjunction with the diversification, to explore the neighbourhood of promising points of the search space. In the scope of the present algorithm, each population determined during the diversification step should be located in the neighbourhoods of the peaks of the fitness function, because these neighbourhoods are determined by the speciation
operator. So, it is natural to perform an intensification on each of these neighbourhhoods, to approach the optima accurately. The intensification consists in applying a genetic algorithm independently on each population during a fixed small number of generations, identical for all the populations.

The size of the populations is set to \( N \), identical for all the populations issued from the diversification. However, the diversification provides variable population sizes. The size \( N \) is obtained from the first intensification generation by selecting only \( N \) individuals for the reproduction. If there is only one individual in a population after the diversification, this individual will be selected \( N \) times. If there are more than \( N \), the best ones will be reproduced to obtain exactly \( N \) individuals, according to the selection algorithm of the GA.

The intensification domain, which is the search space associated with each population \( i \), is centered on the best individual \( x^i_0 \) determined just after the diversification step. Let \( x^i_{dj} \) be the \( j \)-th coordinate of this individual, let \( x^i_{kj} \) be the \( j \)-th coordinate of an individual \( k \) belonging to this population, then:

\[
x^i_{kj} \in [x^i_{dj} - \Delta^i_j, x^i_{dj} + \Delta^i_j],
\]

with

\[
\Delta^i_j = \max(10^{-3}|x^i_{max} - x^i_{min}|, \max_{l=1,...,N}|x^i_{dj} - x^i_{lj}|)
\]

where \( x^i_{max} \) is the upper bound of \( x^i_j \) in the global search domain, and \( x^i_{min} \) is its lower bound. The term \( "10^{-3}|x^i_{max} - x^i_{min}|" \) guaranties that \( \Delta^i_j \) is not null.

## 3 Experiments

The behaviour of the algorithm was studied with success by using several functions. We describe in the paper only the results obtained with the F6 Schaffer’s function [12], shown on Figure 2.

![Schaffer's F6 function](image)

**Fig. 2.** Representation of F6 (left) in one dimension in \([-100, 100]\), (right) in two dimensions in \([-10, 10]\)

It possesses one global optimum at \((0, 0): (F6(0, 0) = 0) \) and a large number of local optima. The global optimum is difficult to find because the value at the
best local optimum differs from only about $10^{-3}$. In addition, that local optimum is not punctual, but it forms a crown around the global optimum, which behaves like a trap. This feature will allow us to test the ability of our algorithm to isolate the region of the global optimum, by assigning to that optimum a specific subpopulation. Indeed, to efficiently deal with that function by using a GA, some authors advocate an hybridization with some local descent algorithm [7].

$$F_6(x_1, x_2) = 0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{(1 + 0.001 (x_1^2 + x_2^2))^2}$$

with $x_i \in [-100, 100], i = 1, 2$.

### 3.1 Experimental conditions

The configuration of the genetic algorithm handling each subpopulation is the following one:
- Linear ranking selection.
- Elitism.
- Stochastic Universal Sampling (SUS) [13].
- Real coding.
- Extended intermediate recombination [14]. Let $P_1$ and $P_2$ be the two parents; the two offspring $C_1$ and $C_2$ are generated in the following way:

$$\begin{align*}
C_1 &= P_1 + \alpha(P_2 - P_1) \\
C_2 &= P_2 + \alpha(P_1 - P_2)
\end{align*}$$

where $\alpha$ is an uniform random number in $[-0.25, 1.25]$. The crossover rate is 0.9.
- BGA Mutation [14]

$$\begin{align*}
x' &= x \pm 0.2(x_{max} - x_{min})\delta \\
\delta &= \sum_{i=0}^{k-1} \alpha_i 2^{-i}
\end{align*}$$

with $k = 16$ and $\alpha_i = 1$ with probability equal to $1/k$, otherwise $\alpha_i = 0$; the mutation rate is 0.9.
- Periodicity of diversification = 5 (one diversification generation after 4 intensification generations).
- The communication graph for migration is complete.

### 3.2 Criteria retained to validate the algorithm

We adopted the following criteria:
- Number of objective function evaluations $\text{NbEval}$ (in case of convergence) necessary to reach the global optimum with an accuracy relating to its position equal to $10^{-3}$. The maximal number of generations is equal to 100.
- Success rate: the genetic algorithm being of stochastic nature, we must proceed to several executions: 30, for all results discussed below. $\text{SuccessRate}$ represents the ratio of the number of successful executions (global optimum reached) to the total number of executions.
4 Results

We study the evolution of the algorithm versus two parameters: the number of individuals of each subpopulation ($N$) and the maximal number of subpopulations ($P_{max}$). Figure 3 shows the final distribution of the subpopulations for one execution of the program, with $P_{max} = 40$ and $N = 10$. We see that the algorithm reduces the search space drastically: all individuals are now within the domain $[-10,10]^2$, to be compared to the F6' initial search domain $[-100,100]^2$.

![Fig. 3. Final distribution of the subpopulations for $P_{max} = 40$ and $N = 10$.](image)

Study of the migration rate

The migration rate was varied from 0% to 100%, in the case: $P_{max} = 40$ and $N = 10$. The results are shown on Figure 4. For each diagram, 3 curves are represented: the evolution of the average and the evolutions of the upper and lower bounds of the 95% confidence interval.

It can be pointed out that when the migration rate increases, the number of evaluations slightly diminishes, and the success rate increases. These tests prove that the migration rate must be advantageously tuned at a high value, about 100%.

The dependence of the algorithm upon $N$ and $P_{max}$ is represented in Figures 5 and 6, for respective migration rates of 10% and 80%. They show that the algorithm is robust from the viewpoint of the success rate: an abrupt fall is observed only for $N < 10$. The evaluation number being increased when the number of individuals is increased, we propose to fix that parameter $N$ around 10.

However the maximal number of subpopulations must be high enough to keep a high success rate: a good compromise seems around $P_{max} = 30$ to 40. Results displayed on Figure 6 are better than those on Figure 5, what confirms the interest of choosing a high migration rate, as shown on Figure 4.
Fig. 4. Evolution of the optimization process versus the migration rate.

Fig. 5. Evolution of the algorithm versus $N$ and $P_{max}$ with a 10% migration rate.
Lastly we compared the results provided by our algorithm with that achieved by a standard GA (one single population), with or without speciation:

<table>
<thead>
<tr>
<th>Speciation + Clearing</th>
<th>N</th>
<th>NbEval (Avg)</th>
<th>Success rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>400</td>
<td>16100</td>
<td>27</td>
</tr>
<tr>
<td>Without speciation</td>
<td>400</td>
<td>24800</td>
<td>4</td>
</tr>
</tbody>
</table>

The rate of success is evaluated after 30 executions: thus, the convergence is observed only once for GA without speciation and 8 times for GA with speciation and clearing. Clearly, the results provided by our algorithm for F6 Schaffer’s function are notably better.

5 Conclusion

We have proposed a new algorithm comprising an island model cooperating with speciation for multimodal optimization. The concepts of diversification and intensification, assumed from Tabu Search, warrant a good diversity of the search process. The diversification steps allow the algorithm to explore the global search space to find new promising regions. The intensification steps prevent the algorithm to lose these regions because of the strong selection pressure generated by the peaks already found.

Compared to a standard real-coded GA’s, we have empirically shown that F6 Schaffer’s function can be optimized with an outstanding rate of success, especially when one selects high migration rates. Two tasks are in progress to more improve the efficiency of the algorithm: first, we empirically search for an optimal tuning of the parameters of the method; secondly we are systematically validating the algorithm over a large set of difficult test problems.
References

Optimizing through Co-evolutionary Avalanches

Stefan Boettcher,1* Allon G. Percus,2** and Michelangelo Grigni1***

1 Emory University, Atlanta, GA 30322
2 Los Alamos National Laboratory, Los Alamos, NM 87545

Abstract. We explore a new general-purpose heuristic for finding high-quality solutions to hard optimization problems. The method, called extremal optimization, is inspired by “self-organized criticality,” a concept introduced to describe emergent complexity in many physical systems. In contrast to Genetic Algorithms which operate on an entire “gene-pool” of possible solutions, extremal optimization successively replaces extremely undesirable elements of a sub-optimal solution with new, random ones. Large fluctuations, called “avalanches,” ensue that efficiently explore many local optima. Drawing upon models used to simulate far-from-equilibrium dynamics, extremal optimization complements approximation methods inspired by equilibrium statistical physics, such as simulated annealing. With only one adjustable parameter, its performance has proved competitive with more elaborate methods, especially near phase transitions. Those phase transitions are found in the parameter space of most optimization problems, and have recently been conjectured to be the origin of some of the hardest instances in computational complexity. We will demonstrate how extremal optimization can be implemented for a variety of combinatorial optimization problems. We believe that extremal optimization will be a useful tool in the investigation of phase transitions in combinatorial optimization problems, hence valuable in elucidating the origin of computational complexity.

1 Natural Emergence of Optimized Configurations

Every day, enormous efforts are devoted to organizing the supply and demand of limited resources, so as to optimize their utility. Examples include the supply of foods and services to consumers, the scheduling of a transportation fleet, or the flow of information in communication networks within society or within a parallel computer. By contrast, without any intelligent organizing facility, many natural systems have evolved into amazingly complex structures that optimize the utilization of resources in surprisingly sophisticated ways [2]. For instance, biological evolution has developed efficient and strongly interdependent networks in which resources rarely go to waste. Even the inanimate morphology of natural landscapes exhibits patterns far from random that often seem to serve a purpose, such as the efficient drainage of water [31].

* e-mail: stb@physics.emory.edu
** e-mail: percus@lanl.gov
*** e-mail: mic@mathcs.emory.edu
Natural systems that exhibit such self-organizing qualities often possess common features: they generally consist of a large number of strongly coupled entities with very similar properties. Hence, they permit a statistical description at some coarse level. An external resource (sunlight in the case of evolution) drives the system which then takes its direction purely by chance. Like flowing water breaking through the weakest of all barriers in its wake, species are coupled in a global comparative process that persistently washes away the least fit. In this process, unlikely but highly adapted structures surface inadvertently. Optimal adaptation thus emerges naturally, without divine intervention, from the dynamics through a selection against the extremely “bad”. In fact, this process prevents the inflexibility inevitable in a controlled breeding of the “good”.

Certain models relying on extremal processes have been proposed to explain self-organizing systems in nature [28]. In particular, the Bak-Sneppen model of biological evolution is based on this principle [3, 10]. It is happily devoid of any specificity about the nature of interactions between species, yet produces salient nontrivial features of paleontological data such as broadly distributed lifetimes of species, large extinction events, and punctuated equilibrium.

In the Bak-Sneppen model, the high degree of adaptation of most species is obtained by the elimination of badly adapted ones instead of a particular “engineering” of better ones. Species in the Bak-Sneppen model are located on the sites of a lattice, and each is represented by a value between 0 and 1, indicating its “fitness”. At each update step, the smallest value (representing the worst adapted species) is discarded and replaced with a new value drawn randomly from a flat distribution on [0, 1]. But the change in fitness of one species impacts the fitness of an interrelated species. Therefore, at each update step in the Bak-Sneppen model, the fitness values on the sites neighboring the smallest value are replaced with new random numbers as well. No explicit definition is given of the mechanism by which these neighboring species are related. Yet, after a certain number of updates, the system organizes itself into a highly correlated state known as self-organized criticality (SOC) [4]. In that state, almost all species have reached a fitness above a certain threshold. These species possess punctuated equilibrium: one’s weakened neighbor can undermine one’s own fitness. Co-evolutionary chain reactions called “avalanches” ensue; large fluctuations that make any possible configuration accessible.

2 Extremal Optimization

Extremal Optimization (EO) is inspired by previous attempts of using physical intuition to optimize. It opens the door to applying non-equilibrium processes, such as SOC, in the same manner simulated annealing (SA) [23] applies equilibrium statistical mechanics. The result is a general method that appears to be a powerful addition to the canon of meta-heuristics [27]. Its large fluctuations provide significant hill-climbing ability, which enables EO to perform well at phase transitions, “where the really hard problems are” [11, 1].
One popular hard optimization problem, to which we have applied EO successfully (see below and Refs. [8, 7]), is the graph bi-partitioning problem (GBP) [14, 23, 21]. In the GBP, we are given a set of \( n \) vertices, where \( n \) is even, and "edges" connecting certain pairs of vertices. The problem is to partition the vertices into two equal subsets, each of size \( n/2 \), with a minimal number of edges cutting across the partition. The size of the configuration space \( \Omega \) grows exponentially with \( n \), \( |\Omega| = \binom{n}{n/2} \), since all unordered divisions of \( n \) vertices into to equal-sized sets are feasible configurations \( S \). The cost function \( C(S) \) (called "cutsize") counts the number of "bad" edges that cut across the partition. A typical neighborhood \( N(S) \) for a local search [27, 30], mapping \( S \rightarrow S' \in N(S) \subset \Omega \), is a "1-exchange" of one randomly chosen vertex from each subset.

EO performs in general a search on a single configuration \( S \in \Omega \). \( S \) usually consists of a large number \( n \) of variables \( x_i \). The cost \( C(S) \) is assumed to consist of the individual cost contributions \( \lambda_i \) for each variable \( x_i \), which correspond loosely to the "fitness" values in the Bak-Sneppen model above. Typically, the fitness \( \lambda_i \) of variable \( x_i \) depends on its state in relation to other variables that \( x_i \) is connected to. Ideally, it is

\[
C(S) = \sum_{i=1}^{n} \lambda_i. \tag{1}
\]

For example, in the GBP the variables \( x_i \) are the vertices, each being assigned to a set "0" or "1." Each vertex has edges connecting it to a certain number of other vertices. Eq. (1) for the cutsize \( C(S) \) is satisfied, if we attribute to each vertex \( x_i \) a local cost \( \lambda_i = b_i/2 \), where \( b_i \) is the number of "bad" edges, whose cost is equally shared with the vertex on the other end of that edge.

For minimization problems in general, EO proceeds as follows:

1. Initialize a configuration \( S \) at will; set \( S_{\text{best}} = S \).
2. For the "current" configuration \( S \),
   (a) evaluate \( \lambda_i \) for each variable \( x_i \),
   (b) find \( j \) with \( \lambda_j \geq \lambda_i \) for all \( i \), i.e. \( x_j \) has the "worst fitness,"
   (c) choose at random a \( S' \in N(S) \) such that the "worst" \( x_j \) must change its state,
   (d) if \( C(S') < C(S_{\text{best}}) \) then set \( S_{\text{best}} = S' \),
   (e) accept \( S \leftarrow S' \) always, independent of \( C(S') - C(S) \).
3. Repeat at step (2) as long as desired.
4. Return \( S_{\text{best}} \) and \( C(S_{\text{best}}) \).

The algorithm operates on a single configuration \( S \) at each step. All variables \( x_i \) in \( S \) have a fitness, of which the "worst" is identified. This ranking of the variables according to individual costs – unique to EO – provides the only measure of quality on \( S \). It implies that all other variables are "better" in the current \( S \). There is no parameter to be adjusted for the selection of better solutions aside from this ranking. In fact, it is only the memory encapsulated in this ranking that directs EO into the neighborhood of increasingly better solutions. Those "better" variables only possess punctuated equilibrium: their memory
Extremal Optimization

Simulated Annealing

Fig. 1. Evolution of the cutsize \( C(S) \) during a typical run of (a) EO and (b) SA for the \( n = 500, c \approx 5 \), random graph \( G_{500} \) introduced in Ref. [21]. The best cutsize ever found for \( G_{500} \) is 206 (see Fig. 2). In contrast to SA, which has large fluctuations in early stages of the run and then converges much later, extremal optimization quickly approaches a stage where broadly distributed fluctuations allow it to scale barriers and probe many local optima.

gets erased when they happen to be connected to one of the variables forced to change. On the other hand, in the choice of move to \( S' \), no consideration to the outcome of such a move is given, and not even the worst variable itself is guaranteed to improve its fitness. Large fluctuations in the cost accumulate over many updates [3], while merely the bias against "bad" fitnesses guides EO back towards improved solutions, see Fig. 1.

Disadvantages of EO are that a definition of fitness for individual variables may be ambiguous or even impossible. Also, variables may be strongly connected such that each update destroys more well-adapted variables than it could ever hope to improve [8]. In highly connected systems, EO is slowed down considerably by reevaluating fitnesses [step (2a)]. For many problems, these disadvantages do not apply or are surmountable. In particular, problems in the important optimization class MAX-SNP [29] fit naturally into the EO-framework. MAX-SNP problems have boolean variables and a collection of bounded-arity boolean terms and we seek an assignment satisfying as many (or as few) terms as possible. Such problems have a natural choice of fitness functions, and typically have low variable connectivity. Indeed, some complete problems for the class have bounded connectivity in the worst case. MAX-SNP complete problems include MAX-\( K \)-SAT, \( K \)-COL, and MAXCUT (similar to GBP), discussed below.

3 Comparison with other Heuristics

The most apparent distinction between EO and other methods is the need to define local cost contributions for each variable, instead of merely a global cost.\(^1\)

\(^1\) Apparently, local costs have previously been used in an otherwise unrelated ensemble Monte Carlo approach [12].
EO's capability appears to derive from its ability to access this local information directly. EO's ranking of fitnesses superficially appears like the rankings of possible moves in some versions of SA [17, 30] and in Tabu search [15, 30]. But these moves are evaluated by their anticipated outcome, while EO's fitnesses reflect the current configuration $S$ without biasing the outcome.

**Simulated Annealing (SA):** SA [23] emulates the behavior of frustrated systems in thermal equilibrium: if one couples such a system to a heat bath of adjustable temperature, by cooling the system slowly one may come close to attaining a state of minimal energy (i.e., cost). SA accepts or rejects local changes to a configuration according to the Metropolis algorithm, requiring equilibrium conditions ("detailed balance") along a well-tuned "temperature schedule."

In contrast, EO drives the system far from equilibrium: aside from ranking, it applies no decision criteria, and all new configurations are accepted indiscriminately. Instead of tuning a whole schedule of parameters, EO often requires fewer choices. It may appear that EO's results should resemble an ineffective random search, similar to SA at a fixed but finite temperature. But in fact, by persistent selection against the worst fitnesses, one quickly approaches near-optimal solutions. Significant fluctuations still remain at late run-times (unlike in SA, see Fig. 1), crossing sizable barriers to access new regions in configuration space.

**Genetic Algorithms (GA):** While similarly motivated, GA [20, 16] and EO algorithms have hardly anything in common. GAs, mimicking evolution on the genotypical level, keep track of entire "gene pools" of configurations from which to select and "breed" an improved generation of solutions. By comparison, EO, based on evolutionary competition at the phenomenological level of "species," operates only on a single configuration, with improvements achieved merely by elimination of bad variables. EO, SA, and most other meta-heuristics perform a local search but in GA cross-over operators perform global exchanges.

### 4 Applications of Extremal Optimization

**Ground States of Spin Glasses:** A simple version of a spin glass [25] consists of a $d$-dimensional hyper-cubic lattice with a spin variable $\sigma_i \in \{-1, 1\}$ placed on each site $i$, $1 \leq i \leq n = L^d$. Every spin is connected to each of its nearest neighbors $j$ via a bond variable $J_{i,j}$ drawn from some distribution $P(J)$ of zero mean and unit variance. Spins may be coupled to an arbitrary external field $h_i$. We try to find "ground states," i.e., lowest energy configurations $S_{\text{min}}$ of

$$C(S) = H(\sigma_1, \ldots, \sigma_n) = -\frac{1}{2} \sum_i \sum_j J_{i,j} \sigma_i \sigma_j - \sum_i \sigma_i h_i. \quad (2)$$

Arranging the spins into optimal configurations is hard due to "frustration" [25]. To implement EO, we define as fitness the local energy for each spin

$$\lambda_i = -\sigma_i \left( \frac{1}{2} \sum_j J_{i,j} \sigma_j + h_i \right), \quad (3)$$
and Eq. (2) turns into Eq. (1). Our implementation suggests that EO may be well suited for problems representable as a spin-Hamiltonian [25].

**Satisfiability (MAX-K-SAT):** Instances of the satisfiability problem MAX-K-SAT consist of a formula composed of $M$ clauses. Each clause contains $K$ literals (i.e. $x_i$ or $\neg x_i$), drawn randomly from a pool of $n$ boolean variables $x_i$. A clause is verified, if at least one of its $K$ literals is true (logical “or”), and the entire formula is verified only if every clause is true (logical “and”). Here, we try to maximize the number of true clauses by some configuration of the variables.

MAX-K-SAT has an obvious EO-implementation: For each variable we set $\lambda_i = 1/K \times \# \text{ of false clauses containing } x_i$. Again, Eq. (1) holds. Typically, $K = O(1)$ and $M = O(n)$ so that each variable appears only in a few ($\approx M/n$) clauses, each connecting it to $\sim K$ other variables. The phase transition in 2-SAT and 3-SAT has been investigated in Refs. [26,1] on small instances using exact methods. We expect that EO would perform very well on those instances.

**Graph Coloring (K-COL):** Given $K$ different colors to label the vertices of a graph, we need to find a coloring that minimizes the number of edges connecting vertices of identical color. We implement EO for K-COL by defining for each vertex the number of equally colored vertices connected to it as fitness. Similar to a spin glass, this problem is hard due to local frustration [25], in distinction to the global constraints in the GBP. A simple neighborhood consists of the re-coloring of a single vertex each update. Below, we present results of using EO in analyzing the phase transition in 3-COL, first investigated in Refs. [11,1].

## 5 Experimental Results

**Simple EO Application to Graph Partitioning:** Following Ref. [21] (Fig. 9 there), we tested early implementations of EO [8] on their $n = 500$ random graph $G_{500}$ of connectivity $c \approx 5$. In a 1000-run sample from different random initial conditions, we determined the frequency of solution obtained, see Fig. 2. For comparison, we have also implemented the SA algorithm as given in Ref. [21] on the same data structure used by our EO program. We have allowed runtimes for EO about three times longer than the time it took for SA to “freeze,” since EO still obtained significant gains. We checked that neither the best-of-three runs of SA, or a three times longer temperature schedule, improved the SA results significantly. While the basic, parameter-free version of EO from Sec. 2 is already competitive, the best results are obtained by $\tau$-EO.

### $\tau$-EO Implementation

$\tau$-EO is a general modification of EO which improves results and avoids “dead ends” that occur in some implementations at the expense of introducing a single parameter [8]. We rank all the variables $x_i$ according to fitness $\lambda_i$, i.e. find a permutation $\Pi$ of the vertex labels $i$ such that

$$\lambda_{\Pi(1)} \geq \lambda_{\Pi(2)} \geq \ldots \geq \lambda_{\Pi(n)}. \quad (4)$$

The worst variable $x_j$ [see step (2b)] is of rank $1$, $j = \Pi(1)$, and the best variable is of rank $n$. Consider a probability distribution over the ranks $k$,

$$P_k \propto k^{-\tau}, \quad 1 \leq k \leq n, \quad (5)$$
for a fixed value of the parameter $\tau$. On each update, for each independent variable $x$ to be moved, select distinct ranks $k_1, k_2, \ldots$ according to $P_k$. Then, execute step (2c) such that all $x_{i_1}, x_{i_2}, \ldots$ with $i_1 = \Pi(k_1), i_2 = \Pi(k_2), \ldots$ change. For instance, in the bi-partitioning problem, we choose both variables in the 1-exchange according to $P_k$, instead of the worst and a random one. Although the worst variable of rank $i = 1$ will be chosen most often, sometimes (much) higher ranks will be updated instead. In fact, the choice of a power-law distribution\(^2\) for $P_k$ ensures that no rank gets excluded from further evolution while maintaining a bias against variables with bad fitness.

Clearly, for $\tau = 0$, $\tau$-EO is exactly a random walk through $\Omega$. Conversely, for $\tau \to \infty$, the process approaches a deterministic local search, only swapping the lowest-ranked variables, and is bound to reach a “dead end.” Indeed, tests of both, $\tau = 0$ and $\tau = \infty$, yield terrible results! In the GBP, we obtained our best solutions for $\tau \approx 1.4 - 1.6$. Under preliminary testing we find that there may be a link between the optimal choice for the parameter $\tau$ and a transition to “non-ergodic” behavior in the sense that for larger values of $\tau$ certain configurations in $\Omega$ may become inaccessible during the time of a complete EO-run. In fact, on the basis of that observation we have developed an argument to approximate $\tau \sim \ln(n)/(\ln(n))$ [9] where $t = An$ with $1 \ll A \ll n$ is the runtime. (Typically, we use $A \approx 10^2$ for graphs of size $n \approx 10^4$, consistent with $\tau = 1.5$.) Tests with longer runtimes indeed favor larger $\tau$ values, while larger graphs require smaller values of $\tau$.

**Results on Large Graphs:** In Tab. 1 we summarize $\tau$-EO’s results on large-$n$ graphs, using $\tau = 1.4$ and best-of-10 runs. On each graph, we used as many update steps $t$ as appeared productive for EO to reliably obtain stable results. This varied with the particularities of each graph, from $t = 2n$ to $200n$, and the reported runtimes are of course influenced by this. It is worth noting, though, that EO’s average performance has been varied. For instance, half of the *Brack2*
Table 1. Best cutsizes (and allowed runtime) for a testbed of large graphs. GA results are the best reported [24] (with a 300MHz CPU). \( \tau \)-EO results are from our runs (200MHz). Comparison data for three of the large graphs are due to results from heuristics in Ref. [19] (50MHz). METIS is a partitioning program based on hierarchical reduction instead of local search [22], obtaining extremely fast deterministic results (200MHz). Runtimes comparisons here are at best qualitative.

<table>
<thead>
<tr>
<th>Large Graph</th>
<th>GA</th>
<th>( \tau )-EO</th>
<th>[19]</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hammond</td>
<td>(n = 4720; c = 5.8)</td>
<td>90 (1s)</td>
<td>90 (42s)</td>
<td>97 (8s)</td>
</tr>
<tr>
<td>Barth5</td>
<td>(n = 15606; c = 5.8)</td>
<td>139 (44s)</td>
<td>139 (64s)</td>
<td>146 (28s)</td>
</tr>
<tr>
<td>Brack2</td>
<td>(n = 62632; c = 11.7)</td>
<td>731 (255s)</td>
<td>731 (12s)</td>
<td>—</td>
</tr>
<tr>
<td>Ocean</td>
<td>(n = 143437; c = 5.7)</td>
<td>464 (1200s)</td>
<td>464 (200s)</td>
<td>499 (38s)</td>
</tr>
</tbody>
</table>

runs returned cutsizes near 731, but the other half returned cutsizes of above 2000. This may be a product of an unusual structure in these particular graphs.

Phase Transitions in Combinatorial Optimization: In extensive numerical studies [7] we have shown that \( \tau \)-EO outperforms SA near phase transitions where graphs begin to "percolate" and cutsizes first become non-zero, see Fig. 3. Studies on the average rate of convergence towards better-cost configurations as a function of runtime \( t \) indicate power-law convergence [18], roughly like \( C(S_{\text{best}}) \sim C(S_{\text{min}}) + At^{-\beta} \) with \( \beta \approx 0.45 \). Of course, it is not easy to assert for graphs of large \( n \) that those runs in fact converge close to the optimum \( C(S_{\text{min}}) \), but finite-size scaling analysis seems to justify that expectation [9].

In an even more impressive performance, we used EO to completely enumerate all optimal solutions \( S_{\text{min}} \) near the critical point for random graphs in 3-COL. Instances of random graphs typically have a high ground-state degeneracy, i.e., possess a large number of equally optimal solutions \( S_{\text{min}} \). In Ref. [26] it was shown that at the phase transition of 3-SAT the fraction of constrained vari-

Fig. 3. Plot of the error in the best result of SA relative to EOs on identical instances of (a) random graphs and (b) geometric graphs as function of the mean connectivity \( c \). The percolation points are at (a) \( c = 1 \) [13] and (b) \( c \approx 4.5 \) [5], the critical points for the GBP (the first time a component of size \( > n/2 \) appears) are slightly above that [e.g., at \( c = 2 \ln 2 = 1.386 \) for (a)]. SA's error relative to EO near the critical point in each case rises with \( n \).
Fig. 4. Plot of the average (a) cutsize and (b) backbone fraction as a function of the connectivity $c$ for random graph 3-COL. We have generated 2,300, 500, 280 and 125 instances for $n = 32, 64, 128, \text{ and } 256$, respectively, at each value of $c$. The prediction for the critical point of $c_{\text{crit}}(3) \approx 4.73$ is indicated by a vertical line. The backbone fraction seems to develop a finite jump at the critical point for $n \to \infty$.

ables, i.e. those that are found in an identical state in all $S_{\text{min}}$, discontinuously jumps to a non-zero value. It was conjectured that the first-order phase transition in this “backbone” would exist for any NP-hard problem. To test those claims for 3-COL, we generated a large number of random graphs and explored $\Omega$ for as many ground states as $\tau$-EO could find. We fixed runtimes at $\approx 100n^2$, well above the times needed to saturate the set of all $S_{\text{min}}$ in repeated trails on some test instances. Such long runtimes favored a large value of $\tau = 2.7$. For each instance, we measured the cutsize, entropy, and the “backbone.” Due to the symmetry under interchanging colors, the backbone here consists of the fraction of constrained pairs of vertices, i.e. those which are in the same relative state (same or opposite color) in all ground states. Averaged results are given in Figs. 4a-b. As predicted in Ref. [26], asymptotically for large $n$ the backbone fraction seems to jump discontinuously at the critical connectivity, $c_{\text{crit}} \approx 4.73$.

This work was supported in part by the URC at Emory University, NSF grant CCR-9820931, and an LDRD grant from Los Alamos National Laboratory.

References

22. G. Karypis and V. Kumar, *METIS, a Software Package for Partitioning Graphs*, see http://www-users.cs.umn.edu/~karypis/metis/main.shtml,
Evolution of Altruism in Viscous Populations: 
Effects of Altruism on the Evolution of Migrating Behavior

Paul den Dulk and Martijn Brinkers

University of Amsterdam, Psychonomics department, 
Roetersstraat 15, 1018 WB, Amsterdam, Netherlands
{dendulk,brinkers}@psy.uva.nl

Abstract. In the study of the evolution of altruism two possible mechanisms have been distinguished. One is reciprocal altruism, the other is kin selection. This paper focuses on altruism based on kin selection in environments in which there is little migration of individuals ("viscous populations"). Two simulations show that the evolution of evolutionarily stable non-reciprocal altruism is possible in societies with limited dispersal, confirming earlier research [1]. It has been questioned, however, whether these conditions occur in natural situations. This paper demonstrates that if the tendency to migrate is itself represented as a gene controlled by an evolutionary process, a tendency not to migrate evolves among altruists. Thus, the conditions which would allow the evolution of altruism are not merely a matter of coincidence, but are actively selected for, increasing the likelihood that it will be present in natural situations.

1 Introduction

It is hard to understand how altruistic behavior, in which the altruistic benefits someone else at the expense of himself, could evolve in a process that selects for the fittest individual [1-3]. The scientific research on this problem has for a large part been done within the Prisoner's Dilemma (PD) paradigm [4, 5]. In a PD two actors have to play a game in which both have the choice to either cooperate (C) or defect (D). If both choose to cooperate they both receive a certain reward R. If both choose to defect they both get a lower reward P, R > P. If however one decides to defect whereas the other decides to cooperate the defector gets the highest reward T, T > R, and the cooperator gets the lowest reward S, P > S. In sum T > R > P > S. In a single trial PD the best strategy is to always defect. In the iterated Prisoner's Dilemma, however, cooperation is an effective strategy because the players play repeatedly against the same opponent and have the opportunity to retaliate for the opponent's defections. Therefore, there is personal gain for the altruist in the long run. This paper studies non-reciprocal altruism, in which there is no long-term reward for a cooperating individual. Even without long term reward for the individual,
altruism can evolve through kin selection (also called inclusive fitness [1]) when it is directed towards close kin, who share many genes with the altruist. If a certain gene for altruism causes the protection of an organism's offspring, while putting itself to risk, it is likely that the offspring also possesses this gene. This way a gene for altruism could spread despite the price paid by the altruist itself. Hamilton mentioned that within a viscous population, where there is little migration of individuals, altruism towards others—not necessarily kin—might evolve because they will tend to be genetically similar to the altruist [1]. Previous research on the evolution of altruism has shown that non-reciprocal altruism can evolve in societies which have the right amount of limited dispersal [5, 6]. In this paper we will present two simulations that show dispersal itself is also subject to evolution.

2 Prisoner's Dilemma simulation: Evolution of altruism

This simulation demonstrates that an evolutionarily stable strategy (ESS) of altruistic behavior can emerge because of limited dispersal. The simulation setup is inspired by the work of Nowak and May [5]. The simulation world was a torus shaped grid of 100 by 100 cells, in which every cell was occupied by an agent. Each update cycle of the grid consisted of two parts. First, all agents played a one trial Prisoner's Dilemma with all of its eight direct neighbors. This resulted in a profit value, which was the summed return of these eight trials. Second, a new generation was created by replacing each agent with a copy of the neighboring agent of the previous generation with the highest profit value. A successful agent could thus get eight children in the next generation. The agent itself was always replaced by a child of one of its neighbors, i.e. it could not get a child at its own position. The agents had one of two possible strategies, either always defect (all-D) or always cooperate (all-C). All-D agents were called egoists and all-C agents were called altruists. For the initial population, every cell was randomly set to altruist or egoist. This strategy was fixed for the duration of the simulation. Apart from this initial randomization the simulation was deterministic (When two cells had the same profit value this could only mean they had identical strategies in an identical situation because all of the 18 possible profit values were unique). The rewards for the PD were: 10 points when you defect a cooperator, 9 points when both cooperate, 1 point when both defect, 0 points when you cooperate with a defector. This is a normal PD pay-off in that it satisfies the $T > R > P > S$ relation, but the difference between $R$ and $P$ was chosen high so that a small cluster of altruists could expand facing a cluster of egoists, see Fig. 2 for an example. With this pay-off scheme there was a relatively high reward in a $C-C$ game, compared to a $D-D$ game, but still less than a $D-C$ game, thus, with this scheme, egoism would still be the best strategy in a one trial Prisoner's Dilemma.

Ten simulations were run, all with a different initialization. All resulted in a stable amount of altruists, 96.5% averaged over all simulations. This demonstrates the main purpose of the simulation: non-reciprocal altruism can be an evolutionarily stable strategy in viscous populations.
Fig. 1. In (a) four altruistic cells (white) are facing a cluster of egoistic cells (gray). Of the middle four cells the total profit value is shown. The cells with profit 26 received their profit value from the two neighboring altruistic cells (2x10) and six neighboring egoistic cells (6x1). While the altruist receives a total profit value of 27 from 3 neighboring altruists (3x9), it gains nothing from the egoists. In the next iteration all cells neighboring the altruist with the highest profit value of 27 will be replaced by its children, as shown in (b).

Most remaining egoists formed a static pair (like Fig. 2 (a) and (b)) or a cycle of states (Fig. 2 (c), (d), (e)). These resemble the flasher and still-lifes in Conway's Game of Life. Interestingly, although not relevant for our purposes, there were also gliders that were very similar to the gliders in the Game of Life (Fig. 2(f)).

In order to investigate the influence of locality in the distribution of agents on altruism, a control simulation was done with the same parameter settings as in the above simulations, but now new agents were placed at random locations instead of next to their parents. In this simulation setup the altruists all died in just a few iterations, demonstrating that viscosity plays a role in altruism and that with a high dispersal no altruism can evolve.

Fig. 2. State (a) and (b) are static pairs of egoist (gray cells) which replace each other in consecutive iterations. States (c), (d) and (e) form a stable cycle; (b) to (c), (c) to (d) and (e) back to (c). State (f) is a 'glider' like configuration, which displaces one cell per iteration in the direction depicted.
Prisoner’s Dilemma simulation: Evolution of migration

In simulation 1 there was low migration of the agents (new agents were placed besides their parent), whereas in simulation 2 there was maximal migration (new agents could be placed anywhere on the map). In the following simulation, migration was made an inherited property of the agents themselves. The migration 'gene' \((p\text{-migrate})\) had a value between 0 and 1, which determined the probability of migration, where 1 meant maximal migration. The initial population had random \(p\text{-migrate}\) values chosen from a flat distribution between 0 and 1. The migration parameter of the parent determined the probability that the child was put next to the parent or whether it was put at a random position. The random position was always a cell where other migrating agents would have been born. So, non-migrating agents could not be replaced by migrating agents. At birth an agent inherited the migration parameter of the parent added with a random mutation between -0.01 and 0.01. Resulting migration values outside the [0,1] interval were clipped.

Ten simulations with different initial seeds were run. All converged to a stable percentage of altruists. After 1000 iterations 79% of the agents were altruists whereas 21% were egoists (averaged over the last 100 iterations and all 10 replications). In Fig. 3(a) we plotted the percentages over the first 200 iterations. It can be seen that initially the egoists rapidly increased in number, almost replacing all altruists. However, in each run a few clusters of altruists remained and these clusters expanded to fill the majority of the cells. The most interesting finding was that the average migration probability for the altruists had dropped to 0.017. The evolutionary process itself had established the limited dispersal which is necessary for the evolution of altruistic behavior. To formulate it another way: if altruism can benefit a coherent sub-population then the coherence is itself subject to evolution.

Another interesting aspect is that there was still a high percentage of egoists left in the population: 21% whereas in the first simulation only 3.5% were egoistic. The migration probability for all the egoists systematically approached one in all simulations indicating that it was not just the result of a random walk introduced by the mutation. Averaged over the last 100 iterations and 10 replications \(p\text{-migrate}\) for egoists was 0.96. The 'preference' for migration of the egoists comes from the fact that it offers a chance to get out of an environment of egoists into an environment of altruist. Although the probability of being born in an environment of altruists is low—because of the low \(p\text{-migrate}\) of the altruists—the benefit is very high once it happens, because an egoist surrounded by altruists will have eight children (see Fig. 2 transition (c) to (d)). These eight children are again swapped somewhere else because of the high \(p\text{-migrate}\) of the egoists.

Fig. 3(b) shows a density plot of all the individuals in the population over the first 160 iterations. All the high migration values belong to egoists and all low values to altruists. It is clear that two distinct groups of migration values emerge.
4 Discussion on the Prisoner's Dilemma simulation

The main conclusion of the previous simulation is that the limited dispersal, which is necessary for altruism as an ESS, is itself subject to selection. The effect such a mechanism might have on natural situations is that individuals of the same species will tend to stick together in groups. In our simulation the non-migrating individuals stay on exactly the same location, whereas in a natural situation social groups might migrate together, like herds do.

A further conclusion from our simulations is that evolutionarily stable altruism depends on the initial conditions. In simulation setups with too few initial altruists (unreported), all altruists died out after just a few iterations. Once egoists had spread through the entire population it was resistant against invasion from altruists.

A problem with simulation research is that arbitrary parameter settings can be crucial for the obtained results. Even if the parameters within a simulation are carefully chosen, the choices made in the design of the simulation, or even the structure of the computer program, could be crucial for the outcome. No matter how carefully the parameter choices are made it will remain an open question how general such a principle is. In order to investigate the generality of the results from the first simulation we designed a different simulation, in which we again showed that altruism could evolve in viscous populations, and again made migration subject to an evolutionary process. As Wiener put it 'Generality has to be found it can not simply be declared' [7].

5 The river cross simulations

In the next simulation a population of agents had to survive and reproduce in a world consisting of islands. In order to survive they had to gather food from the islands, but they could only reach these by using another agent's body as a bridge. An agent could form a bridge with its body for other agents by stepping
into the water, the consequence was that it died of drowning. (These simulations are continuations of earlier work [8]).

5.1 Simulation Setup

The world consisted of 220 by 220 cells placed on a grid (torus). Each cell could either be a water cell or a land cell. Within this grid, 400 islands, each 10 by 10 cells, were placed. Canals of 1 cell in width separated the islands. Land cells could contain food or an agent, and water cells could contain a dead agent forming a bridge. A living agent could cross the canal by using this bridge.

Agent behavior: The agent's behavior was determined by four parameters: $p_{-}water$, $p_{-}food$, $p_{-}land$, $p_{-}cross$. Only the first was subject to mutation the others were fixed; $p_{-}food$ at 1.0, $p_{-}land$ at 0.8 and $p_{-}cross$ at 1.0. These values were based on earlier simulations, where these parameters were also mutated. Each of these parameters corresponded with one of four cells the agent could see in front of it: empty water, food on land, empty land, or a floating agent forming a bridge. The parameters represented the probability of stepping forward when in front of the corresponding type of cell. A random number, from the interval $[0,1]$ was drawn. When this number was below the corresponding $p$ value the agent would step forward, otherwise it would randomly turn left or right on the spot. Apart from the cell in front of it the agent used no other information, and it had no internal state. Agents could not step on top of other living agents. When one was in front of it, it always turned randomly left or right on the spot. The agents were synchronously updated.

Food and Energy: After each cycle, hundred cells were randomly chosen anywhere on the map. On these cells a food patch was placed when it was an empty land cell. When stepping on a food patch the energy of the food patch ($Energy_{Food}$) was added to the agent's energy and the food patch was removed. The agent always lost one unit of energy per iteration. When the agent used up all its energy it died of starvation.

Water: When stepping into water the agents immediately drowned. The agents remained 'floating' during a number of iterations as determined by the $BridgeTime$ parameter. While floating, other agents could step on top of the agent using it as a bridge.

Reproduction: The agents could multiply through asexual reproduction. The agent gave birth when it had gathered enough energy ($Energy_{ToGetChild}$). When a child was born the parent's energy was reduced by a certain amount of energy ($Energy_{CostChild}$) and this formed the starting energy of the child. The child inherited the parent's $p_{-}water$ value, but for each child a slight mutation was added. For each mutation, a random number was drawn from the interval $[-1,1]$ and this was multiplied by the $Mutation$ parameter and was added to the $p_{-}water$ value. Resulting $p_{-}water$ values outside the $[0,1]$ interval were clipped. A newborn agent was placed on the cell directly behind its parent.

Age: When agents reached the maximum age ($MaxAge$), expressed in the number of iterations after birth, they died of old age.
Initialization: At the start of each simulation, a number of agents ($InitNumAgents$) and a number of food patches ($InitNumFood$) were placed at random locations, excluding water cells. For $p$-water initial $p$ values were randomly chosen between zero and one. The initial agent’s energy was randomly chosen from the interval $[0, EnergyToGetChild]$. Its initial age was randomly chosen from the interval $[0, MaxAge]$.

Simulator: The simulations were performed with a self-built simulator. To rule out any effect of bugs two simulators were independently developed by the two authors, without accessing each other’s code, using two different languages.

$p$-migrate: Sometimes the agent’s child was placed at a random location instead of next to the parent in order to distribute relatives over the map. The probability of migration was determined by the $p$-migrate parameter. As a consequence of this an altruist that would body-bridge would help unrelated others with this. The population was mixed up with the following procedure. When a parent $A$ gave birth to a child $X$, the child was not immediately placed on the map, but had to wait until a second parent $B$ gave birth to a child $Y$. Child $X$ was then placed next to the parent $B$, while child $Y$ had to wait for a third parent $C$ to give birth before it was placed on the map, etc. In practice the delay of placement was on average $3/4$th of a cycle. With this swap method the balance between food and newborns was not affected in any way, and it avoids placing newborns at random locations on the map, which would take away the need for island hopping because new agents could then be born onto an uninhabited island.

Parameter settings: $MaxAge$ 250, $EnergyToGetChild$ 120, $EnergyCostChild$ 60, $EnergyFood$ 9, $BridgeTime$ 30, $Mutation$ 0.0001, $InitNumFood$ $30/\text{Number Islands}$, $InitNumAgents$ $15/\text{Number Islands}$.

6 River cross simulations: Evolution of altruism

Two simulations were run (each replicated 10 times); one with $p$-migrate set to zero and one with $p$-migrate set to one. Both simulations were run for 2,500,000 iterations. In the no-migrate condition there was a rapid decrease in average $p$-water, which makes sense because stepping into water causes instant death, nevertheless it stabilized at 0.0071 (all reported values are averaged over the last 100,000 iterations of the simulation). This seems quite low, but the percentage of agents that died of drowning, instead of old age or starvation, was still 18%. This is relatively high because during the agents life there is a chance to walk into water each time the agents stand in front of water. We conclude from this that altruistic behavior is stable in this no-migration condition. Fig. 4(a) shows a sample of $p$-water values over time.

In the migrate condition (Fig. 4(b)), $p$-water reached values of about 0.0005 on iteration 80,000. However, due to the lack of transportation to new islands, the whole population went extinct. This result stresses that the intuition that altruism evolves because of its value for the whole, is flawed. Without the proper
Fig. 4. The p-water values plotted for all agents for a number of iterations

7 River cross simulation: Evolution of migration

In this simulation \( p\text{-migrate}\) was treated as a genetic trait, which was mutated in the same way as the \( p\text{-water}\) parameter. Initially all agents were given a \( p\text{-migrate}\) of 0.0. The simulation was run for 10 million iterations. A density plot of the \( p\text{-migrate}\) values over time is shown in Fig. 5(a). It can be seen that two threads emerge. In Fig. 5(b) a density plot is shown of the \( p\text{-migrate}\) values against the \( p\text{-water}\) values. Two clearly separate groups can be distinguished that differed in both \( p\text{-water}\) value and \( p\text{-migrate}\) value, where the group with low \( p\text{-migrate}\) has a high \( p\text{-water}\) (altruists) and the group with high \( p\text{-migrate}\) has a low \( p\text{-water}\) (egoists). At the end of the simulation the \( p\text{-migrate}\) of the egoistic group was close to 1.0 and that of the altruistic group was close to 0.0. These two groups are thus very similar to the two groups that arose in the Prisoner’s Dilemma simulation.

8 Discussion river cross simulation

As in the first simulation the initial conditions were crucial for the evolution of altruism. The simulation presented had the \( p\text{-migrate}\) values initialized to 0.0. Although in simulations with initial \( p\text{-migrate}\) values of around 0.05 altruism also evolved, but it did not with 0.1 or higher. Values higher than 0.1 or random values led to an immediate extinction of altruists. Once egoism was established altruist could not return in the population. The same phenomenon was also observed in the PD simulations. The problem to evolve altruism from such initial conditions comes from the fact that altruism depends on mutual benefit. A single altruist is always worse off among egoists, and only by the clustering of
altruists can they remain evolutionarily stable. Given the fact that \( p\text{-water} \) was randomly chosen at the start of the simulation in combination with a randomly chosen \( p\text{-migrate} \), the chance of having an almost uniform group of altruists that also has a low \( p\text{-migrate} \) was almost zero. Once altruism has disappeared, single mutations towards altruism are strongly selected against. It is hard to judge whether in natural situations the initial conditions are met that allow the evolution of altruism, in fact there are no clearly distinguishable initial conditions in nature, rather evolution is a continuous process where different phyla evolve through all different kinds of situations. What we can conclude, however, is that if initial conditions allow altruism, there will also be a selection pressure to preserve these conditions when benefits of mutual altruism have a significant advantage compared to mutual egoism.

9 Conclusions

An interesting aspect of the simulations, not expected by the authors, is that in both simulations a group of egoistic parasites emerged, which differed in both strategy and migration rate. Once the two groups had fully evolved, the ratio between the two groups reached an equilibrium with relatively little fluctuations. Such an equilibrium is reminiscent of the equilibrium found in the hawk and dove game [9]. There is, however, an important difference with the hawk and dove situation. In the hawk and dove situation the individuals play for the highest personal gain and could in principle change their strategy depending on the circumstances. The altruists in our simulations do not maximize their personal gain; if they did they should always defect, just like in a normal one trial PD.

Both simulations show that the evolution of altruism depends crucially on the initial conditions. Without sufficient altruist friendly circumstances altruistic individuals will go extinct and will not return. The egoists are thus resistant to invasions from altruists. This is because altruists need be clustered in order to stand up against egoists.
Our main conclusion is that the tendency to migrate (or not) is itself subject to selection. We think this is an important finding because it can reverse the causal arrow in thinking about the evolution of altruism. For instance, when there is altruism within groups of closely related insects one could say the altruism evolved because of the genetic similarity within the group. On the other hand the habit to live in groups of relatives could evolve because it allows evolutionarily stable altruism and the benefits that come with it. Whereas one is usually inclined to question the evolution of altruism with the dispersal of individuals as a given premise, we show how dispersal itself is affected by the benefit of mutual altruism. In practice, however, there is probably an interaction between these two.

References

A Game-Theoretic Approach to the Simple Coevolutionary Algorithm

Sevan G. Ficici and Jordan B. Pollack

DEMO Lab—Department of Computer Science
Brandeis University, Waltham Massachusetts 02454 USA
www.demo.cs.brandeis.edu

Abstract. The fundamental distinction between ordinary evolutionary algorithms (EA) and co-evolutionary algorithms lies in the interaction between coevolving entities. We believe that this property is essentially game-theoretic in nature. Using game theory, we describe extensions that allow familiar mixing-matrix and Markov-chain models of EAs to address coevolutionary algorithm dynamics. We then employ concepts from evolutionary game theory to examine design aspects of conventional coevolutionary algorithms that are poorly understood.

1 Introduction

While formal models of evolutionary algorithm (EA) dynamics burgeon in general [24,26,21], co-evolutionary algorithms, in particular, still have few formal tools for their analysis—investigations of coevolutionary algorithm dynamics are typically empirical in nature [7,3,1,14,16]. The reason for this divide stems from the need to formally account for the defining characteristic of coevolutionary algorithms: the interaction of coevolving entities. We believe that this property of coevolution is fundamentally game-theoretic in nature. Thus, we incorporate notions from evolutionary game theory (EGT) [15] and replicator dynamics [12] into the familiar mixing-matrix [26] and Markov-chain [24] models of (non-coevolutionary) EAs to arrive at a preliminary, yet principled framework for coevolutionary algorithm analysis.

While we are unaware of other formal coevolutionary algorithm models in the evolutionary computation (EC) community, we have discovered at least two precedents that originate, perhaps not surprisingly, from the economics community, which has come increasingly to study agent-based models of economic evolution [4,19]. Both discuss Markov-chain models of genetic algorithms (GA) (i.e., bit-string encodings) where agent fitness is dependent upon population state. While the game structure assumed in the former study is not explicitly stated, the description is consistent with that of the latter study, which assumes an N-player game, where N is the size of the agent population. Such games typify economic domains, in contrast to the two-player games implied by the pair-wise contests frequently used by the EC community.

Coevolutionary dynamics have also been studied from the perspective of mathematical biology, for example in [10] and [5]. The “street-car” theory of the
former study is concerned with reconciling the peculiar constraints of Mendelian genetics with game-theoretic notions of phenotypic stability. In contrast, the latter study presents a continuous-time dynamical systems approach to coevolution that operates strictly in the phenotype realm, and does not assume any constraints due to an underlying genotype structure. Nevertheless, both stress the existence of short-term and long-term time scales in the dynamics. Coming from an entirely different approach, computational learning theory (COLT) has been used to analyze machine learning of competitive (zero-sum) two-player games and construct a specialized form of coevolutionary algorithm [20].

Thus, we see that many sources exist for insight into coevolutionary dynamics. Nevertheless, neither the fields of economics nor biology ultimately concern themselves with modeling coevolutionary algorithms, and N-player and zero-sum games do not represent the entirety of coevolutionary domains. Many important aspects of coevolutionary algorithm dynamics remain to be explicated, for the gap between the hypothesized potential of coevolutionary algorithms and realized practice is substantial—the many successes of coevolution (e.g., [11, 22, 13]) are balanced by many irksome modes of failure that commonly recur (e.g., as discussed in [7, 18]). Investigation into these modes of failure has proven to be a challenge. We believe that an understanding of evolutionary game theory and replicator dynamics will help to meet this challenge.

Our fundamental assumption is that we are using coevolution as an optimization method. This assumption has important game-theoretic consequences in phenotype space that exist independently of any underlying genetic representation. These consequences must be appreciated in their own right before the dynamics of genetical constraints can be taken into account. After describing the game-theoretic extensions necessary to allow mixing-matrix and Markov-chain methods to model the "simple coevolutionary algorithm" (SCA), we move on to demonstrate how a principled game-theoretic approach can illuminate our understanding of coevolutionary algorithms.

2 The Simple Coevolutionary Algorithm

Building on familiar mixing-matrix and Markov-chain models of the so-called Simple Genetic Algorithm (SGA) [26, 24], we describe extensions that utilize a game-theoretic approach to formalize coevolutionary algorithms. Using the notational convention of Vose [24], the SGA heuristic \( G \) is the composition of the function \( M \) (which includes the mixing matrices that implement the variational operators) with the function \( F \) (which includes the objective function, or evaluator), as shown in Equation 1. The input to \( F \) is a column vector \( \mathbf{p} \) in the \( n \)-dimensional simplex \( \Delta \), where \( n \) is the number of possible phenotypes and \( p_i \) is the proportion of agents in the population that have phenotype \( i \). The output of \( F \) is another \( n \)-dimensional vector on the simplex representing the fitness of each phenotype. Assuming fitness-proportionate selection, \( M \) performs the appropriate variational operations to produce the phenotype distribution of the next generation. If the population size is infinity, \( G \) represents a mixing-matrix model.
that is sufficient to study the dynamics of the EA. If the population is finite, however, one must distinguish between population states that are representable, given the population's size, and those that are not. This can be accomplished with a Markov-chain model. Each representable population state is a state of the Markov chain, and each transition probability is the likelihood of a finite-population approximation of the map $G$ to produce the various representable population states.

$$G = M \circ F$$

(1)

Since the distinguishing feature of coevolution is agent interaction, the necessary extensions revolve around the computation of fitness by the objective function. The modifications we describe below are compatible with mixing-matrix and Markov-chain models. In accord with the discrete-time equations of basic evolutionary game theory [15] and much of current EC practice, our modifications assume generational reproduction and domains where agents interact pair-wise (though, steady-state dynamics and $N$-player games are not excluded by our general approach). Coevolving agents may exist in a single population (e.g., [14,1]), or two genetically isolated populations (e.g., [11]).

### 2.1 The Objective Function $\mathcal{F}$

**Single-Population Domains** We abstract the domain of agent interaction into an $n$ by $n$ payoff matrix, $G$, where $G_{i,j}$ gives the expected payoff for an agent of phenotype $i$ when played against an agent of phenotype $j$. We interpret each distinct phenotype as a genetically determined pure strategy for the game (domain) in question. Note that a behavior that blends two or more phenotypes into a “mixed” strategy is not excluded—we simply require it to have a genetic basis and can therefore give it the status of a pure strategy, as well.

The fitness of strategy $i$ represents the fitness value that each agent (and genotype) playing strategy $i$ receives after complete mixing, i.e., pair-wise encounters with every agent in the population (including itself, in the case of a finite population [8]). As shown in Equation 3, we first calculate for each strategy $i$ a weighted sum of the payoffs in row $i$ of $G$, the weights being determined by the phenotypic composition of the population, $p$; this operation is accomplished by matrix multiplication. Since $G$ may contain negative payoff values, we add a constant baseline fitness, $w_0$, to the result of the multiplication so that all fitness values are greater than zero; this is to allow normalization, as performed in Equation 2.

$$F(p) = Gp + w_0 1$$

(3)

$$w_0 = 1 - \min(G)$$

(4)

**Objective Function $\mathcal{F}$**

$$\mathcal{F}(p) = \frac{F(p)}{1 \cdot F(p)}$$

(2)

where ‘$\cdot$’ is inner product.
Two-Population Domains In coevolution between two genetically isolated populations, agents from Population 1 only play against agents from Population 2. Let the column vector $q$ represent the phenotype (strategy) distribution of the second population. The payoffs for agents from the second population are given by an additional payoff matrix, $H$. A pair-wise interaction between an agent (from Population 1) playing strategy $i$, and a second agent (from Population 2) playing strategy $j$, will yield the payoff $G_{i,j}$ for the first agent and $H_{j,i}$ for the second. The calculation of fitness requires two equations, Equations 5 and 6, which assume that every agent in one population plays against every agent in the other population. Because the two populations can have entirely distinct genotype representations, a second set of mixing matrices is required to implement the variational operators. The state space of a two-population Markov chain is the Cartesian product of the two sets of representable population states, $P \times Q, p \in P, q \in Q$.

\[
F_1(q) = Gq + w^G 1 \\
F_2(p) = Hp + w^H 1
\]  

2.2 The Selection Operator $S$

A common assumption of evolutionary game theory, as well as mixing-matrix and Markov-chain models of EAs, is that agents reproduce in proportion to fitness. Nevertheless, many popular alternative selection methods exist that are used both in ordinary EAs and coevolutionary algorithms. We wish our model to accommodate these alternatives. Thus, to Equation 1 we add a new function, $S$, representing the selection operator, as shown in Equation 7. $S$ accepts a vector of normalized fitness values and produces a new normalized distribution of agent proportions.

Fitness-proportionate selection is achieved by making $S$ the identity operator. Linear rank selection, for example, is easily implemented by replacing an agent’s fitness value $f_i$ with a new value based upon the agent’s fitness rank, $r_i$: $S(f) = \frac{R(f)}{15R(f)}$, where $R$ computes the rank of each agent based on fitness, $r_i = n \ldots 1$ (higher fitness results in higher rank values). Implementations of other selection methods for infinite populations are found in [6]. We discuss the effects of alternative selection methods below.

\[
G = M \circ S \circ F
\]  

3 Evolutionary Game Theory

If we are using coevolutionary algorithms for optimization, and understand the coevolutionary domain to be representable by a payoff matrix, then the appropriate optimality concept is that of the Nash equilibrium [2]. This is a strategy that, when used by one player, offers no better alternative to the second player.
than to use the same strategy. If there exists another alternative that does no worse than the Nash strategy, then the Nash strategy is *weak*, otherwise it is *strict*. Thus, classical Nash *equilibrium* is achieved through the assumption of agent rationality.

In a biological context, however, we cannot appeal to rationality to achieve optima. The central contribution of evolutionary game theory [15] is the substitution of agent rationality with Darwinian selection. In EGT, *replicator dynamics* allow Nash equilibria to be achieved. But, we find that only certain Nash equilibria are attractors of the dynamical system, while others are not.

Evolutionary game theory operates exclusively in phenotype space; the replicator equations perform selection only, since no genotype space is assumed upon which to construct variational operators. This fact has been the focal point of debate between Darwinian adaptionists, who have used EGT with great success in analyzing empirical data, and population geneticists, whose mathematics clearly show that Mendelian genetics can easily contradict Darwinian reasoning. Hammerstein’s “streetcar” theory of evolution [10] addresses this apparent paradox and shows how the long-term outcome of coevolution can, in fact, be adequately described on the phenotypic level. Of course, we do not presume this result to apply directly to coevolutionary algorithms, for the genotypic constraints imposed by representations used in EAs differ considerably from those found in Mendelian genetics. Nevertheless, the pressure to achieve optima is clearly found in the selection mechanism, which can be described in a game-theoretic framework. Thus, we suggest that game-theoretic properties must underlie the behavior of coevolutionary algorithms if they are to optimize successfully.

### 3.1 Polymorphic Attractors

Polymorphic *attractors* [15] can occur in single-population coevolution when the game is not constant-sum. A polymorphic attractor is a point attractor of the replicator dynamical system where the population contains more than one phenotype: $\hat{p}_i > 0$ for $\geq 2$ values of $i$. All phenotypes present in a polymorphic attractor $\hat{p}$ (and hence all agents) receive the same fitness: $\forall \hat{p}_i > 0 : f_i = c$. The proportions with which the strategies appear in the population can be interpreted as a mixed Nash equilibrium. Surprisingly, a pure strategy with the same behavior as the polymorphism need not be an attractor of replicator dynamics [15]. This is because replicator dynamics act upon individual fitness rather than population-wide fitness.

Thus, we see that the simple coevolutionary algorithm, as with the simple genetic algorithm, operates with an *agent-centric* view of performance. This makes recognizing (as opposed to achieving) polymorphic attractors problematic. For example, the Hawk-Dove game is a simple two-strategy, non-constant sum game that has a single, and polymorphic, attractor where $\hat{p}_{Hawk} = \frac{7}{12}$ and $\hat{p}_{Dove} = \frac{5}{12}$. How is this equilibrium to be comprehended by an agent-centric view? To begin, sorting the population of agents by fitness has an ill-defined result due to ties—all agents receive the same fitness, and the “best” agent, with respect to sorting, could be either a Hawk or a Dove. Of course, neither strategy
alone constitutes an optimal solution. Not only are both strategies needed, but they are needed in a particular proportion. The "solution" to the Hawk-Dove game is not a single strategy, but an ecology.

We may argue, then, that a coevolutionary algorithm should report the population's phenotypic proportions once fitness converges. This is straight-forward if the genotype-phenotype mapping is known to be one-to-one. But, with real-world representations, a particular phenotype is often achievable through a variety of genotypes. Since we only have knowledge of an agent's genotype, the question of phenotypic equivalence between two agents is generally undecidable. How, then, are we to distinguish between fitness convergence with phenotypic polymorphism, on the one hand, and genuine phenotypic convergence, on the other? This distinction is not important in the Simple Genetic Algorithm—any optimal phenotype will do. But, in coevolution, polymorphic optima are achieved through specific phenotype proportions, which must remain intact.

3.2 Time Scales of Selection and Search

Given an initial population state, $p^0$, what will iteration of the selection operator on $p^0$ (without variation) do in the limit? Because variation operators are not applied, we know that phenotypes that are absent in $p^0$ cannot appear later. Also, we know that some phenotypes present in $p^0$ may be eliminated by selection. In ordinary evolution, the limit behavior is the point attractor $\hat{p}$, which is composed exclusively of the most fit phenotype in $p^0$ (assuming the absence of neutrality). In coevolution, however, the limit behavior can be very different: assuming fitness-proportionate selection, the attractor may be a limit cyclic rather than a fixed point. Further, if a point attractor $\hat{p}$ exists and the game is not constant-sum, then $\hat{p}$ may easily be composed of a phenotype other than the most fit phenotype in $p^0$.

Linear stability analysis [23] tells us that the strategies in support of a coevolutionary point attractor—that is, all strategies $i$ where $\hat{p}_i > 0$—must be the highest scoring strategies in all population states within some epsilon of the attractor. Otherwise, the attractor could not be locally stable. But, the region within which this is true need not be the attractor's entire basin of attraction. Indeed, population states that are outside of the epsilon, $|\hat{p} - p| > \epsilon$, but still within the basin of attraction, may contain very many phenotypes that out-score those in support of $\hat{p}$, but which eventually drive each other to extinction, leaving only the strategies in $\hat{p}$. Fitness values in this region of population-state space are "deceptive" in the sense that they favor phenotypes that will eventually be selected against.

For example, in randomly generated non-constant sum games of 100 strategies, we can easily find examples of an initial population state, $p^0$, that leads to an attractor composed of a single strategy, $i = Win : \hat{p}_i = 1.0, Vi \neq Win : \hat{p}_i = 0$, but that exhibits fitness deception at the beginning of its orbit towards the attractor. In one instance of $p^0$ that we have found, the eventual "winner" strategy $\hat{p}_{Win}$ is actually out-scored by 54 of the 100 game strategies and over 50% of the population in $p^0$, as well as in the next ten generations (iterations
of selection), \( p^1 \ldots p^{10} \). Indeed, some 39 generations are required before \( \hat{p}_{\text{win}} \) becomes the most fit in the population. Other initial conditions in the neighborhood of \( p^0 \) yield similar behavior. This result is in stark contrast to the dynamics of selection in ordinary evolution, where the winning strategy of the attractor \( \hat{p}_{\text{win}} \) is made apparent after a single round of evaluation.

This example raises a concern about the efficiency with which conventional coevolutionary algorithms may perform search in non-constant sum games. In each generation, decisions are made regarding which strategies should be kept and form the basis for future search; these decisions can easily be misled by fitness values. Note that this form of "deception" does not involve the genotype space at all (as it does in GA trap functions, for example)—it exists purely within the game-theoretic relationships of phenotype proportions. Strategies that exhibit superior performance only in transient population states, \( p^{0\ldots n} \), may be less useful guideposts for search than the strategies in \( \hat{p} \), which at least have some stability properties with respect to \( p^{0\ldots n} \). Should the application of variation operators be delayed until fitness values are "believable"? If so, how should the coevolutionary algorithm be modified to improve efficiency yet avoid premature convergence?

3.3 Dynamics of Alternative Selection Methods in Coevolution

The constraint of finite populations creates at least three problems for the canonical fitness-proportionate roulette-wheel selection method used in EAs. First, the multinomial distribution of roulette-wheel operation introduces a great deal of noise into the selection process. Second, this noise makes resolving small differences in genotype fitness impossible for realistic population sizes. Third, an excessive difference in fitness between the best individuals and the rest of the population can cause premature convergence. These problems have lead to the development of many alternative selection methods for evolutionary algorithms, such as truncation selection, \((\mu, \lambda)\)-ES selection, and ranking [17]. We have used EGT to test these selection methods and have shown that none of them are able to attain polymorphic attractors, even if the population is infinite [6]. Instead, these selection methods exhibit attractors (sometimes chaotic) that are unrelated to the game. Thus, implementation decisions that may be sound for ordinary evolutionary algorithms can be pathological in an idealized coevolutionary context (where the search problem is solved, leaving only the problem of selection), and therefore dubious in a real-world algorithm. Indeed, we have shown [8] that the use of truncation selection is one reason why the simulations by Fogel, et al [9] lead them to claim that EGT loses predictive power in finite populations.

3.4 Population Structures

Coevolutionary algorithms that use two genetically isolated populations typically involve asymmetric games, where agents from one population can only play against agents from the other [11,7]. Evolutionary game theory tells us that
asymmetric games never result in polymorphic equilibria [12]. This feature is actually not a property of the games themselves, but rather a property of the two-population structure, which asymmetric games happen to require. Thus, if we use a two-population structure on a symmetric game that has a polymorphic attractor, then the polymorphic attractor will disappear. In the case of the Hawk-Dove game, a single population of agents yields the polymorphic attractor of $\frac{7}{12}$ Hawks and $\frac{5}{12}$ Doves. But, when the game is played between two separate populations, one population will converge to all Hawks and the other to all Doves.

Multiple-population structures are often used in evolutionary algorithms (including coevolution) to maintain genetic diversity (e.g., [22]). In a coevolutionary context, however, the influence of population structure can transcend the process of genotypic search to exert an independent (and unintentional) effect on the process of phenotype selection. Evolutionary game theory shows that, even when the search problem is solved a priori, different population structures may lead to different outcomes, even though the game remains the same.

3.5 Local Optima

When variational operators are unable to reach new phenotypes that are sufficiently good to survive selection with respect to the current population (and thereby allow search to continue), the state of the evolutionary system is at a local optimum. In stationary fitness environments, local optima are easy to conceptualize. In coevolutionary domains, however, fitness landscapes are dynamic. In this case, game theory provides a natural way of describing a coevolutionary local optimum: it is a search-constrained attractive Nash equilibrium. That is, the coevolutionary state is a "best reply" to itself only to the extent that all locally reachable population states are inferior to the Nash state—all paths to superior population states are blocked by the convergent force of selection.

3.6 Dominating Strategies

An important concept in game theory is that of dominance [2]. Strategy $s_a$ (pure or mixed) is said to dominate strategy $s_b$ if $s_a$ does as well or better than $s_b$ against all strategies in the game and strictly better against at least one: $s_a \succ s_b \Leftrightarrow \forall i : G(s_a, s_i) \geq G(s_b, s_i) \land \exists j : G(s_a, s_j) > G(s_b, s_j)$. If $s_a \succ s_b$, but there exists some strategy against which $s_a$ does no better than $s_b$, then $s_a$ weakly dominates $s_b$. Otherwise, $s_a$ strictly dominates $s_b$. If $s_a$ is a pure strategy and strictly dominates $s_b$, then the replicator dynamics will remove $s_b$ from the population. This is not necessarily true for weak dominance [12].

That replicator dynamics removes strategies that are strictly dominated (by pure strategies, in the case of non-overlapping generations [25]) suggests that it is performing an operation akin to multi-objective optimization, where every strategy represents an objective of the coevolutionary problem. That is, in comparing two strategies, $s_a$ and $s_b$, the salient question is not whether $s_a$ beats $s_b$ in the game, or out-scores $s_b$ when played against a series of opponents, but
rather whether $s_a$ Pareto dominates $s_b$ with respect to a given set of opponent strategies (including themselves). (The notion of Pareto dominance, as used in multi-objective optimization, is not to be confused with Pareto dominance with respect to game payoffs, as used in game theory.)

This observation leads to the realization that multi-objective optimization techniques may be explicitly applied to coevolutionary domains without the use of replicators, such that both strictly as well as weakly dominated strategies are rejected in favor of Pareto-optimal strategies. Though such an approach may no longer qualify as a coevolutionary algorithm, we suspect that it may nevertheless provide a powerful method of optimizing coevolutionary domains.

4 Conclusion

We have argued for the relevance of evolutionary game theory to the study of coevolutionary algorithms. We have presented game-theoretic extensions that allow mixing-matrix and Markov-chain models of EAs to address coevolutionary algorithm dynamics. Additionally, we have shown that a game-theoretic view of coevolutionary algorithm design and operation reveals that the simple coevolutionary algorithm 1) cannot recognize polymorphic equilibria, 2) can be made inefficient by a newly-recognized form of fitness deception, 3) exhibits pathological behaviors if certain commonly-used selection methods are employed, and 4) can have different attractors depending upon population structure. Finally, we have used game-theoretic concepts to formulate a notion of coevolutionary local optima, and to understand coevolution as a form of multi-objective optimization. We believe that EGT can help with the design and validation of new search methods for coevolutionary domains. This is the subject of our current work.

5 Acknowledgments

The authors thank members of the DEMO Lab, particularly Richard Watson (for fruitful discussions regarding connections between coevolution, multi-objective optimization, and Pareto optimality), and Ofer Melnik.

References


The Number of People with Whom a Man Interacts

Masao KUBO, Hiroshi SATOH, Yuichiro INOUE, Kimitaka UNO, Akira NAMATAME

Dept. of Computer Science, National Defense Academy, Yokosuka Japan, 239-8686
masaok@nda.ac.jp, hsato@nda.ac.jp, nama@nda.ac.jp

Abstract. Recently, there has been growing interest in the interactions among persons. It is needed to consider how a man behaves in the society where the number of people with whom he has to interact is varying. In this paper, we model the society with agents who play iterated prisoner’s dilemma game and show that an appropriate interaction size is needed to emerge cooperative behaviors through computer simulations.

1 Introduction

The interaction between human has been investigated by many researchers [1] [3] [2] [4]. We now need to consider what behavior we should take in the society where one’s behavior is subject to the other’s action, especially, if the number of agents is varying. Intuitively, it seems to be good to interact other persons as many as possible. However, there may be an appropriate size of interaction in some type of society. The purpose of this paper is to investigate the dynamics of the society through computer simulations.

We assume that a person behaves rationally and his behavior is subject to payoff that he will get. The iterated prisoner’s dilemma is the model that is used to study these situations. In two person iterated prisoner’s dilemma (2IPD) game, two players may select cooperation(C) and defect(D). Table 1 shows the payoff matrix of 2IPD. This is a non zero-sum game and players do not know when they will finish the iterations of the game.

<table>
<thead>
<tr>
<th>Own's Strategy</th>
<th>$S_1$ (Cooperate)</th>
<th>$S_2$ (Defect)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$ (Cooperate)</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>$S_2$ (Defect)</td>
<td>S</td>
<td>P</td>
</tr>
</tbody>
</table>

Table 1. Payoff matrix of the 2IPD game, ($T > R > P > S, 2R > T + S$)
If this game is played only once, "D" is the optimal behavior. However, if the game is played many times, "D" is not the optimum any more. There are many researches in various environments [1] [3]. It is not reasonable to discover the best strategy because human can change his mind when he get information through interactions. We adopt mechanism that enable agents to change their decision responding their payoff in order to make agents in IPD adaptive.

Ordinarily, a person interacts with many persons. There is an approach that uses N-players IPD (NIPD) for interactions among many person [9]. NIPD deals with the game (for example, congress) in which N players interact all at once, then the payoff is subject to the number of agents who choose "C". Yao et. al. state that the cooperation rate is inversely proportional to the number of the players [9].

In contrast, we extend 2IPD for interactions among many person and adopt the payoff matrix given by Table 1 in this paper, because we are interested in different situation in which players interact individually (one to one interactions).

In addition to using the payoff matrix on Table 1, we restrict the interactions with localization; agents play the game only with their neighborhood. The size of neighborhood $f$ is very important, since the spreading speed of information is depend on the size. If $f$ is small, two agents who are widely separated are almost independent, on the other hand, if $f$ is large, the effects of an agent's behavior is far and wide. Although every person has its own neighborhood size in the real world, we set all agents have the same $f$ value because we have to seek the appropriate size of $f$ as first step.

The speed of producing a new strategy is also important. Agents change their strategies based on the behaviors of other agents. Old strategies can go through the population if new good strategies are not produced in their way. We use genetic algorithm to produce new strategies.

The remainder of this paper is presented as follows. In section 2, we define players and society; representation of strategies, mechanism of changing behaviors and measuring cooperation rate. Section 3 shows some empirical results and discuss them. Computer simulations are done in several size of $f$ (2, 4, 8, 16, 24, 40, 80). The results show a dramatic change of dynamics at some $f$ value. This is different from other existing research results.

2 The Society

In this section, we firstly define the strategies of players. The strategies are represented as if–then rules. Secondly, payoff matrix, measurement of cooperation level and neighborhood structure are described. Finally, complete definition of an player is presented and ability of adaptiveness is discussed.

2.1 Encoding Strategy for the NIPD Game

The players of IPD are defined as [5]. They can select cooperate (C) or defect (D) per a bout and repeat this game over again. The players choose their next
move taking into account the series of previous games. Let us consider \( a(t) \) as

\[
a_i(t) \in \{0 \text{ (Cooperate)}, 1 \text{ (Defect)}\},
\]

then the strategy is represented as a rule set. For the NIPD game remembering \( n \) previous steps, there are \( 2^{2n} \) combinations of possible histories. Therefore, at least \( 2^{2n} \) bits are needed to represent a strategy. Besides, agents need more \( 2n \) bits for an early stage of iterations in which the size of their history is too short to use their rule. Thus, we represent the strategy as \( 2^{2n} + n \) bits.

In this paper, an agent interacts with their neighborhoods and each pair of agents plays the games 20 times. The size of history is two \((n = 2)\). Table 2 shows the strategy of TFT agent. Let us call \( s_{ik} \) strategies of \( i \) th agent, where \( k \) denotes the one of the possible history. For example, if the history of the game is \(((C, D), (C, C))\), then agent \( i \) choose “C” according to the position where \( k = 8 \) \((0100 = 8)\).

| \( s_{ik} \) | \( t = 1, 2 \) | memory | \( 0 \ | 1 \ | 2 \ | 3 \ | 4 \ | 5 \ | 6 \ | 7 \ | 8 \ | 9 \ | 10 \ | 11 \ | 12 \ | 13 \ | 14 \ | 15 \) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | C | . . . | C | D | C | D | C | D | C | D | C | D | C | D | C | D |

| Table 2. Example of player \( i \) (TFT) |

2.2 Payoff Matrix

We model the N-player’s interactions as 2IPD with N-players, so the profit of an agent is sum of each 2IPD games. The payoff matrix of 2IPD is shown by Table 1. Prisoner’s dilemma is subject to following two conditions; (1) \( T > R > P > S \), (2) \( 2R > T + S \). In our experiments, \((R, S, T, P) = (3, 0, 5, 1)\).

Let us denote the profit of an agent \( i \) who play games with agent \( j \) at time \( t \) as \( p(a_i(t), a_j(t)) \). The payoff of \( \tau \) times iteration with same agents is

\[
\sum_{k=t-\tau}^{t} p(a_i(k), a_j(k)),
\]

and the sum of the payoff which is gathered from the interactions with \( f \) neighborhoods is

\[
\sum_{j} \sum_{k}^{t} p(a_i(k), a_j(k)).
\]

Next, we define the cooperation rate of population. This is the ratio of cooperate actions to all actions [7][8]. Assume that agent \( i \) has played the game \( b \)
times at moment $t$, the average cooperation rate $r_i(t)$ is

$$r_i(t) = \frac{(b - \sum_{k=1}^{h} a_i(t - k))}{b}. \quad (4)$$

In this paper, we set $b = f r$. The cooperation rate of entire population is written as follows,

$$R(t) = \frac{\sum_{i}^{N} r_i(t)}{N}. \quad (5)$$

### 2.3 Localization and Player’s Definition

We now define the neighborhood structure. There are a number of research that deal with neighborhood structure in IPD [2] [9]. For example, ring, Moore, Gaylord and so on. In this paper, agents are settled at each grid of a plane. Neighborhood structure is defined considering the symmetry because the speed of spreading information should be uniform.

Let us assume that the width of the plane is $W$ and the position of agent $i$ is $(x_i, y_i)$. We then define the projection $\delta$ of agent $i$ as follows,

$$\delta | W \times W \rightarrow A, \quad A = \{A_i\}, \quad |A| = N = W^2, \quad (6)$$

$$A_i = (s_i, m_i, p_i(t), r_i(t), x_i, y_i, H_i, G_i)$$

$$s_i = \{s_i\}, \quad m_i = \{0, 1 \}^l,$$

where $A$ is the set of players and its size $N$ equals to $W^2$. The $i$th agent $A_i$ consists of strategy $s_i$, memory $m_i$ which stores history of $l$ times plays, profit $p_i(t)$, average cooperation rate $r_i(t)$, position $x_i, y_i$, the function $G_i$ that changes his strategy and neighborhood structure $H_i$. Agents select one neighborhood structure among the set of them.

The neighborhood structure is comprised of relative positions from one agent. $H_u$ is defined as follows,

$$\psi|\{1, \ldots, f\} \rightarrow H, \quad H = \{H_u\}. \quad (7)$$

$$H_u = \{H_{us} | H_{us} = (H_{usx}, H_{usy}), \quad H_{usx}, H_{usy} \in W, \quad |H_{us}| = f \},$$

$$\forall s_1, s_2 H_{us1x} \neq H_{us2x} \quad \text{or} \quad H_{us1y} \neq H_{us2y},$$

$$\{H_{us} \in H_u | H_{usx} = 0 \text{ and } H_{usy} = 0 \} = \phi,$$

where $\psi$ is an injection from $f$ to $H$. $H_u$ is an element of $H$, and is comprised of the set of relative distance $(H_{usx}, H_{usy})$ except $H_{usx} = H_{usy}$.

Suppose that opponents of agent $i$ is determined by $H_{is}$. The opponents $A_o$ is represented as

$$A_o = \delta(\text{mod}^W(H_{isx} + x_i), \text{mod}^W(H_{isy} + y_i)), \quad (8)$$
where \( \text{mod}^W \) returns the remainder divided by \( W \). Equation 8 is a feasible at the edge of the plane; leftmost agents interact rightmost agents and the agents at top interact agents at bottom, and vice versa.

Figure 1 shows some neighborhood structures up to \( N \leq 16 \). For larger size, neighborhood structures are given as follows,

\[
H_u = \{ H_{us} | H_{us} = (H_{usx}, H_{usy}), \quad f = |H_{us}| = d^2 - 1 \} - (0, 0),
\]

where \( H_{usx} \in \{ -d, \ldots, 0, \ldots, d \} \), \( H_{usy} \in \{ -d, \ldots, 0, \ldots, d \} \).

In this paper, \( f = 24, 48, 80 \) using \( d = 2, 3, 4 \).

\[\text{Fig. 1. Neighborhood structure}\]

### 2.4 Learning Strategies of the Players

Learning ability is one of the most important issues for agents who wish to adapt dynamically to changing society. A person learns behavior of his friends who go about with well. Also, he take care not to make same mistakes which his friends make. We refer these examples as one of the learning by imitation. Ohno said that "copy strategy is the most important law of the world. This is applicable for gene, human, etc ..." [6], and we agree the opinion. Thereby, we adopt Uno's method [7][8], which is one of the copy strategy in this research area.

Uno proposed two learning strategies inspired from the imitation. He named his strategies Complete Mimicry strategy (= copy) and Mutant strategy (= crossover) respectively. Both procedures were examined but the following experiments used only the mutant strategy only. Therefore, the procedure of the mutant strategy for player \( i \) is described as follows.
(Step1) Select a player who scores highest profit $p(t)$ than other its competitors.
(Step2) Apply one-point crossover operation to $s_{ik}$ and $s_{jk}$. Set one of the new offsprings to its new string candidate.
(Step3) Apply mutation operation to the candidate $s_{ik}$. Each $s_{ik}$ is examined whether the operation is applied or not. Note that the probability per bit is called as mutation rate. If the operation is executed, the value is flipped.
(Step4) $p(t) = 0.0$ $r_i(t) = 0.0$

In the following experiments, the mutation rate is one of 0.1, 0.01, 0.001 and 0.0001.

3 Simulation Results

We set the grid width $W$ to 20 and the number of the players $N$ to 400 (see 2.3). The parameter $f$ and the mutation rate of all players were set equally. Their initial strategies were defined randomly and each run was executed as the following sequence that we call a turn. Firstly every player competes with its neighborhoods (the number of the player is $f$). Secondly, all players update their strategies. This sequence, a turn, was repeated 3,000 times. The 48 societies were examined ($f$=2, 4, 8, 16, 24, 48, 80 $\times$ mutation rate $\mu = 0.1, 0.01, 0.001, 0.0001$).

Figure 2, 3, and 4 show the results. The x-axes show the turn, the y-axes indicate the cooperation rate, $R(t)$. Figure 2 illustrates progress of $R(t)$ of small $f$ societies. Figure 3 and 4 present the societies with $f$ = 24, 48, 80. These graphs suggest the following remarks.

![Figure 2. $f = \{2, 4, 8, 16\}$](image)

1. The society will not reach high cooperative state if its mutation rate is high. In figure 2, several graphs exist from 0.2 to 0.4. It means that it was difficult for a society which $f$ is smaller than 24 and its mutation rate $\mu$ was high, for example, $\mu = 0.1$, to shift to cooperative state (over 0.8). Among this group, the more the $f$ enlarged, the higher cooperation it achieved.
In figure 3 and 4, there is a band around 0.2 like the last figure. It remarks that cooperation of the remainder of the systems with high mutation rate, namely, $f$ is larger than 16, was low extremely.

2. The societies with mutation rate 0.001 and 0.0001 tended to keep high cooperative state. Figure 5 illustrates the progress of the cooperative societies with $f = 2, 4, 8, 16$. They scored higher than the last group. Especially, among the group, the 2 societies achieved higher than the others did. The one is the society with $f = 24, \mu = 0.01$. The $f$ and mutation rate of the other is 80, 0.001, respectively. However, the figure manifests that these were not stable consistently and their cooperation rates were fluctuated from 0.92 to 0.8. The fluctuation of the society with $f = 24$ is small as same as the societies with $f = 8, 16$. On the other hand, magnitude of the fluctuation of the society with $f = 80$ was larger than the others shown by figure 6, in spite that this is one of the 2 highest cooperative societies. It expressly provides that the society needed much time until it reached cooperative state and the state was broken down. On the other runs which we do not show because the space limitations, this kind of breakdown phenomena was observed frequently. Also, the cost was sometimes larger, sometimes smaller.
Therefore, we conceived that it needed to compare the number of the breakdown and the cooperation rate for classification of societies. Hence, we plotted 2-tuple, variance and average of $R(t)$. The $x$-axes and $y$-axes of figure 7, 8, and 9 mean the average of $R(t)$, the variance of $R(t)$, respectively. Note that there is a difference in meaning among these graphs. In figure 7 and 8, points that have same $f$ were connected. For example, a line was drawn between the point ($f = 4$ and $\mu = 0.01$) and the point ($f = 4$ and $\mu = 0.001$). On the other hand, in figure 9, points which has same $\mu$ value were connected. Moreover, take care that the ranges of $y$-axes between figure 7 and 8 are different. Figure 8 is a macrograph of figure 7 and it ranges within smaller interval. Therefore, in figure 8, the graphs of societies with $f$ that is over 16 were vanished. Using by this analysis, the following results are highlighted.

1. When $f$ is very small (e.g. $f = 2$), the cooperation rate is not improved.
2. If the $f$ is gradually increased ($f = 4, 8$), the cooperation rate is improved. Within this range of $f$, the smaller the mutation rate is, the smaller the variation is.
3. If the $f$ enlarges over the level described at second remark, the variation will be high suddenly ($f = 16, 24$). In figure 7, the variation at $f = 16$ and $\mu$
= 0.0001 is larger than that of $\mu = 0.001$. Also, the variation of $f = 24$ in figure 8 is suddenly enlarged at $\mu = 0.0001$.

4. Mutation rates that could produce cooperative society before can not work, if $f$ is larger than the level denoted by the last remark. The cooperative rate is quickly converged to 0.0. Of course, if the mutation rate is too small and $f$ is huge, the initial state of society makes big influence to its state of future.

These experiments suggest that there is a critical neighborhood size $f$ and critical mutation rate $\mu$ to emerge cooperative society. According to the traditional approach, the larger $f$ is, the more difficult a society reaches a cooperative state.
4 Conclusions

To study about the interaction among persons, we considered how a man behaves in the society where the number of people with whom he has to interact is varying. In this paper, we model the society with agents who play 2-player iterated prisoner's dilemma game with his n-players around him. From several computer simulations, we showed that an appropriate interaction size and an appropriate learning speed are needed to emerge cooperative behaviors.

References


Fig. 9. The effects of mutation rate
A^J?^-landscapes as Test Functions for Evaluation of Host-Parasite Algorithms

Björn Olsson
Dept. of Computer Science, University of Skövde, Box 408, 541 28 Skövde, Sweden
bjorne@ida.his.se

Abstract. Previous research on host-parasite algorithms has shown that the co-evolutionary arms race is difficult to sustain when the tasks faced by hosts and parasites are heavily asymmetric. We have therefore proposed an asymmetry-handling algorithm, AHPA, with a capacity for self-adapting the allocation of generations to hosts and parasites, so that the problem asymmetry is counteracted. In this paper we discuss the need for systematic evaluation of this algorithm, so that its behaviour under varying levels of asymmetry can be studied in detail. We propose the use of Kaufmann's NK landscape model for this purpose, and show how the model can be adapted for the testing of host-parasite algorithms. Using the adapted model, we present simulation results which confirm AHPA's ability to sustain a stable arms-race under varying levels of asymmetry.

1 Introduction

Host-parasite algorithms use co-evolution of solutions and test cases in two populations, where fitness of a candidate solution is based on the number of test cases it solves, and fitness of a test case is based on the number of candidate solutions which fails to solve it. This idea was introduced by Hillis in [2] along with promising results in evolving sorting networks. It has later been used for a number of applications, including cellular automata [8] [12], tic-tac-toe endgame classification [8], game playing [15] [16], and protein sequence classification [10].

In previous research, very little effort was made to systematically evaluate the behaviour of host-parasite algorithms, and we therefore attempted in [11] to find an approach for such evaluation. Our efforts were focused on the adaptation of standard test functions for use with host-parasite algorithms. This paper explores the subject in more detail, and proposes the use of Kaufmann's NK model for this purpose. We show how the standard NK model can be modified and adapted for the study of host-parasite algorithm behaviour.

Our previous work in [8] identified problem asymmetry as an important factor limiting the usefulness of simple host-parasite algorithms. In [9] we proposed an extended host-parasite algorithm, AHPA, with ability to adapt the evolutionary process to the level of problem asymmetry. We also presented initial promising results on evolving sorting networks, which indicated that significant improvements over simple host-parasite algorithms are possible. In [11] we discussed the need for systematic evaluation of the effects of problem asymmetries,
and analysis of the behaviour of different types of host-parasite algorithms. We proposed the use of function optimization tasks for this purpose, and we were able to show how the test functions used in EA research can be adapted for evaluation of host-parasite algorithms. Our results indicated that AHPA consistently gives improvements over simple host-parasite algorithms at intermediate levels of asymmetry. In this paper we extend the previous work by proposing the use of Kaufmann’s NK model for a more thorough study of how asymmetry affects host-parasite algorithms.

2 Asymmetric problems

The term symmetry means that two opposing sides of an object are exactly the same in size, shape, form, etc, so that they are “mirror images” of each other. For a host-parasite algorithm to be fully symmetric, the two opposing populations would have to be exactly the same regarding population size, chromosome representations, genetic operators, etc. Similarly, for the problem to which the algorithm is applied to be fully symmetric, both populations would have to be solving the same task. This can be the case in fully symmetric games, where the fitness cases can be opponent strategies and use the same representation scheme as candidate solution strategies. However, many of the problems to which host-parasite algorithms are applied can be shown to be asymmetric. In [11] we provided an analysis of the sorting networks application introduced by Hillis [2], showing that it is in fact a heavily asymmetric problem since the sizes of the search spaces are very different. For 16-input sorting networks of length 60-120 there are $4 \times 10^{126}$ candidate solutions, while the set of input sequences only need to contain $2^{16}$ bit strings for complete testing.

The experiments described in this paper will compare three different algorithms and test their ability to handle different levels of asymmetry. The algorithms can be described briefly as follows.

**CEA, Canonical Evolutionary Algorithm**, represents traditional evolutionary algorithms and uses a single population. It does not use co-evolution, and for fitness testing CEA draws random fitness cases from the problem domain. CEA uses tournament selection. In the experiments presented in this paper, CEA is a standard bit-string genetic algorithm implemented by slight modification of the “Simple Genetic Algorithm” (SGA) from [1].

**SHPA, Simple Host-Parasite Algorithm**, represents a very simple form of co-evolution. SHPA uses separate populations for candidate solutions (“hosts”) and fitness cases (“parasites”). Fitness of a host is determined by counting the number of parasites (fitness cases) it solves, while fitness for a parasite is determined by counting the number of hosts which fail to solve it. Selection for reproduction is done by tournament selection in each population.

In every generation, SHPA performs selection and reproduction in both populations. As pointed out in [9] and [11] this reflects a hidden assumption that an equivalent number of generations is useful for both populations. This assumption is perhaps likely to be valid for symmetric problems, where hosts and parasites
face problems of the same difficulty, but it is not likely to be valid for asymmetric problems. We would therefore like to allow the algorithm to self-adapt the number of generations it uses for each population.

**AHPA, Asymmetric Host-Parasite Algorithm**, contains the mechanism for self-adaptation that is missing in SHPA. In AHPA, selection and reproduction is applied to one population at the time, while the other population is kept unchanged. Evolution of hosts is performed until a host $h^*$ is found which solves every current parasite. AHPA then saves $h^*$ and switches focus to the parasite population, evolving parasites until a fitness case $p^*$ is found which no current host solves. AHPA then saves $p^*$ and switches back to evolution of hosts.

Fig. 1 shows an overview of AHPA (for a more detailed version, see [11]). It uses two sub-populations each for both hosts and parasites. The sub-populations $H^e$ and $P^e$ are evolvable, meaning that they undergo selection and reproduction. The other two sub-populations, $H^s$ and $P^s$, are static, in the sense that they are used for saving the $h^*$ and $p^*$ which are found during the search. Fitness for a host in $H^e$ is determined using $P^e \cup P^s$ as fitness cases, so that the host is tested not only against the current evolvable population of parasites, but also against previously found challenging fitness cases. This mechanism is similar to the “Hall of Fame”, as described in [14]. In AHPA, the static populations are implemented as queues with a maximum sub-population size, where the oldest individual is deleted when a new one is added. For more detailed descriptions of CEA, SHPA, and AHPA, see [8], [9], and [11].

```
t = 0
initialize random $H_t^e$ and $P_t^e$
$H_t^e = P_t^e = \{}$
while ($t < \text{max generations}$) do {
    do evaluate and reproduce $H_t^e$ using $P_t^e \cup P_t^s$ as fitness cases
    until ($\exists h_t((h_t \in H_t^e) \land (h_t \succ (P_t^e \cup P_t^s)))$
    do evaluate and reproduce $P_t^e$ using $H_t^e \cup H_t^s$ as fitness cases
    until ($\neg((H_t^e \cup H_t^s) \succ P_t^e))$
    find $p_t$, such that $((p_t \in P_t^e) \land \neg(h_t \succ p_t))$
    $H_{t+1}^e = H_t^e \cup h_t$
    $P_{t+1}^s = P_t^s \cup p_t$
    $H_{t+1}^s = H_t^s$
    $P_{t+1}^e = P_t^e$
    $t = t + 1$
```

**Fig. 1.** Pseudo-code of AHPA. The notation $h \succ p$ represents that host individual $h$ solves the parasite indivual $p$, while $h \succ P$ means that $h$ solves every parasite in the population $P$, and $H \succ P$ means that every host in $H$ solves every parasite in $P$. 
3 The NK Landscape Model

The NK model is a family of tunably rugged fitness landscapes, introduced by Kaufmann in [3] and [4] and used in, for example, [6] and [5] to study the dynamics of evolutionary adaptation. They have been used frequently in EA research as model landscapes for testing how different EAs behave in the presence of varying levels of landscape ruggedness (see for example [7] or [13]).

The general idea in the NK model is to define a set of $N$-dimensional landscapes in which $K$ represents the level of ruggedness. A particular NK landscape corresponds to a genotype representation using $N$ genes, where the contribution from each gene depends on itself and $K$ other genes. The value $F$ of an individual with chromosome $x$ given a particular NK landscape $NK_j$ is set to the average of the contributions from all of the individual’s genes:

$$F(x|NK_j) = \frac{1}{N} \sum_{i=1}^{N} c(x_i)$$

The contribution $c(x_i)$ of gene $i$ is determined from a randomly created lookup table of $N$ columns and $2^{K+1}$ rows. An example of such a lookup table is given in Fig. 2 for a landscape with $N = 5$ and $K = 2$.

By varying the parameter $N$ we can create landscapes of different sizes, and by varying the parameter $K$ we can adjust the level of epistasis, i.e. the degree to which the contribution of each allele depends on its context. The degree of epistasis determines the ruggedness of the landscape, i.e. the number of local optima. At one extreme, we have the very simple landscapes of $K = 0$, where the contribution of each gene is independent of all other genes, and which are solved trivially by hill-climbing algorithms. Such landscapes are said to be highly correlated, since the fitness of any genotype is highly correlated with the fitness of all its one-bit mutants. At the other extreme are landscapes with maximum ruggedness where $K = N - 1$. Such landscapes are fully uncorrelated, since the fitness of any genotype says nothing about the fitness of any of its one-bit mutants. In between these extremes are landscapes of varying ruggedness, and the NK model is therefore a useful tool for studying the behaviour of an EA under varying levels of epistasis.

In our case, we will use the NK model to study to what degree host-parasite algorithms are sensitive to asymmetries in the degree of ruggedness and the size of the search spaces. We will use two populations, where each is evolving on a separate NK landscape. The landscapes are here viewed as functions which we try to maximise, and fitness of an individual will be based on the number of individuals from the other population which it “beats” in terms of its NK function value. The experiments will allow us to compare host-parasite algorithms with CEA, and to see to what degree the algorithms are sensitive to asymmetries between the two landscapes. To investigate the effects of asymmetries, we will run experiments where we vary the parameters of the two landscapes independently.

As Fig. 2b shows, the landscape for $N = 5$ has a global maximum of relatively low fitness. The exact value of this maximum depends on chance, since all
Fig. 2. NK landscapes for $N = 5, K = 2$. Left: Landscape where each gene is assigned a random fitness contribution for each configuration of the genes to which it has epistatic couplings. Right: Normalised landscape with global optimum at genotype 11111.

fitness contributions are randomly chosen. However, the global maximum will increase for higher values of $K$. This is because each gene has more epistatically coupled neighbours, and therefore more allele combinations to choose from, so that expected maximum fitness contribution is higher. For our purposes, this is problematic because we want to assign the host and parasite landscapes different levels of epistasis, but as a side-effect they would also get landscapes with global optima of different heights. If the parasite landscape has a higher global optimum than the host landscape, a parasite could evolve which is unbeatable by any host, which would cause AHPA to become stuck. To see this, consider the first of the inner do-until-loops in the pseudo code of AHPA in Fig. 1. Execution stays in this loop until a host $h_t$ is found, which "beats" all current parasites. In other words, if the parasite landscape has an optimum which is higher than the global host optimum, execution may become stuck in this loop.

To overcome these problems we create normalised NK landscapes, which are guaranteed to have a global optimum of 1.0, as can be seen in the example lookup table in Fig. 2. In the normalised table, the value of the contribution $c(i)$ of every gene in row $j$ is recalculated to

$$c(i) = c(i) + \frac{j}{2^K + 1} (1 - c(i))$$

In the last row of the table, all the values will be 1.0, so that all landscapes will contain a globally optimal genotype having fitness 1.0 and consisting of all 1s.

4 Results

All experiments presented in this section use a population size of 50, one-point crossover with a crossover probability of 0.7, a mutation rate of 0.01 per bit, and tournament selection. The selection is performed by picking two random individuals, and choosing the one with the highest NK function value.

Fig. 3 compares the average results of the two host-parasite algorithms with CEA on landscapes with $N = 100$ and four different levels of epistasis $K = 1, 3, 7, 11$. Both SHPA and AHPA use population sizes of 50 for both hosts and
parasites, and each population evolves on a separate $NK$ landscape, with fitness for an individual being set to the number of individuals in the opposing population having a lower function value on its landscape. Parents were selected for reproduction using the same form of tournament selection as in CEA. The chromosome representation and genetic operators were also the same as in CEA, and the genetic operators were applied with the same frequencies.

As can be seen in Fig. 3, both host-parasite algorithms perform more poorly than CEA in this experiment. However, it is important to realise that the main point of these experiments is to compare the two host-parasite algorithms with each other. The choice of test problem is primarily based on finding a way of systematically varying the degree of asymmetry, whereas comparison of CEA with host-parasite algorithms is mainly of interest in real applications. The CEA results will however be included in some result graphs to provide a reference.

The next set of results, in Fig. 4, show the effect of letting the populations evolve on landscapes of different ruggedness, in this case using $K_H = 1$ and $K_P = 3, 7, 11$. AHPA and SHPA perform very similarly on the less rugged landscapes, but AHPA significantly outperforms SHPA on the more rugged landscapes. This is an important result, given our hypothesis that asymmetry is an important factor in host-parasite algorithm performance. Additional runs made for $K_H = 3, K_P = 7, 11$ and $K_H = 7, K_P = 11$ gave very similar results (not shown).

In the final experiments we vary the size of the landscapes. Fig. 5 shows results on landscapes with $K = 3$ and asymmetric sizes. Hosts evolve on landscapes with $N = 100$, while parasites evolve on landscapes with $N = 200$ and $N = 400$. Fig. 6 shows results when the landscapes are asymmetric both in their levels of epistasis and in their sizes. Hosts evolve on a $K = 1, N = 100$ landscape, while parasites evolve on more rugged landscapes with $K = 3$ and three different

**Fig. 3.** Results on $NK$ landscapes with $N = 100$ and $K = 1, 3, 7, 11$. SHPA and AHPA used symmetric landscapes, i.e. two landscapes with the same level of epistasis. The x-axis shows the number of evaluated individuals.
Fig. 4. SHPA and AHPA results for three pairs of NK landscapes with one population using a $K = 1$ landscape, and the other landscapes with $K = 3, 7, 11$. Graphs on the left: Results from $K = 1$ population. Graphs on the right: Results from landscapes with $K = 3, 7, 11$. CEA results from landscapes with $K = 1, 3, 7, 11$ included for reference.

Fig. 5. SHPA and AHPA results on two asymmetric pairs of NK landscapes with one population evolving on a $K = 3, N = 100$ landscape, and the other on landscapes with $K = 3$ and $N = 200, 400$. Graphs on the left: Results for the $K = 3, N = 100$ population. Graphs on the right: Results from the landscapes with $N = 200, 400$. 
sizes $N = 100$, $N = 200$, and $N = 400$. The results of this final experiment show the same trend of AHPA performing well under high levels of asymmetry, as in all of the earlier experiments.

Since AHPA has a mechanism for allocating a different number of generations to the populations, it is of interest to monitor how AHPA uses this mechanism. In Table 1 we show statistics of how AHPA allocates generations to the two populations. These statistics indicate that $N$ is the variable which has the largest effect on the degree of asymmetry. All of the landscape pairs with symmetric landscape sizes, i.e. $N_H = N_P$, give very similar AHPA behaviour, regardless of whether $K_H = K_P$ or not. In all such cases AHPA allocates approximately 50% of generations to the $K_H$ population. Observing the statistics from landscape pairs with $N_H \neq N_P$ indicate that asymmetric landscape sizes have a much greater impact on AHPA's behaviour than asymmetric levels of ruggedness. In the two most asymmetric cases, with $N_H = 100$ and $N_P = 400$, AHPA only allocates 12-14% of the generations to the $N_H$ population.

5 Conclusions

We have shown how the $NK$ model can be adapted to host-parasite algorithms, and how it can be used for a systematic evaluation of the effects of asymmetry. Our simulations were limited to a relatively narrow range of asymmetries, but within this range they show a consistent advantage of AHPA over SHPA. By
Table 1. Statistics of AHPA for NK landscapes. $G_H$ is the average proportion of generations allocated to the host population. Switches is the average number of times that AHPA switched focus from one population to the other during the run.

<table>
<thead>
<tr>
<th>$K_H$</th>
<th>$K_P$</th>
<th>$N_H$</th>
<th>$N_P$</th>
<th>$G_H$</th>
<th>Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>0.48</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>100</td>
<td>100</td>
<td>0.49</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>100</td>
<td>100</td>
<td>0.50</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>100</td>
<td>100</td>
<td>0.48</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>100</td>
<td>100</td>
<td>0.53</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>100</td>
<td>100</td>
<td>0.54</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>100</td>
<td>100</td>
<td>0.49</td>
<td>52</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>100</td>
<td>200</td>
<td>0.25</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>100</td>
<td>200</td>
<td>0.28</td>
<td>67</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>100</td>
<td>400</td>
<td>0.12</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>100</td>
<td>400</td>
<td>0.14</td>
<td>66</td>
</tr>
</tbody>
</table>

studying how AHPA allocates generations to the two populations, we have also seen that it seems to make efficient use of its capacity for self-adaptation.

Further simulations should investigate the boundaries for AHPA's capacity to counteract asymmetry. We would also like to study the dynamics of its behaviour in more detail, for example by recording more data of when switches from one population to the other occur. On difficult applications, AHPA can show a tendency to get stuck in one of the populations for large instances of the problem, and it would be desirable to find a method for avoiding this problem.

Together with our previous research, these simulations indicate that AHPA is a promising step towards creating a more robust host-parasite algorithm. In the long run, we would like to achieve a "standard" host-parasite algorithm, similar to the "standard" genetic algorithm, which can be widely used by researchers and applied to a wide range of practical problems.

Acknowledgments: This research was partly funded by grant 1507/97 from the Foundation for Knowledge and Competence Development, Sweden.

References


Towards Balanced Coevolution

Jan Paredis

MATRIKS, Universiteit Maastricht, Postbus 616,
NL-6200 MD Maastricht, The Netherlands
jan@riks.nl
http://saturn.matriks.unimaas.nl/CE/paredis/

Abstract. This paper shows that the performance of coevolutionary genetic algorithms can be improved considerably by introducing a balancing mechanism. This is to prevent one population from "out-evolving" the other one. As a result, fitness variance is maintained and can be used to guide coevolution.

Two different balancing mechanisms are introduced here. Their performance is compared to an unbalanced coevolutionary genetic algorithm. Finally, causal links are suggested between: a lack of balance, the loss of important niches and coevolutionary cycles.

1 Introduction

Inspired by the work of Hillis [1], Paredis [2] introduced a Coevolutionary Genetic Algorithm (CGA) which uses a partial but continuous fitness evaluation resulting from encounters between individuals belonging to different species. The first CGA was used to train a Neural Network (NN) on a classification task. Next, an abstract class of problems which can be solved by CGAs was defined: so-called test-solution problems [3]. This large class of problems in Artificial Intelligence and Computer Science in general, involves the search for a solution which satisfies a given set of tests. Inductive learning is an example of such a test-solution problem. It involves searching for an abstract concept description which generalizes given examples and excludes the given counter examples. Constraint satisfaction is another example of the same type: one searches for a solution which satisfies the given constraints.

A CGA uses two populations to solve test-solution problems. One contains the tests which a solution should satisfy. The other consists of potential solutions to the problem. CGAs have been applied to various other test-solution problems, such as: constraint satisfaction [4], process control [5], path planning [6], and the evolution of cellular automata (CA) for density classification [7]. The main purpose of the applications described above is to help us understand, explore, and illustrate the operation of CGAs. Recently, researchers have been using CGAs in real-world applications such as object motion estimation from video images [8] and time tabling for an emergency service [9].

Some CGA applications used a test population whose contents remained fixed over time [2, 4, 9]. In others, both populations fully evolved [5, 6, 7, 8]. In the latter case, there is pressure on both populations to improve, i.e. the solutions "try" to correctly satisfy as many individuals as possible in the test population. At the same time the tests "try" to make life hard for the solutions. It is not in their "inter-
that the problem is solved. Especially if there is no solution satisfying all possible tests then the tests might keep the solutions under constant attack. Even if there exists a solution satisfying all tests, it might be virtually unreachable given the co-evolutionary dynamics. This might be caused by the high degree of epistasis in the linkages between both populations. Due to such linkages a small change to the individuals in one population might require extensive changes in the other population. This was exactly what happened in the CA application [6]: the tests were able to keep ahead of the solutions by "jumping" back and forth between two regions - in close proximity of each other - in the space of all tests. After each such jump of the test population, extensive changes to the "genes" of the solutions, i.e. CAs, were required in order to satisfy the tests. Paredis [6] provides a simple diversity preserving scheme which prevents the tests from keeping ahead of the solutions. In fact, it achieves this by randomly inserting and deleting individuals from the test population. It also suggests possibly better mechanisms, such as: fitness sharing or the use of different reproduction rates in both populations. The latter approach is investigated here. The goal is to obtain a good balance such that there is a steady progress in both populations. This is to avoid the tests from "out-evolving" the solutions and to ensure that there is sufficient variation in the fitness of the members of a population. The latter is necessary to guide (co)evolution.

The use of fitness sharing to preserve diversity in a competitive coevolutionary context has been investigated by a number of researchers. The Othello application of Smith and Gray [10] is probably one of the earliest. Rosin and Belew [11] combine competitive fitness sharing and shared sampling. The latter ensures that representative opponents are selected for the "competition duels". Jullié and Pollack [12] proposed another way to ensure population diversity: in a classification application the fitness is calculated by counting the number of cases a solution classifies correctly but which the solution it encounters wrongly classifies. More recently, Rosin [13] has identified the causes why coevolutionary progress can come to a halt: the loss of important niches, coevolutionary cycles, or a lack of balance. He observed that, even with fitness sharing, solutions might "forget" how to compete against old extinct types of individuals which might be difficult to rediscover. For this reason he introduces a "Hall of Fame", containing the best individuals of previous generations which are used for testing new individuals. This introduces some kind of memory in the system. In order to keep the progress in both population balanced, Rosin [13] introduces a "phantom parasite" which provides a niche for "interesting" individuals.

The final goal of a CGA solving a test-solution problem is to find a solution which satisfies all possible tests. Any other "victory" of the solution population is only a Pyrrhic victory. A solution population whose members satisfy all members in the test-population, but who do not satisfy tests which are unlikely to make their way in the test-population, is useless. This is another reason why it is important to preserve the genetic diversity of the test population for long enough a period.

In this paper literature on coevolutionary algorithms is given only when it is directly relevant to the topic treated here. An extensive overview of research on coevolutionary algorithms can be found in [14]. The structure of this paper is as follows. The next section describes the problems which will be used in the experiments. Sections 3 and 4 introduce an unbalanced CGA and two balanced CGAs, respectively. Next, the empirical results are analyzed. Finally, conclusions are given.
2 The Problem Setup

The class of test problems used here originates from [15]. The goal is to solve the matrix equation \( \mathbf{A} \mathbf{x} = \mathbf{b} \). The matrix \( \mathbf{A} \) (dimension \( n \) by \( n \)) and the \( n \)-dimensional vector \( \mathbf{b} \) are given. The \( n \)-dimensional vector \( \mathbf{x} \) has to be found. Or, equivalently, the following system of linear equations has to be solved:

\[
b_i = \sum_{j=1}^{n} a_{ij} x_j \quad i = 1, \ldots, n
\]

The function below has to be maximized. In order to avoid confusion with the fitness function used in the CGA, this value will be called the strength of an individual.

\[
-\sum_{i=1}^{n} \left( \sum_{j=1}^{n} a_{ij} x_j \right) - b_i
\]

The diagonal elements of \( \mathbf{A} \) are randomly chosen from the set \{1, 2, 3, \ldots, 9\}. When all off-diagonal elements of \( \mathbf{A} \) are zero then the problem is not epistatic, i.e. each component \( x_i \) only contributes to its corresponding \( b_i \). By setting off-diagonal entries to non-zero, the degree of epistasis is increased: the more non-zero off-diagonal elements in \( \mathbf{A} \), the higher the degree of epistasis.

When only the elements on the diagonal (\( a_{ii} \)) and immediately above it (\( a_{i-1,i} \)) are non-zero then the \( i \)-th equation couples \( x_i \) and \( x_{i+1} \). Due to transitivity, this means that all \( x \)-components are coupled. Hence, given an array with only non-zero diagonal elements one can gradually increase the degree of epistasis by setting elements immediately above the diagonal to non-zero. This procedure is used in the experiments reported here in order to control the degree of epistasis. In order to allow for easy experimentation, the \( b_i \) are chosen such that all components \( x_i \) of the solution to the system of equations are equal to 1. In all experiments reported here, \( n \) is equal to 10, i.e. \( \mathbf{A} \) is a 10 by 10 matrix, and \( \mathbf{x} \) and \( \mathbf{b} \) both contain 10 components.

In ascending order of epistasis the three problems used here are named: TRIVIAL, SIMPLE, and MEDIUM. All use a matrix \( \mathbf{A} \) with non-zero diagonal elements. They differ in the number of non-zero elements immediately above the diagonal. All other elements of \( \mathbf{A} \) are zero in all problems.

TRIVIAL is a problem without any epistasis: all off-diagonal elements are zero. SIMPLE is slightly more epistatic, its variables are linked pairwise. These pairs are non-overlapping. Or more formally, the set of linkages is: \{(x_1,x_2), (x_3,x_4), (x_5,x_6), (x_7,x_8), (x_9,x_{10})\}. These linkages are realised by setting the elements \( a_{i-1;i} \) to non-zero and this for all even \( i \), i.e. \( a_{12}, a_{34}, a_{56}, a_{78}, a_{910} \) are assigned values randomly drawn from the set \{1, 2, 3, 4, 5, 6, 7, 8, 9\}. MEDIUM has still more linkages, namely: \{(x_1,x_2), (x_2,x_3), (x_3,x_4), (x_4,x_5), (x_5,x_6), (x_7,x_8), (x_9,x_{10})\}. Hence, in MEDIUM the variables \( x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \) are completely interlinked. In addition to this, \( x_7 \) is linked with \( x_8 \) and \( x_9 \) with \( x_{10} \).

For most instances belonging to the class of problems described above, the performance of a GA critically depends on the representation used. Only in two extreme
cases is the choice of representation irrelevant. The first one is when there is no epistasis, i.e. all elements of $A$ above the diagonal are zero. In this case, different orderings of the $x_j$ on the string give - on average - the same result. At the other extreme - with all elements above the diagonal different from zero - any representation will perform equally poorly.

3 An Unbalanced CGA

In the experiments described here, two populations compete. Each one climbs its own strength landscape. Both landscapes can have different degrees of epistasis. As a first step, both initial populations are filled with randomly generated individuals, each one representing one potential solution - $x$ - to the system of linear equations to be solved. This initializes each of the 10 components of each individual with a real number uniformly drawn from the interval $[-30,30]$. Next, their fitness is calculated by pairing them up with 20 randomly selected members of the other population. In order to calculate the fitness of an initial individual, one counts the number of times it is paired up with an individual whose strength is lower than its own. As a matter of fact, each individual has a history which stores the fitness feedback - or pay-off - resulting from such an encounter with an individual from the opposite population. As was the case for test-solution problems, the pay-offs are restricted to two values, zero and one, instead of just being equal to the respective strengths. This is to facilitate the generalization of the conclusions drawn from the experiments described here to test-solution problems in general. We will also stick to the same terminology. Hence, one population is called the solution population, the other the test population. A solution receives a pay-off of one if its strength is larger or equal than that of the test. Otherwise it receives a zero. The opposite is true for tests: they get a pay-off of one if the strength of the solution encountered is smaller than its own strength.

The fitness of an individual is equal to the sum of the pay-offs in its history. Note that when the individuals involved in an encounter have the same strength, the solution gets the pay-off of one and the test gets zero pay-off. This to ensure that there exists a solution which beats all tests. Ideally, the CGA should find this individual. It is also good to note that the populations are sorted on fitness: the fitter the individual the higher its rank in the population.

The pseudo-code below describes the basic cycle the CGA executes after the initial populations are generated. First, 20 encounters are executed in which a test and solution are paired up. The SELECTion of these individuals is biased towards highly ranked individuals, i.e. the fitter individuals are more likely to be SELECTed. In a CGA, the most fit individual is 1.5 times more likely to be selected than the individual with median fitness. Next, the actual ENCOUNTER happens during which the strengths of the solution and test are compared as described above. As above, the result of the encounter is one if the strength of the solution is larger or equal than that of the test. Otherwise it is zero. This result is also the pay-off received by the solution. The test receives the "inverse" pay-off. The function TOGGLE in the basic cycle implements this inverse fitness interaction between tests and solutions. It changes a one into a zero and vice versa. Again, each individual stores the pay-off it receives in its history. At the same time, the pay-off that the individual received least recently is removed from the history. This guarantees that the
history will always contain the 20 most recently received pay-offs. This UPDATE of the HISTORY of an individual, is followed by an UPDATE of the FITNESS of the individual: it is set equal to the sum of the pay-offs in its history. Because both populations are kept sorted on fitness, an individual might move up and down in its population as a result of the update of its fitness.

After the execution of the encounters, one offspring is generated in each population. This happens as follows. First, two parents are SELECTED from the population, again with the same bias towards fitter individuals as explained above. Next, (2-point reduced surrogate) CROSSOVER and (adaptive) MUTATION is used in order to generate the child from the bit-string of each of its parents. Then the FITNESS of the CHILD is calculated as the sum of the pay-off received from encounters with 20 SELECTed individuals from the other population. If this fitness is higher than the minimum fitness in the population to which its parents belong then the child is INSERTED at the appropriate rank location in this population. At the same time, the individual with the minimum fitness is removed. In this way, both populations remain sorted on fitness.

DO 20 TIMES
    sol:= SELECT(sol-pop)
    test:= SELECT(test-pop)
    res:= ENCOUNTER(sol,test)
    UPDATE-HISTORY-AND-FITNESS(sol,res)
    UPDATE-HISTORY-AND-FITNESS(test,TOGGLE(res))
ENDDO

sol-parent1:= SELECT(sol-pop)
sol-parent2:= SELECT(sol-pop)
sol-child:= MUTATE-CROSSOVER(sol-parent1,sol-parent2)
f:= FITNESS(sol-child)
INSERT(sol-child,f,sol-pop)
test-parent1:= SELECT(test-pop)
test-parent2:= SELECT(test-pop)
test-child:= MUTATE-CROSSOVER(test-parent1,test-parent2)
f:= FITNESS(test-child)
INSERT(test-child,f,test-pop)

Some additional remarks are in place here. In order to avoid unreliable fitness feedback resulting from possibly mediocre offspring, the fitness calculation of new individuals do not result in fitness feedback for the individuals on which they are tested.

Some additional bias towards beating strong, possibly not yet beaten, opponents is introduced by testing new individuals on 19 SELECTed individuals and on the opponent ranking at the top of its population.

Finally, the size of the histories, as well as the number of encounters per cycle, both set to 20. This is quite arbitrary but the same values have been used in all iter CGA applications, as well.

1 Balancing Mechanisms

In the unbalanced CGA described above, each population generates one offspring per cycle. This may result in unbalanced coevolution when the difficulty (e.g. the degree
of epistasis) of both strength landscapes and/or the amount of progress in both populations is substantially different.

The author has experimented with different balancing mechanisms. Each such balancing mechanism generates only one offspring per cycle. It is the balancing mechanism which determines which population reproduces. A first mechanism uses a fixed ratio between the probability of generating offspring from a particular population, e.g. the solution population has a probability of 0.1 to generate offspring. In this case, the test population, reproduces with a probability of 0.9. This approach has a couple of disadvantages. First of all, it is too static. Due to the stochastic nature of a CGA, the degree to which the populations are out of balance may differ during a run or from run to run. Secondly, just as is the case with GAs, CGAs are typically used for problems about which little knowledge is available. Hence, it might be difficult to find an optimal ratio, even if it would exist.

For the reasons given above one would like the ratio to adapt in line with the progress in both populations. If the test population gets ahead, more search effort - i.e. reproduction, should be invested in the solution population, and vice versa. There are a couple of alternatives how this could be done.

Paredis [16] introduced the so-called X-method which allocates reproductions stochastically depending on the number of successes during the twenty encounters executed per cycle. If in all twenty encounters the solutions are successful then the test population should reproduce. The probability that the test population reproduces decreases linearly towards zero as the solutions are less successful during the encounters. Figure 1 shows why this method is called the X-method. This graph has a clear X shape. It represents the probability that the test or solution population reproduces as a function of the number of successful encounters of solutions. In practice, this graph is used as follows. First a random real number is drawn from the closed interval [0, 1]. If this number is smaller than the probability that the solution population
reproduces, given the number of successful encounters, then the solution population reproduces else offspring is generated for (and from) the test population.

Obviously, the number of successes during the encounters is not the only possibility to measure the degree of balance between two populations. One can also compare the fitness of the individuals of both populations. The AVG-method chooses the population with the lowest average fitness for reproduction.

The following section provides experiments which allow us to understand the merit of the balancing mechanisms introduced above.

5 Empirical Results

The first set of experiments compares the unbalanced CGA - UNBAL - with X and AVG. The nine different combinations with each population adapting on one of the three problems - TRIVIAL, SIMPLE, MEDIUM - are investigated. Each run was allowed to create a total of 75000 offspring. Hence, UNBAL which creates two offspring per cycle - executed 37500 basic cycles (cf. the code given in section three). The balancing mechanisms - X and AVG - on the other hand, execute 75000 cycles to generate the same amount of offspring. In order to ensure that all variants use about the same amount of computing time and fitness feedback, the balanced algorithms perform 10 encounters per cycle instead of 20. Furthermore, both populations have a size of 100 and each algorithm was run on 50 different problem instances.

Table 1 gives for each combination how often a particular algorithm found the best individual - i.e. with the highest strength - of all three algorithms. For example, the lower leftmost entry of table 1, 2 / 21 / 27, indicates that out of 50 runs on a MEDIUM solution landscape and a TRIVIAL test landscape, UNBAL encountered a better solution than the balanced algorithms twice. For X this was the case in 21 runs. AVG outperformed the other two in 27 runs. Note that for each run different landscapes were created.

<table>
<thead>
<tr>
<th>sol \ test</th>
<th>TRIVIAL</th>
<th>SIMPLE</th>
<th>MEDIUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIVIAL</td>
<td>0/24/26</td>
<td>0/28/22</td>
<td>4/24/22</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>0/25/25</td>
<td>1/24/25</td>
<td>3/21/26</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>2/21/27</td>
<td>4/22/24</td>
<td>9/17/24</td>
</tr>
</tbody>
</table>

Table 1: Number of times one algorithm outperforms the others. The following format is used: UNBAL/X/AVG

In each of the nine possible combinations of problems given in table 1, the unbalanced CGA is clearly inferior to the other two. It came to a surprise that this is also the case when both problems - test and solution - have the same degree of epistasis. This indicates that the balancing mechanisms also play an important role in situations in which one population gets ahead by chance without there being a significant difference in the degree of epistasis in both problems. These results support our earlier rejection of an a priori fixed difference in reproduction rates: one cannot know beforehand which population will progress the fastest, how much difference in progress there will be and when this will happen.
It should also be noted that as the degree of epistasis increases, the performance of a (C)GA will decrease. Eventually, when this degree of epistasis is maximal - i.e. all $x_i$ are coupled - a (C)GA will become a random search algorithm. This is the reason why UNBAL outperforms the other two algorithms somewhat more as the degree of epistasis increases. In fact, with a MEDIUM solution landscape and a MEDIUM test landscape, UNBAL outperforms the other 2 algorithms 9 times (out of 50). On maximally epistatic problems, all three algorithms, on average, have the same performance.

The difference in performance between the balanced mechanisms is far less outspoken. In fact, their average final best-sofar strengths are always close to each other, within one standard deviation. At a second glimpse, both balancing methods are quite similar. They are both based on a discrepancy of the success of both populations. X does base this on the 20 most recent encounters and operates stochastically. AVG, on the other hand, takes into account all the fitness feedbacks received by the entire population to deterministically choose which population is allowed to reproduce. It is not that strange that over the entire run their average behavior will be similar.

6 Conclusions and Discussion

It is good to look whether nature also "uses" a balancing mechanism. When a predator species becomes too successful then more prey is killed. In the short term, the population size of the prey decreases while that of the predators increases. This puts a natural brake on the progress of the predator species because food (i.e. prey) becomes more scarce. Similarly, too effective prey will, on one hand, lead to a rapid increase of the prey population size, which, in its turn, provides a larger food resource for the predators. On the other hand, the predator population size decreases, i.e. the selective pressure to improve increases. Hence, in nature, fluctuations in population size help to keep (co)evolution in balance. This in contrast with the fixed population sizes in CGAs. The balancing mechanisms introduced here adapt the selective pressures through a change of reproduction rates.

As described in the introduction, other researchers have used fitness sharing to preserve genetic diversity in a coevolutionary environment. The balanced CGA deals with the direct cause of the loss of diversity: selection. When selection is too severe, the population rapidly converges to a (sub)optimum. In a CGA, selection occurs through competition between new offspring and individuals already in the population. The balancing mechanisms proposed here make the reproduction rate (and hence the amount of selection) dependent on the success of the population. In this way, the algorithm avoids that the better population "out-evolves" the weaker one. In fact, once an imbalance occurs, the weaker population is allowed to catch up: more search effort is spent on it. This at the expense of the stronger population whose progress - and convergence - is slowed down.

---

1 Analogous to the CGA, a two-population model with one prey population and one predator population is assumed. When a population predates on multiple species then a decline of one of its prey populations might result in more intensive predation on the other population(s).
Two balancing variants were compared here. On the problems described in this paper, performance of both variants is rather comparable. The experiments, however, clearly show a considerably better performance of the balanced algorithms in comparison with the unbalanced CGA. Two reasons can be identified for this success. A first one was already given in the previous paragraph: balanced coevolution maintains genetic diversity. In this way, individuals are confronted with a diverse opposition. Secondly, the balancing mechanism, together with the inverse fitness interaction between both populations, ensures that the fitness variance between members of a population is maximal. This because both balancing mechanisms restore an imbalance in pay-off, and hence fitness. Due to the inverse fitness interaction, the rise of fitness in one population will lead to a decrease of the fitness of the other population. Hence, a situation in which (almost) all individuals in a population have a fitness of one (or zero) is avoided as long as possible. The conservation of fitness variance, in its turn, guides (co)evolution.

This also suggests strong causal relationships between the three causes which can bring coevolutionary progress to a halt mentioned by Rosin [13]. These causes were already enumerated in the introduction. When there is a lack of balance, one population "out-evolves" the other. This results in too much exploitation which, in its turn, results in a loss of diversity. In this way, important niches might disappear. In the case solution niches disappear then tests can exploit these "empty niches". Which might "force" the solution population to re-populate these niches at the cost of leaving other niches. If this process iterates coevolutionary cycles may occur. If it are test niches which disappear then the solutions are no longer tested on a representative set of tests. Hence, a good balancing mechanism might decrease the need for memory and/or fitness sharing.

Recently, Olsson [17] introduced a balancing mechanism which lets a population evolve until it contains an individual (a "champion") beating all the individuals in the other population. Next, the other population evolves until it finds an individual beating all members of the first population. This process is then repeated but at each stage all the previous champions as well as the current members of the opposite population should be beaten. Such a mechanism could be introduced in a balanced CGA by marking champions. These marked individuals should always remain in the population.

In conclusion, the current paper shows the worth of balancing mechanisms. It does, however, not resolve which balancing mechanism is to be used for a given problem. Possibly other mechanisms then the ones mentioned here might result in further improvement. Different domains might even require different balancing mechanisms. In fact, initial experiments in the cellular automata domain seem to confirm this.

**Acknowledgements**

The author is indebted to Roos-Marie Bal for proofreading an earlier version of this paper.

---

Clearly, this is analogous to premature convergence in traditional single population genetic algorithms.
References


Spatial Games with Adaptive Tit-for-Tats

Elpida S. Tzafestas
Institute of Communication and Computer Systems, Electrical and Computer Engineering Department, National Technical University of Athens, Zographou Campus, Athens 15773, GREECE
brensham@softlab.ece.ntua.gr
http://www.softlab.ece.ntua.gr/~brensham

Abstract. This paper presents an adaptive tit-for-tat strategy and a study of its behavior in spatial IPD games. The adaptive tit-for-tat strategy is shown elsewhere to demonstrate high performance in IPD tournaments or individual IPD games with other types of strategies, and obtains higher scores than the pure tit-for-tat strategy. In spatial IPD games, the strategy exhibits stability and resistance to perturbations, and those properties are more pronounced in variations of the spatial game model that induce some degree of “noise”: probabilistic winning, spatial irregularity and continuous time. The adaptive tit-for-tat strategy is also compared to pure tit-for-tat and found to be more stable and predominant in perturbed environments.

1 Introduction

A major research theme in theoretical biology and evolutionary computation is the emergence and evolution of cooperative behavior between selfish agents. The cooperation problem states that each agent has a strong personal incentive to defect, while the joint best behavior would be to cooperate. This problem is traditionally modeled as a special two-party game, the Iterated Prisoner’s Dilemma (IPD). At each cycle of a long interaction process, the agents play the Prisoner’s Dilemma. Each of the two may either cooperate (C) or defect (D) and is assigned a payoff defined by the following table.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Opponent</th>
<th>Payoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>3 (= Reward)</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>0 (= Sucker)</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>5 (= Temptation)</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>1 (= Punishment)</td>
</tr>
</tbody>
</table>

Usual experiments with IPD strategies are either tournaments or ecological experiments. In tournaments, each strategy plays against all others and scores are summed in the end. In ecological experiments, populations of IPD strategies play in tournaments and successive generations retain the best strategies in proportions analogous to their score sums.

The first notable behavior for the IPD designed and studied by Axelrod [1][2] is the Tit For Tat behavior (TFT, in short):

Start by cooperating,
From there on return the opponent’s previous move
This behavior has achieved the highest scores in early tournaments and has been found to be fairly stable in ecological settings. TFT demonstrates three important properties, shared by most high scoring behaviors in IPD experiments.

- It is good (it starts by cooperating)
- It is retaliating (it returns the opponent's defection)
- It is generous (it forgets the past if the defecting opponent cooperates again).

Further strategies include stochastic ones ([11]), the Pavlov strategy ([13]) that cooperates when it has played the same move as its opponent etc. In the literature we may also find studies in an evolutionary perspective ([6]), theoretical or applied biological studies ([3][5][10]) and studies of modified IPD versions ([14]).

We have designed an adaptive tit-for-tat strategy that is analyzed in detailed elsewhere ([15]) and is shown to score better than pure tit-for-tat and most of the other known strategies. Our initial motivation for this work was to find a strategy that would be able to cooperate with cooperative behaviors and defect against defective behaviors, while maintaining reversible, not permanent memory. This strategy would demonstrate behavioral gradualness and possess the potential for stability in front of changing worlds. In what follows, we are studying the issue of stability of the adaptive tit-for-tat strategy in spatial games of the sort introduced by Nowak and May ([12]).

2 The Adaptive Tit-for-Tat Strategy

An adaptive tit-for-tat strategy should be essentially tit-for-tat in the sense of being good, retaliating and forgiving. Moreover, it should demonstrate some behavioral gradualness that would show as fewer oscillations between C and D. To this end, it should use an estimate of the opponent's behavior, whether cooperative or defecting, and should react to it in a tit-for-tat manner. The estimate should be continuously updated throughout the interaction with the opponent. The above may be modeled with the aid of a continuous variable, the world's image, ranging from 0 (total defection) to 1 (total cooperation). Intermediate values will represent degrees of cooperation and defection. The adaptive tit-for-tat model can then be formulated as a simple linear model:

\[
\text{Adaptive tit-for-tat basic model}
\]

\[
\begin{align*}
\text{If (opponent played C in the last cycle) then} & \\
& \text{world} = \text{world} + r*(1-\text{world}), \text{r is the adaptation rate} \\
\text{else} & \\
& \text{world} = \text{world} + r*(0-\text{world}) \\
\text{If (world >= 0.5) play C, else play D}
\end{align*}
\]

The usual tit-for-tat model corresponds to the case of \(r=1\) (immediate convergence to the opponent's current move). Clearly, the use of fairly small \(r\)'s will allow more gradual behavior and will tend to be more robust to perturbations. We have observed that the adoption of two different rates, one for cooperation \(r_c\) and one for defection \(r_d\), allows us to assume the right behavior against retaliating or irrational strategies. Retaliating strategies are those that basically seek cooperation, but start by exploring the opponent's reaction to a few initial D moves. For example, the suspicious tit-for-tat (or STFT) strategy starts by defecting, and then plays usual tit-for-tat. On the
contrary, irrational strategies are those that do not employ any feedback from the game and play blindly using some innate law. For example, periodic strategies repeat patterns of C’s and D’s, such as CDD, CCD, CDCD etc. We have shown ([15]) that a high cooperation rate and a low defection rate allow us to converge to cooperation against a retaliating agent, while the opposite setting allows us to converge to total defection against an irrational agent, in both cases achieving maximum score. Thus, we need a method for the adaptive tit-for-tat agent to discover whether the opponent uses a retaliating behavior or is just irrational, so as to adopt the proper rate setting. We have designed and examined several such variants for estimating the opponent’s irrationality and we have finally found the following rule:

**Adaptive tit-for-tat model**

If (opponent played C in the last cycle) then

world = world + \( r_c \cdot (1 - \text{world}) \), \( r_c \) is the adaptation rate for cooperation

else

world = world + \( r_d \cdot (0 - \text{world}) \), \( r_d \) is the adaptation rate for defection

If (world \( \geq 0.5 \)) play C, else play D

Throughout an observation window, record how many times (\( n \)) the agent’s move has coincided with the opponent’s move. At regular intervals (every “window” steps) adapt the rates as follows:

If (\( n \geq \text{threshold} \)) then \( r_c = r_{\text{min}}, \quad r_d = r_{\text{max}} \)

else \( r_c = r_{\text{max}}, \quad r_d = r_{\text{min}} \)

The rule may be translated as:

If (the world is cooperative)* then \( r_c = r_{\text{min}}, \quad r_d = r_{\text{max}} \)

else \( r_c = r_{\text{max}}, \quad r_d = r_{\text{min}} \)

(*) recall that “my move = opponent’s move” is the so-called pavlovian criterion of cooperation (Nowak and Sigmund [13])

Note that the agent drops its cooperation rate when the world is assumed cooperative, and increases it otherwise, that is, it uses negative feedback at the rate regulation level. Another alternative interpretation for a cooperative world is a world that tries to manipulate the agent (so as to get it to respond with the same value found in the world). In this case it makes sense to drop the cooperation (potential manipulation) rate and become less adaptive to the world.

We have shown in simulations ([15]) that the adaptive tit-for-tat strategy with this meta-regulation mechanism converges to the proper behavior and achieves the highest score against both retaliating and irrational strategies, such as STFT and periodic strategies, respectively.

The adaptive tit-for-tat strategy manages to differentiate between a retaliating strategy and an irrational one that has initially the same behavior. For example it manages to converge to total defection against CDCD, that resembles STFT in the beginning.

The meta-regulated model is insensitive to the initial value of its world variable, provided that it is at least equal to 0.5 (remember that a tit-for-tat like behavior should start by cooperating). However, even for a defective initial value of the world variable, the adaptive agent may converge to cooperative behavior against the tit-for-tat agent.

The model is also insensitive to the exact values of \( r_{\text{max}} \) and \( r_{\text{min}} \). Different values for the two rates will only result in scaling or stretching of the resulting curve for the world variable, the qualitative performance remaining intact. The same thing applies
to the values of the observation window and the threshold, although they must be constrained so that the window will be sufficiently large and the threshold sufficiently small compared to the window.

3 Spatial Games I: Basic Model

We have run spatial games ([12][9]) in two-dimensional grids consisting of diverse agents, i.e., agents having different IPD strategies. In these games, all agents run a finite IPD game against their neighbors at each cycle. After all games are finished, the agent at each place assumes the winner strategy among itself and its neighbors, i.e., the strategy whose score sum is greatest. The spatial game continues in the next cycle with the new generation of strategies and so on. Such experiments show the evolution of different types of strategies in spatial grids. In our work, we assume the classical IPD game with the typical payoff matrix already presented and we study the behavior of our adaptive tit-for-tat strategy and its potential for stability.

Experiments starting with randomly placed agents of three kinds (ALLC, ALLD and adaptive TFT) have been found to quickly settle on a stable configuration consisting of high percentages of adaptive TFT and very low percentages of ALLC with occasional small stable territories occupied by ALLD. Figure 1 shows such a stable configuration with a stable ALLD territory and the corresponding experiment’s evolution of strategies proportions in the population. As is obvious from figure 1, ALLD’s safe territories lie always between ALLCs (that ALLD can exploit) and adaptive TFTs (that ALLD cannot cheat).

![Figure 1](image)

**Fig. 1.** (a) A stable configuration of adaptive TFT, ALLC and ALLD, that has occurred after a few cycles in a spatial IPD game (20x20 grid). Blue (Gray) = Adaptive TFT, White = ALLC, Black = ALLD. ALLD has built a safe territory. Adaptive TFT has dominated to the detriment of ALLD and ALLC has been invaded by ALLD. (b) Corresponding evolution of strategies.

The experiment has been repeated with populations of various strategies and it has been found that most strategies die out quickly, i.e., in a few generations, while ALLC, ALLD, pure TFT and adaptive TFT persist.

We have also repeated the experiments for initially regular configurations of strategies and we have obtained similar results. Figures 2 to 5 show characteristic results for some regular initial configurations. In figure 2, ALLD builds a stable territory, while in figure 3 (with similar initial population composition) ALLD dies out. Figure 4 demonstrates the process of safe ALLD territory emergence through “chasing” by adaptive TFT agents. Finally, figure 5 gives a type of almost regular configuration that
may lead to two completely different final stable configurations: either to a 100% ALLD population, or to a 100% adaptive TFT population. In all cases, we have observed the emergence of stable configurations of varying composition and form, which may include ALLD blocks next to cooperative strategies.

In order to examine the potential of adaptive TFT as far as stability and resistance to perturbations is concerned, we have performed two kinds of perturbation studies. First, we have performed the “injection test”, where, after the population has stabilized, we inject a number of ALLD agents in random places on the grid. In this case, we have found that the system restabilizes to a new configuration, where the adaptive TFT strategy has risen to higher values than its former stable value, and this to the detriment of ALLC. Figure 6a gives the results of a typical run of this experiment.

The second kind of test that we have performed is the “shuffle test”, where, after the population has stabilized, we shuffle the agents of the population, i.e. we move each one of them in a random place on the grid. In this case as above, we have found that the system restabilizes to a new configuration, where the adaptive TFT strategy has risen to higher values than its former stable value, and this to the detriment of ALLC. Figure 6b gives the results of a typical run of this experiment. As is obvious from the figure, the shuffling of the agents has the potential of leading ALLD agents to total extinction. The conclusion from both these tests is that a perturbation of a stable population—either in the form of injection of irrational agents or in the form of shuffling of the agents—results in a rise of the adaptive tit-for-tat population. This happens to the detriment of other strategies and most notably to the detriment of the ALLC strategy.

---

Fig. 2. (a) Initial configuration of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid). (b) Final configuration. (c) Evolution of strategies in the population.

---

Fig. 3. (a) Initial configuration of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid). Note that the relative positions of ALLD and adaptive TFT blocks are inversed, in comparison to the situation of figure 2. (b) Final configuration. (c) Evolution of strategies in the population.
Fig. 4. Configuration of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid). (a-d) Configuration at t=0, t=6, t=12, t=18. (e) Evolution of strategies in the population.

Fig. 5. A regular configuration of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid). (a) Configuration leading to a 100% ALLD population. (b) Configuration leading to a 100% adaptive TFT population. (c) Evolution of strategies in the population of the configuration (b).

Fig. 6. Perturbation tests in a population of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid). (a) Injection of ALLD agents at t=10, after the system has stabilized (t=7). (b) Agent shuffling at t=35, after the system has stabilized (t=28).

4 Spatial Games II: Model Extensions

We have also performed experiments with the modified game models of May et al. [9] because they represent some kind of "perturbed game models". First, we have experimented with games of probabilistic winning, where the winner strategy in an agent's neighborhood is not determined in an absolute way, as the one achieving the highest score. Instead, it is determined probabilistically according to its score (the
higher the score, the higher the probability of determining the strategy as winner). In this case as before, the system settles to a configuration, where the adaptive TFT strategy rises to high values to the detriment of ALLC. There are, however, three major differences.

- The system does not settle definitely to a configuration, but keeps changing all the time, while the relative proportions of the various strategies are stable around some value.
- The stabilization of the system is much slower than that of absolute winning and a little “noisy”, which shows in irregularity of the resulting curves.
- The ALLD strategy gets always extinct, because of symmetry breaking: All ALLD agents are eventually replaced by suboptimal ALLC or adaptive TFT neighbors.

Figure 7 gives the results of probabilistic winning in a random and a regular initial configuration.

Next, we have experimented with games of spatial irregularity, where the opponents of an agent are not necessarily its immediate neighbors or not all of them. In our implementation, an agent has 50% probability of playing against each one of its immediate neighbors, so that practically at each round it plays against half of them, chosen randomly. In this case the general conclusions of previous game models apply as well. A detailed analysis in the same spirit as that of probabilistic winning reveals the following:

- The stabilization of the system is slower than that of absolute winning but quicker than that of probabilistic winning. The stabilization is steady, as that of absolute winning, and not “noisy”, as that of probabilistic winning.
- The ALLD strategy gets always extinct, because every ALLD agent is eventually replaced by an ALLC or adaptive TFT neighbor that has achieved a higher score in its own different spatial tournament.

Figure 8 gives the results of spatial irregularity in a random and a regular initial configuration.

Finally, we have experimented with continuous-time games ([7][9]), where agents do not play against their opponents simultaneously and are not updated synchronously, but they play and are updated one after the other in random order. In this case the general conclusions of previous game models apply as well. A detailed analysis in the same spirit as that of probabilistic winning reveals the following:
Stabilization is comparable in speed and "noisy", as in probabilistic winning. However, the ALLD strategy is not always extinct, but manages to build safe territories, although a little less often than in the default game model. Figure 9 gives the results of continuous-time games in a random and a regular initial configuration. The conclusion from the above three studies is that a perturbed game model results in higher adaptive tit-for-tat population proportions. This happens to the detriment of other strategies and most notably to the detriment of the ALLC strategy.

Fig. 8. Evolution of strategies in a population of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid) with spatial irregularity. (a) A random initial configuration such as the one of figure 1. (b) A regular initial configuration such as the one of figure 5.

Fig. 9. Evolution of strategies in a population of adaptive TFT, ALLC and ALLD in a spatial IPD game (20x20 grid) with continuous time. (a) A random initial configuration such as the one of figure 1. (b) A regular initial configuration such as the one of figure 5.

5 Spatial Games III: Adaptive TFT versus Pure TFT

Because the adaptive TFT strategy has been shown better than pure TFT in individual games against other strategies as well as in tournaments ([15]), we have examined their relative strength in front of perturbations. There are two notable results in this direction. First, we have performed experiments with arbitrary initial populations including a fair amount of pure and adaptive TFT agents, in the basic game model. As has been already explained, these systems settle eventually to a stable configuration. After such systems stabilize, we inject at regular intervals a number of agents of arbitrary strategy. At each such "injection", the adaptive TFT strategy rises rapidly to the detriment of ALLC and pure TFT. In figure 10a, the initial configuration is a random one that contains 20% STFT, 20% ALLC, 20% CDCD, 20% ALLD, 10% pure TFT and 10% adaptive TFT agents. The system settles at t=5. At t=10, t=15 and t=20, we inject 10% CDCD, 10% lunatic (random) and 10% STFT agents, respectively. As is obvious from the figure, at each perturbation the proportion of the
adaptive TFT rises to the detriment of ALLC and pure TFT. Hence, perturbations strengthen the adaptive TFT and weaken the pure TFT strategy.

A second observation concerns the relative evolution of adaptive TFT and pure TFT in initial configurations consisting of ALLC, ALLD, pure TFT, adaptive TFT and irrational agents (such as CDCD and STFT). In such experiments, it has been observed that the pure and the adaptive TFT strategies stabilize to different values even if starting from the same initial value. More specifically, when the two strategies start from about the same initial value, they settle to a configuration such that the adaptive TFT strategy is of a significantly higher value than the pure TFT strategy, as figure 10b shows. This is an early indication of some sort of Baldwin effect ([4][8]) that deserves further study.

![Fig. 10](image)

Fig. 10. (a) Evolution of strategies in an initial population that contains 20% STFT, 20% ALLC, 20% CDCD, 20% ALLD, 10% pure TFT and 10% adaptive TFT agents in a spatial IPD game (20x20 grid). At t=10, t=15 and t=20, we inject 10% CDCD, 10% lunatic (random) and 10% STFT agents, respectively. (b) Evolution of strategies in an initial population that contains 7% STFT, 38% ALLC, 7% CDCD, 32% ALLD, 8% pure TFT and 8% adaptive TFT agents in a spatial IPD game (20x20 grid). See text for comments on Baldwin effect.

6 Conclusion

We have presented an adaptive tit-for-tat strategy for the IPD game and have studied its stability and resistance to perturbations in spatial IPD games on a 2D grid. The adaptive TFT strategy uses a continuous estimate of the opponent's behavior that it regulates between two bounds. Moreover, at a meta level, it regulates the rates of the previous level regulation, so that it manages to achieve maximum scores in individual games against other strategies and in tournaments. The behavior of the adaptive TFT strategy in spatial IPD games is studied systematically and several perturbation studies are performed, in an attempt to identify the strategy's features and advantages. In typical spatial IPD games between adaptive TFT, ALLC and ALLD on a 2D grid, a population settles to a certain configuration, where adaptive TFT rises to high percentages in the population, while ALLC and ALLD fall to very low values. Surprisingly at first hand, ALLD manages occasionally to survive in such environments and build stable blocks between large ALLC and adaptive TFT areas. As far as the form of resulting final configurations is concerned, even slightly dissimilar initial configurations may lead to completely different final configurations (cf. fig. 5), thus exhibiting some form of chaotic behavior.
Similar results are also obtained in the case of perturbed or noisy game models (probabilistic winning, spatial irregularities and continuous-time games), with differences in convergence speed and steadiness. The relative strength and dominance of adaptive TFT compared to pure TFT is finally demonstrated in a number of experiments that involve subtle perturbation schemes. In all cases, persistent perturbations of various forms strengthen the adaptive TFT and weaken pure TFT and ALLC, while they lead to extinction of irrational strategies in the population. Adaptive TFT demonstrates thus a selective advantage in perturbed environments, because it is fairly stable and resistant to perturbations.

Future work includes a study of parameterized games, in the same spirit as May et al. [9] and an extension of the IPD spatial game model to a two-species game, each one having as a goal to dominate the whole grid.

References

Competitive Segmentation:  
A Struggle for Image Space

C.J. Veenman, M.J.T. Reinders, and E. Backer

Information and Communication Theory Group  
Department of Information Technology and Systems  
Delft University of Technology  
Delft, The Netherlands  
{C.J.Veenman, M.J.T.Reinders, E.Backer} @its.tudelft.nl

Abstract. In this paper, we propose a competitive image segmentation algorithm. It is a dynamic evolving optimization method, which we call the population algorithm. The method is inspired from nature, where the image segments are a population of entities that struggle for the limited image space and settle territory expansion conflicts locally without central authority. Hence, it is a region-based segmentation approach that locally considers region boundary adjustments in a dynamic way. Experiments confirm that this metaphor indeed applies when the image segmentation problem is modeled accordingly.

1 Introduction

Computer vision is the research field that aims at automatic interpretation of the three dimensional world through images with as highest attainable goal a semantic labeling of all regions in the image. Current achievements in this research field are far from this holy grail and successes have been reported only in very restricted domains, e.g. [6], and [14]. The main problem is certainly the lack of appropriate models to describe rich environments.

Computer vision systems can be thought of as the interaction between two modules, an image segmentation module and a high-level interpretation module. The objectives of the segmentation and interpretation modules are quite different. The segmentation module aims at dividing the image up into disjoint homogeneous regions, while the interpretation module groups the regions and assigns semantic labels to them. Consequently, segmentation is merely a dimension reduction process for the complex interpretation task, and hence, a segmentation result should consist of as few regions or segments as possible. There is, however, a constraint imposed on the segmentation result; once regions are merged they can not be separated by the interpreter, i.e. merging too greedily will hamper the interpretation severely (see for overviews on computer vision [5], and [11]).

Segmentation algorithms can be classified as being either region-based or contour based. Region-based algorithms group pixels based on homogeneity in spatial related image features. These algorithms are usually less sensitive to
noise, but quite often depend on some random initialization. Contour-based algorithms impose a parameterized shape on the segments. If the shape model is too flexible it becomes sensitive to noise and otherwise it may restrict the shape of the segments too much (see e.g. [1], [2], [8], [9], and [17]).

In this paper we propose a method for region-based segmentation that is less sensitive to its initialization. We continuously exchange pixels between segments, and we do so only if that results in a higher quality for the two segments involved. As a consequence, the segments compete each other for the same region if it fits both their local criteria. The metaphor from nature on which the proposed method is based, is that of a possibly oversized population that is exploring a limited area, while occupying as much territory as possible. The consequent conflicts are settled between those concerned without the intervention of a central authority. The analogy can for instance be formulated as follows; a segment represents a kingdom and a part of the image space is the territory owned by the kingdom. Then, the kingdom may try to expand at its borders and can merge with another kingdom by marriage if both expect to take benefit of it. Because of its metaphor we have called the proposed optimization scheme the population algorithm.

In this method, we start with a relatively high number of segments. Then, pixels repeatedly migrate to the segment that currently fits best. Consequently, some segments grow and others shrink or even disappear. The main difference with genetic algorithms is that in this case the entities do not try to satisfy the same criterion. The current set of pixels maintained per segment determines the fitness of other pixels.

In the next section we describe the motivation for the chosen approach and the characteristics of the system. Further, we relate it to other reported approaches. Then, we formalize the problem and give the details of the algorithm in the following sections. In the experiments section we illustrate the effectiveness of the algorithm by applying it to synthetic and real images.

2 Distributed Computer Vision

The image segmentation system we are currently building is part of a larger project that aims at the recognition of facial features in images. For a number of reasons we have chosen to adopt the multi-agent paradigm for the design of this facial feature recognition system [16]. The distributed nature of our cooperative image interpretation system inspired us in the design of a distributed algorithm for image segmentation, among others because the best results can be obtained by close cooperation between segmentation and interpretation algorithms. That is, a global segmentation preprocessing step for a distributed interpretation system is less flexible and makes feedback of the interpretation task to the segmentation task very difficult.

Besides design considerations as motivation for our approach, we expect improvements in quality and efficiency. Quality and efficiency aspects are both related to the size of the problem. There is a huge amount of possible image
partitionings, namely over $2^N$, where $N$ is the number of pixels in the image ($10^4 \leq N \leq 10^6$). Clearly, enumeration of the candidate solutions is out of the question. Moreover, the problem is not well structured, so any segmentation algorithm must be an approximation algorithm. However, finding coherent homogeneous regions is partly a local search problem. We claim that repeatedly optimizing this problem locally in a non-greedy way will be more efficient while preventing premature convergence.

In this paper we do not consider parallel implementations of known segmentation algorithms (e.g. [4], [10]), because these approaches usually have the same drawbacks as their sequential counterparts. The algorithms are essentially centralized and need (simulated) shared memory and/or a lot of communication overhead to be able to compute and satisfy global criteria.

To our knowledge, only one distributed image segmentation system as part of an image interpretation system has been reported [12]. This agent-based image understanding system for aerial image interpretation uses a contour-based segmentation scheme, which they call a cooperative distributed region segmentation system. For the algorithm the number of segments must be known in advance. Moreover, the initial segment seeds must be positioned inside the a priori known segments. Segments grow (and never shrink) while satisfying the contour fit constraints. When two segments have conflicting (expand) intentions, these conflicts are resolved centrally (which violates the autonomy of the agents). Once a region has occupied a part of the image there is no way in which it can become part of another segment later on.

In contrast with [12] our method doesn't need to have the number of segments nor their positions to be known a priori. Moreover, we resolve the segment expansion conflicts locally, and we dynamically reconsider segment boundaries. Regarding establishing the inter-segment boundaries, we state that in the following order, our proposed dynamical population algorithm, the distributed segmentation system in [12], and classical region growing algorithms, can be said to be increasingly greedy.

3 Problem Statement

The image segmentation problem is concerned with partitioning the image into non-intersecting regions that are connected and homogeneous with respect to basic image characteristics like grey values, color or texture, while the union of adjacent regions is not homogeneous. For simplicity reasons, we deal with grey-valued images and will not consider texture in this paper.

We represent the image in a undirected graph $G$, where the vertices represent the pixels, $N$ in total. In the image the pixels form a regular grid. We define contiguous vertices as being contiguous on that grid, so that every vertex has exactly four contiguous vertices. We write $v_1 \equiv v_2$ if $v_1$ and $v_2$ are contiguous. Only if two vertices are contiguous, there can be an edge connecting them. That is, every vertex can have at most four edges to contiguous vertices. Now a candidate segmentation graph is a graph for which the following holds: if there
is a path between any two vertices and the vertices are contiguous, then the vertices are adjacent, i.e. there is an edge between the vertices (see Fig. 1).

![Fig. 1](image_url)

**Fig. 1.** (a) is an image containing a background and two 'objects', making three segments in total. (b) shows the corresponding segmentation graph and a blown up detail, where $v$ has two adjacent vertices $v_1$ and $v_2$ and four contiguous vertices, $v_1, v_2, v_3,$ and $v_4$.

Every maximal connected subgraph (component) in the candidate segmentation graph is a candidate segment $S_i$, where $1 \leq i \leq n$ (and $n$ being the number of segments). A candidate segment has a set of vertices $V(S_i) = \{v_1, \ldots, v_{m_i}\}$, where $m_i$ is the number of vertices in $S_i$ and every vertex has a grey value attribute $g(v_j)$. We define the quality of a candidate segment\(^1\) as the variance $\sigma_i^2$ on the grey value attributes of the vertices:

$$\sigma_i^2 = \frac{1}{m_i} \sum_{j=1}^{m_i} [g(v_j) - \mu_i]^2, \text{ where } \mu_i = \frac{1}{m_i} \sum_{j=1}^{m_i} g(v_j)$$

(1)

The segmentation problem is to find that segmentation graph for which the overall variance is minimal. Therefore we need to find the graph that minimizes the criterion $C(G)$, where the number of segments $n$ is unknown:

$$C(G) = \sum_{i=1}^{n} \sigma_i^2$$

(2)

If the number of segments is completely free then a trivial and optimal solution is $N$ segments (one for each vertex (pixel)), which makes $C(G) = 0$. For most natural images this will be the only solution, since noise in the image as well as gradual changes will cause contiguous vertices to have different grey-values. (Hence, merging them into one segment will violate the homogeneity constraint.)

\(^1\) In the remainder, we leave out the 'candidate' addition for the segmentation graph and the segment subgraphs.
Clearly, this is not a desired solution, since we aim at a high dimension reduction (low number of segments). Because this effect is mostly due to noise in the images, a lower limit can be set on the homogeneity criterion, i.e. each segment has a certain minimum grey-value variance $\sigma_{\text{min}}$, that forces contiguous segments with a low combined variance to be merged. The determination of a reasonable $\sigma_{\text{min}}$ is related to the kernel width estimation for non-parametric density estimators [13], [15]. In this study $\sigma_{\text{min}}$ is given together with the image to segment.

4 Algorithmic Approach

To find the overall minimum criterion $C(G)$, we repeatedly consider the migration of a vertex from one segment to another. The migration is considered as contributing to the minimization of $C(G)$ if the sum of the variances of the two segments considered becomes smaller by migrating the vertex. Clearly, in this way $C(G)$ decreases monotonically. We assume that we will find a reasonable estimate of the minimum of $C(G)$ in this way, and that the final segmentation graph represents a good segmentation of the image. In the experiments section, we elaborate on this.

Initially, the number of segments is much higher than the number of actual segments in the image. The segments are regularly and densely spread over the image. After initialization, the segments only grow and shrink with one vertex at a time.

In the algorithm we separate three phases. In the first phase, the creation phase, the segment population is created. Then, during the competition phase, segments compete for the limited space in a sequence of epochs. In the final termination phase, convergence is detected and the competition halts.

4.1 Creation

Initially, we define the $n$ segments $S_i$ at regular distances $d$ in the vertex grid containing one vertex each. The remaining vertices are contained in a vertex collector called the world $S_W$. The vertex collector $S_W$ does not adhere to the segment constraints, i.e. during the processing it may become disconnected, nor does it compete with other segments for the possession of vertices. $S_W$ becomes the empty set after the processing of approximately $d^2$ epochs. When $S_W$ becomes empty, $G$ becomes and will remain a valid segmentation graph.

4.2 Competition

The competition phase is a sequence of epochs. In every epoch $n_a$ expansions will be considered, where $n_a$ is the number of active segments. An active segment is a segment which is not empty, so that $n_a \leq n$. For the selection of a candidate that may consider expansion, we need to have an expansion fitness measure. The segment quality, which is defined as the variance of the vertices, is not useful for
this measure because it is not an indication for successful expansion. A better fitness is an estimate of the expected result of an expansion trial. Therefore, we compute the success rate \( r(S_i) \) of each segment, which is defined as the average number of successful expansion \( e(k, S_i) \) in the last \( n_e \) expansion trials:

\[
    r(S_i) = \frac{1}{n_e} \sum_{k=1}^{n_e} e(k, S_i),
\]

where \( e(k, S_i) \in \{0, 1\} \). We add a small fraction \( r_e \) to the success rate, to ensure that segments with a zero success rate still have a small chance of being selected for expansion. This gives the following expansion fitness per segment:

\[
    E(S_i) = r(S_i)(1 - r_e) + r_e
\]

The reason for adding the \( r_e \) fraction is that over the course of the competition the chances of a segment may change. Without this fraction a segment would be excluded from expansion, when the success rate degrades to \( r(S_i) = 0 \). Consequently, the addition of \( r_e \) makes the algorithm less greedy. To select a number of segments to allow for an expansion trial, we use stochastic uniform sampling [3] on the expansion fitness \( E(S_i) \) of the active segments.

Since segments must be connected, they can only expand at the vertices that have a vertex degree less than four (recall that the degree of a vertex \( d(v) \) is the number of edges connected to it). For efficiency reasons we maintain a set of contiguous segment labels:

\[
    L_c(S_i) = \{S_j | \exists v_1 \in V(S_i), \exists v_2 \in V(S_j) : v_1 \Leftarrow v_2\}
\]

We also define a tuple of contiguous vertices in each segment,

\[
    V_c(S_i, S_j) = \{v_1 \in V(S_j) | \exists v_2 \in V(S_i) : v_1 \Leftarrow v_2\},
\]

where the vertices in the tuple \( V_c(S_i, S_j) \) are ordered clock-wise around \( S_j \).

In the next sections, we continue the description of the expansion trial for a segment that consists of three phases; contiguous vertex selection, followed by vertex negotiation and eventually possible segment update. The segment that attempts to expand, we call the initiator \( S_I \).

**Vertex selection** A vertex is selected by first selecting a segment from \( L_c(S_I) \). If \( S_W \) is contiguous to \( S_I \) (\( S_W \in L_c(S_I) \)), then it will be selected immediately as target \( S_T \). Otherwise the target segment will be selected proportional to the expected expansion success. Therefore, we differentiate the success rate \( r(S_i) \) per contiguous segment and again add a small fraction \( r_e \) as in Eq. [4] to the success rate. By means of a roulette wheel selection scheme [7], we select a target segment \( S_T \) from \( L_c(S_I) \). Then, we select a target vertex \( v_T \) uniform randomly from \( V_c(S_I, S_T) \).

Since a segment subgraph must satisfy the connectivity constraint, vertex migrations from one segment to the other, that violate this constraint, are not
allowed. Therefore, we check if the deletion of $v_T$ from $S_T$ would divide this
subgraph into two disconnected components, in other words, we check whether
$v_T$ is a cut-vertex°. If $v_T$ is a cut-vertex then we search in the tuple $V_c(S_I, S_T)$
for the closest non-cut-vertex.

**Vertex negotiation** Once we have selected a target vertex, the vertex nego-
tiation starts. The negotiation scheme is cooperative, that is, if the segment
update is favorable for the ensemble of the two segments, only then they agree
upon exchange. To this end, both $S_I$ and $S_T$ compute their current variances.
Additionally, they compute the segment variance for the hypothetical case that
the vertex would migrate from $S_T$ to $S_I$. There are three conditions that make
both segments agree upon segment update.

Condition I: *improved quality:*

$$\sigma_I + \sigma_T > \sigma'_I + \sigma'_T,$$  \hspace{1cm} (7)

where $\sigma_I$ and $\sigma_T$ are the variances before and $\sigma'_I$ and $\sigma'_T$ are the variance after
migration. This condition takes care of the variance minimization. If $S_I$ could
not take over $v_T$ because it apparently fits $S_T$ too well, it tries to meet the
following condition.

Condition II: *contiguous homogeneity:*

$$\sigma'_{I\cup T} \leq \sigma_{\text{min}}$$  \hspace{1cm} (8)

The condition of contiguous homogeneity lets segments merge, when their com-
bined variance is small. That is, segments decide to form a coalition if their union
is homogeneous, and hence, they expect to struggle for the same vertices.

Condition III: *occasional defect:*

$$U(0, 1) < r_d,$$  \hspace{1cm} (9)

where $U(0, 1)$ is a number uniformly random generated in the interval $[0,1]$.

Occasionally, we allow a vertex to migrate even if it is not favorable for the
ensemble of segments. The reason is that because of the grain granularity of the
migration process, local minima may stop the vertex exchange between contigu-
ous segments. The defect ratio $r_d$ regulates the escape from these situations.

**Segment update** Once the segments agree, both segments are updated ac-
cordingly. In case of vertex migration, $S_T$ removes all edges from its subgraph,
that are connected to $v_T$, and $S_I$ inserts edges in its subgraph between the $v_T$
and all contiguous vertices. In case of merging, $S_I$ inserts all relevant vertices
and edges into its subgraph. In both cases, the sets $V_c(S_I, S_T)$, and $V_c(S_T, S_I)$
and possibly $L_c(S_I)$ and $L_c(S_T)$ are updated.

° Clearly, as contiguous vertices always become adjacent, adding a contiguous vertex
to $S_I$ can never divide it up into two components.
4.3 Termination

The population algorithm terminates when the success rate of all segments equals zero. That is, every segment failed its last $n_e$ expansion trials. For the question whether this will happen, we don't take Condition III into account, because this condition exactly aims at getting out of (local) minima. Otherwise, this termination criterion will certainly be met, since Condition I enforces a monotonically decreasing sum of segment variances and the variance has a lower limit of zero. Although Condition II may lead to temporarily increase in overall variance, it will in the long term also contribute to convergence. Namely, when Condition II applies, the number of segments decreases and the number of segments is finite.

5 Experiments

Since ground truth is a difficult issue in image segmentation [13], we validate our claims with respect to the problem statement by testing if the proposed method succeeds in partitioning the image while satisfying the segmentation constraints as formulated. Here we show the results for an artificial image and a natural image. In both experiments we fixed $d = 5$, $r_e = 0.05$, $n_e = 50$ and $r_d = 0.05$. Both experiments are run several times, but we only show the results of one run, since they are similar. This implies that for these examples, the method is not sensitive to its initial conditions.

In the first experiment, we use a synthetic 100x100 image consisting of four regions having grey value distributions $N(50, 10^2)$, $N(80, 10^2)$, $N(110, 10^2)$, and $N(150, 10^2)$ respectively, where $N(\mu, \sigma^2)$ is the normal distribution. Thus, all regions have different means but the same variances. We set $\sigma_{min} = 12$. The results in Fig.2 show that the initial 64 segments gradually merged into the correct four segments.

![Fig. 2](image.png)

In the second experiment we use a 64x64 real-life image. We have set $\sigma_{min} = 25$, because the objects and background are more irregular than in previous experiment, i.e. there is higher variance. Fig.3 illustrates the results. Again, it can be seen that the initial (144) candidate segments are reduced to a low number of
segments. Most objects, however, still have small contour 'segments' surrounding them. Clearly, this leaves room for improvements in the homogeneity criterion. We discuss some of them in the next section.

Although still in its infancy, the experiments support the possibilities of the proposed population algorithm. The experiments clearly show that the population competes over the limited space. It further shows that the dynamic reconsiderations of local criteria produces true boundaries between the final segments.

![Fig. 3.](a) is the input image consisting of approximately 18 segments (17 foreground and the background). (b), (c), (d), and (e) show the segmentation results after 50($n_a = 89$), 100($n_a = 53$), 150($n_a = 41$), and 500($n_a = 30$) epochs respectively.

6 Discussion

In this paper, we proposed a new image segmentation scheme. In this scheme candidate segments act as autonomous entities that compete each other locally and dynamically for the limited image space. Experiments showed the effectiveness of the paradigm, although the natural image example showed that the number of segments sometimes remains too high. Here, we want to emphasize the opportunities of this paradigm by discussing some future enhancements that exploit its local behavior.

Shape and homogeneity constraints. By collecting pixels, the segments already differentiate, but we want to extend this in a number of ways. First, we intend to incorporate a measure that constrains the shape of the segments. For now any shape is allowed as long as it improves the variances of the segment itself and the contiguous segments. Second, we plan to include texture descriptions, such that regular grey value variations will not degrade the segment quality.

Local noise estimation. Currently, we allow for a certain inhomogeneity $\sigma_{min}$ in each segment. Estimating the $\sigma_{min}$ parameter makes the method more flexible, while the method also allows for local differentiation in $\sigma_{min}$.

Vertex selection by sampling. We foresee modifications with respect to the vertex selection. Currently, vertices are randomly selected from $V_c(S_I, S_T)$. Sampling a set of vertices and then selecting the best will probably improve the method. Especially, it will result in a better initial situation, when the world $S_W$ is explored more selectively.
The new image segmentation scheme is an application of our introduced population algorithm that is characterized by the competition within a population for limited space ('data') by considering local criteria dynamically. It is challenging to investigate whether this metaphorical optimization scheme can be successfully applied to other domains as well.

References

Constraint Handling Techniques
An Adaptive Algorithm for Constrained Optimization Problems

S. Ben Hamida and Marc Schoenauer

CMAP, URA CNRS 756, Ecole Polytechnique, Palaiseau 91128, France;
sana@cmap.polytechnique.fr
Marc.Schoenauer@polytechnique.fr

Abstract. Adaptivity has become a key issue in Evolutionary Algorithms, since early works in Evolution Strategies. The idea of letting the algorithm adjust its own parameters for free is indeed appealing. This paper proposes to use adaptive mechanisms at the population level for constrained optimization problems in three important steps of the evolutionary algorithm: First, an adaptive penalty function takes care of the penalty coefficients according to the proportion of feasible individuals in the current population; Second, a Seduction/Selection strategy is used to mate feasible individuals with infeasible ones and thus explore the region around the boundary of the feasible domain; Last, selection is tuned to favor a given number of feasible individuals. A detailed discussion of the behavior of the algorithm on two small constrained problems enlightens adaptivity at work. Finally, experimental results on eleven test cases from the literature demonstrate the power of this approach.

1 Introduction

The general nonlinear parameter optimization problem is defined as:

\[
\text{optimize } f(x), \quad x = (x_1, \ldots, x_n) \in \mathcal{F} \subseteq \mathcal{S} \subseteq \mathbb{R}^n,
\]

such that:

\[
\begin{align*}
& g_i(x) \leq 0, \text{ for } i = 1, \ldots, g \quad \text{inequality constraints} \\
& h_j(x) = 0, \text{ for } j = q + 1, \ldots, m \quad \text{equality constraints}
\end{align*}
\]

where \( f, g_i \) and \( h_j \) are real-valued functions on the search space \( \mathcal{S} \). The satisfaction of the set of constraints \( \{g_i, h_j\} \) defines the feasible region \( \mathcal{F} \).

Though many specific methods to handle such constrained problems within Evolutionary Algorithms have been proposed (see e.g. [10] for a survey), this paper will concentrate on the penalty function method.

The idea of penalty methods is to penalize infeasible solutions by adding to the objective function \( f \) a positive quantity (when the goal is to minimize \( f \)) in order to decrease the quality of such infeasible individuals:

\[
eval(x) = \begin{cases} 
  f(x), & \text{if } x \in \mathcal{F} \\
  f(x) + \text{penal}(x), & \text{otherwise}
\end{cases}
\]
where \( \text{penal}(\mathbf{x}) \) is positive for minimizing and negative for maximizing.

The design of the penalty function \( \text{penal} \) is the main source of difficulty of penalty methods. The most popular approaches use measures of the constraint violations, such as:

\[
\text{penal}(\mathbf{x}) = \sum_{j=1}^{q} \alpha_j g_j^+(\mathbf{x}) + \sum_{j=q+1}^{m} \alpha_j |h_j| (\mathbf{x}),
\]

(1)

where \( x^+ \) denotes the positive part of \( x \). The real numbers \( \alpha_j, j = 1 \ldots m \) are called the \textit{penalty coefficients}, and the main difficulty is to determine appropriate values for each one of them.

This paper presents ASCHEA, the Adaptive Segregational Constraint Handling Evolutionary Algorithm, based on a population level adaptive way [3] to adjust the values of the penalty coefficients, an adaptive selection/seduction mechanism for mate choice and a specific feasibility-oriented selection operator. The idea is to maintain in the population both feasible and infeasible individuals: the search can either concentrate around the boundary of the feasible region, as it is well known that the optimum lies on that boundary in many real problems, or normally explore the feasible region otherwise. This is achieved by increasing the penalty coefficients if there are not enough feasible individuals in the population, keeping the feasible individuals in the population using segregated selection, and mating feasible and infeasible individuals together to explore the surroundings of the boundary.

The paper is organized as follows. Section 2 briefly surveys previous constraint handling methods based on penalty functions. Section 3 introduces the components of the algorithm and a priori discusses their advantages. Section 4 is devoted to a detailed study of the behavior of the algorithm on two selected test cases. Finally, an extensive experimental study is presented in section 5, giving some results obtained by this technique on some reference test problems taken from [10, 6].

2 Evolutionary penalty methods

First note that the general penalty approach to handle constraints is not specific to Evolutionary Computation, as any optimization method can be applied to the penalized objective function. For instance, the static methods reviewed here are very general. The dynamic methods could probably be adapted to any iterative optimization algorithm. But the other methods are very specific to Evolutionary Computation. Anyway, all methods will be presented in the framework of Evolutionary Computation.

2.1 Static methods

The penalty function is fixed once and for all at the beginning of the algorithm.

The \textit{"death penalty"} method simply rejects infeasible solutions from the population. Though not exactly a penalty method (the rejection takes place in the selection step), it can be viewed as a penalty method with infinite penalty
[1]. This method might have great difficulties to find even a first feasible point.

The static penalty methods use user-defined values for the penalty coefficients. Some improvement was brought in [4] by increasing these values with the level of violation – but the main difficulty results from the lack of any hint about the value these penalty coefficient should have.

2.2 Dynamic methods

In these methods, the penalty coefficients are modified along evolution according to a user-defined schedule – usually increased in order to ensure feasible individuals in the end. These methods suffer the same defect than static penalty, i.e. it is not easy to tune the values of their parameters.

In the dynamic penalty method [5], the penalty coefficients increase like $C \times t^\beta$, for user-defined $C$ and $\beta$ ($t$ is the current generation).

Genocop II [8] uses the method of annealing penalties: the penalty coefficients are computed by $\frac{1}{2^{t(t)}}$ with a “freezing temperature” ($\tau$) which decreases every generation. Note that Genocop II also uses for linear constraints a set of special operators maintaining feasibility of solutions [9].

2.3 Adaptive methods

In these methods, information is gleaned from the population to update the value of the penalty functions.

A first example of adaptive method is the method based on the superiority of feasible points [11]: Some positive term, depending on the current infeasible individuals, is added to the constraint violations in the penalty function, ensuring any feasible individual will be better than any infeasible individual.

Different adaptive penalty methods have been proposed. In [2], the penalty coefficient is increased (resp. decreased) if the best individual is infeasible (resp. feasible) during a given number of generations. In [15], the penalty is proportional to the difference between the best feasible objective function ever seen and the best overall objective value ever seen - the first goal is to find at least one feasible solution. However, both method rely only on the best individuals in the past populations to adjust the penalty, and this clearly might not reflect precisely what is going on at the population level.

2.4 Yet another penalty method

The segregated GA [7] tries to make a balance between heavy and moderate penalties. It uses a specific selection strategy whose effect is to somehow maintain two cooperating subpopulations: A first subpopulation is evaluated using large values of penalty parameters, thus containing mostly feasible individuals; A second subpopulation uses small values for penalty coefficients, and hence is likely to contain mostly infeasible individuals.
3 The adaptive segregational algorithm

This section introduces the original Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA). The main idea in ASCHEA is to maintain both feasible and infeasible individuals in the population, at least when it seems necessary. In order to achieve this goal, three main ingredients will be used: An original adaptive penalty method that uses global information of the population to adjust the penalty coefficients; a constraint-driven recombination, where in some cases feasible individuals can only mate with infeasible individuals; a segregational selection that distinguishes between feasible and infeasible individuals.

In the rest of the paper, $\tau_t$ will denote the proportion of feasible individuals in the population at generation $t$, and $\tau_{\text{target}}$ will be a user-defined proportion. The idea of the proposed method is to maintain $\tau_t$ as close as possible from $\tau_{\text{target}}$ along evolution.

3.1 Population-based adaptive penalty

The penalty function used here is that of equation (1). For the sake of simplicity, only the case of a single coefficient will be considered - whether there is a single constraint, or all penalty coefficients are set to the same value $a(t)$ at generation $t$.

Increasing the value of the penalty coefficient $a$ in equation (1) clearly favors feasible individuals in subsequent selections, while decreasing it favors infeasible individuals. Hence, in order to try to maintain a given proportion $\tau_{\text{target}}$ of feasible individuals (and hence $1 - \tau_{\text{target}}$ of infeasible individuals!) the following strategy is used for coefficient $a$:

$$\begin{align*}
\text{if } (\tau_t > \tau_{\text{target}}) & \quad a(t+1)(x) = a(t)/\text{fact} \\
\text{otherwise} & \quad a(t+1)(x) = a(t) * \text{fact}
\end{align*}$$

where $\text{fact} > 1$ is a user-defined parameter.

The initial penalisation $a(0)$ is computed using the first population, to try to balance between objective function and constraint violation:

$$a(0) = \left[ \frac{\sum_{i=1}^{n} f(x_i)}{\sum_{i=1}^{n} v(x_i)} \right] * 1000,$$

where $v(x_i)$ is the sum of the constraint violations of individual $x_i$.

3.2 Constraint-driven mate selection

In many real-world problems, it is known that the constrained optimum lies close to the boundary of the feasible domain (e.g. when minimizing some cost with technological constraints). On the other hand, restricting the search to that boundary, though very powerful when applicable [13], is highly problem-dependent. Moreover, it might prove too restrictive, too, as the solution sometimes lies very close to, but outside from, that boundary.

In order to both achieve better exploration of the boundary region and attract infeasible individuals more rapidly toward feasible ones, it is proposed to
use a selection/seduction mechanism [12], choosing the mate of feasible individuals to be infeasible. However, to also allow exploration of the feasible region, this mechanism is only applied when too few (with respect to \( T_{\text{target}} \)) feasible individuals are present in the population: otherwise, i.e. if \( T_t > T_{\text{target}} \), it is assumed that exploration should proceed normally – and the mate is chosen in the whole population. More precisely, to select the mate \( x_2 \) for a first parent \( x_1 \):

\[
\text{if } (0 < T_t < T_{\text{target}} ) \text{ and } (x_1) \text{ is feasible} \\
\text{select } (x_2) \text{ among infeasible individuals only} \\
\text{else select } (x_2) \text{ according to fitness only}
\]

Such strategy can of course be used together with any recombination operator.

### 3.3 Segregational selection

To further enhance the chances of survival of feasible individuals, a specific selection operator is used in the algorithm. This selection, called segregational selection, can be viewed as intermediate between the method based on the superiority of feasible points (see section 2.3 and [11]) and the replacement method used in the Segregated GA [7] (see section 2.4).

The segregational selection is a deterministic replacement mechanism used in an ES-like scheme [14]: from a population of \( \mu \) parents, \( \lambda \) offspring are generated (all parents giving birth to \( \frac{\lambda}{\mu} \) offspring on average). Among those \( \lambda \) offspring (for the "+" strategy) or among the \( \mu \) parents plus the \( \lambda \) offspring (for the "-" strategy), \( \mu \) individuals are selected to become the new parents the following way:

First, let \( \tau_{\text{select}} \) be a user-defined proportion: the segregational selection starts by selecting without replacement among feasible individuals, based on their fitness, until \( \tau_{\text{select}} * \mu \) have been selected, or no more feasible individual is available. The population is then filled using standard deterministic selection on the remaining individuals, based on the current penalized fitness.

So only a proportion \( \tau_{\text{select}} \) of feasible individuals is considered superior to all infeasible points. Moreover, the above segregational selection can be viewed as a Segregated GA [7] where the large penalty coefficient would be set to \( +\infty \).

### 3.4 Discussion

A first consequence of the use of the segregated selection is some sort of feasible-elitism: as soon as a feasible individual appears, it can only disappear from the population by being replaced by a better feasible guy, even if the penalty coefficient later reaches very small values and greatly favors infeasible individuals.

This in turn ensures that the constraint-driven mate selection can play its role, which is to accelerate the movement toward the feasible region of infeasible individuals, and to explore in detail the region close to the boundary of the feasible domain.

But all the above mechanisms are activated based on the proportion of feasible in the population. This enables the proposed algorithm to adaptively handle both the case where the optimum lies on the boundary of the feasible region, and the case where the optimum lies inside the feasible region. Indeed, in the first case, the constraint-driven mate selection allows to explore both sides of
the boundary. And in the second case, it is likely that the whole population will rapidly tend to enter the feasible region, and standard "blind" mate selection will be applied while the segregational selection will not be different from pure deterministic selection for the feasible fraction of the population.

4 Experimental behavior

This section studies the application of ASCHEA to two test-cases in detail, selected from [6]. The first function (G6) has 2 nonlinear inequality constraints and its optimum lies on the boundary of the feasible region (Fig. 1, left). The second one (G8) has also two nonlinear inequality constraints and 2 quasi-optima inside the feasible region (Fig. 1, right).

![Fig. 1. G6 (left) and G8 (right) landscapes. The constraint boundaries have been artificially made visible. Note that function G8 has huge peaks toward the x2 axis that are not shown here.](image)

4.1 Experimental conditions

ASCHEA uses here a (100+300)-ES segregational selection strategy (section 3.3). Mutation is the standard Gaussian mutation and self-adaptive standard deviations [14], with initial values of 0.03, global learning rate of 0.1 and local learning rate of 1. Recombination is the standard arithmetical crossover – plus of course constraint-driven mate selection when triggered (section 3.2). For all experiments, $\tau_{select}$ is set to 0.3, $\tau_{target}$ to 0.6 and $fact$ to 1.1, the crossover and mutation rates are set to 0.9. At least 31 independent runs of 5000 generations are performed for all tests.

4.2 The solution lies on the boundary

For each trial, the optimum of G6 is found pretty quickly (Fig. 2 and 3). In the first generations, as soon as feasible individuals are found, the segregational selection keeps them in the population. The constraint-driven mate selection is then applied, which rapidly increases the proportion of feasible points in the population and aggregates some individuals close to the boundary of the feasible region. Fig. 2 shows how the algorithm helps the individuals to enter rapidly (5 generations) into the feasible region.
The deterministic ES-like selection increases the exploitation capability of the algorithm. But, as can be seen on Fig. 3, exploration continues around the boundary, and even in the infeasible region, thanks to the adaptive penalisation strategy. The peaks in the figure demonstrate that the optimum of the fitness can change (low objective values correspond to infeasible individuals). However, the best feasible individual is never lost.

### 4.3 The solution lies inside the feasible domain

As for the first test, 31 independent runs are performed on function G8. As can be seen on Figure 4-left, whereas all individuals of generation 3 are still widely spread over both feasible and infeasible domains, the whole population has become feasible at generation 5. Later, at generation 10, the population is already concentrated around the optimum. The evolution then proceeds normally with standard deterministic selection, while the penalty coefficient decreases rapidly. Hence, when a first mutant hits an outside peak (the one that can be seen on Fig. 1), it quickly attracts part of the population (Fig. 4-right). However, thanks to the segregated selection, a fraction $T_{select}$ of the population remains feasible, and the whole population has converged in less than 30 generations (not shown). Note however that if evolution proceeds, this phenomenon will happen again.
Fig. 4. Evolution of population during G8 optimization: at generation 3 and 5 (resp. stars and squares on left) and at generation 10 and 20 (resp. stars and squares on right). While the population had started to converge at generation 10, a lucky mutation toward the closest peak outside the feasible domain has drawn many individuals toward infeasibility. Thanks to the segregated selection and the mate selection, the population will be back around the optimum rapidly.

5 Extensive experimental results

In this section, in order to demonstrate the robustness of our approach, experiments are made for the eleven benchmark functions proposed in [10] and [6] (from G1 to G11). For each one, 31 independent runs were performed using the parameters described in section 4.1 – except for function G10 where meaningful results could only be found using a mutation rate of 0.3.

The results of all experiments are summarized in Table 1. All runs could find a feasible solutions. The exact optimal solutions were found for 7 problems out of 11.

The first conclusion of these results is that ASCHEA gave satisfactory results for all test cases, except for the test case G5, where the replacement of both equality constraints by inequalities did not prove efficient. The problem seems to come from the use of a single penalty coefficient. Further work will use one penalty coefficient for each constraint.

Moreover, it is possible to further enhance the performances of the algorithm by a more precise tuning of its parameters. For example, for test case G3*, the number of successful runs increases when the algorithm is run for longer time, as expected! Table 2 shows the influence of the number of generations on the quality of the results.

Another parametric study should (and will) be devoted to parameter $\tau_{\text{target}}$: it can have a great influence on the quality of the results, as for instance for test case G2: higher values of $\tau_{\text{target}}$ and $\text{fact}$ gave better results than those of Table 1; With $\tau_{\text{target}} = 0.7$ and $\text{fact} = 1.2$, the algorithm gave as best result 0.803603 with average 0.606.

In the opposite, on test case G4, ASCHEA gave better results with lower values of $\tau_{\text{target}}$. But this is probably due to the fact that the optimum lies on the boundary of the feasible region, a lower $\tau_{\text{target}}$ value emphasize exploration of the feasible domain boundary, as confirmed by the results of Table 3.
Table 1. Summary of results on 11 test cases, giving the best, the median and the average best (feasible) objective value out of 31 runs after 5000 generations (the 3 last columns). When the optimal solution could be found at least once, the first 3 columns give the number of hits, the best and the average time needed to find the optimum (for the successive runs). The test cases G2 and G3 were run with respectively 20 and 10 variables. The stars denote that each equality constraint has been turned into one inequality constraint - the solution staying the same.

<table>
<thead>
<tr>
<th>Function</th>
<th>Opt. value</th>
<th>Opt. found</th>
<th>After 5000 gen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hits</td>
<td>best</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(nb gen)</td>
<td>Avg. (nb gen)</td>
</tr>
<tr>
<td>G2</td>
<td>0.803553</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G3*</td>
<td>1.0</td>
<td>1</td>
<td>4205</td>
</tr>
<tr>
<td>G4</td>
<td>-30665.5</td>
<td>7</td>
<td>109</td>
</tr>
<tr>
<td>G5*</td>
<td>5126.4981</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G6</td>
<td>-6961.81</td>
<td>31</td>
<td>42</td>
</tr>
<tr>
<td>G7</td>
<td>24.3062</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G8</td>
<td>0.095825</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td>G9</td>
<td>680.630</td>
<td>1</td>
<td>4267</td>
</tr>
<tr>
<td>G10</td>
<td>7049.33</td>
<td>31</td>
<td>27</td>
</tr>
<tr>
<td>G11*</td>
<td>0.75</td>
<td>31</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 2. Results of the algorithm on test case G3* for different generation number.

<table>
<thead>
<tr>
<th>Target</th>
<th>Avg.</th>
<th>Hits</th>
<th>Avg.</th>
<th>Hits</th>
<th>Avg.</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0.99984</td>
<td>1</td>
<td>0.99990</td>
<td>3</td>
<td>0.99999</td>
<td>7</td>
</tr>
<tr>
<td>0.55</td>
<td>-30650.745</td>
<td>7</td>
<td>-30656.56</td>
<td>10</td>
<td>-30653.15</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3. Results of the algorithm on test case G4 for different Target values.

<table>
<thead>
<tr>
<th>Target</th>
<th>Avg.</th>
<th>Hits</th>
<th>Avg.</th>
<th>Hits</th>
<th>Avg.</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>-30650.745</td>
<td>7</td>
<td>-30665.15</td>
<td>15</td>
<td>-30658.93</td>
<td>20</td>
</tr>
<tr>
<td>0.45</td>
<td>-30653.15</td>
<td>15</td>
<td>-30658.93</td>
<td>20</td>
<td>-30664.53</td>
<td>26</td>
</tr>
<tr>
<td>0.35</td>
<td>-30654.62</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Otherwise, for the test case G7 and G9, even if the algorithm didn’t found the optimum, the best solutions were very close to the optimum value: all results are between 24.36 and 25.75 for G7, and between 680.63 and 680.7 for G9 - and could probably be improved by a careful tuning of the parameters.

6 Conclusion and further work

This paper has introduced ASCHEA, an original evolutionary algorithm for constrained optimization. ASCHEA tries to maintain a given proportion of feasible individuals in the population using an adaptive penalty mechanism coupled with a segregational selection that favors a given number of feasible individuals and a constraint-based choice of mate that allows the detailed exploration of both side of the boundary of the feasible region. However, ASCHEA is also able to explore the interior of the feasible domain as all these features are adaptive, and only triggered when few feasible individuals exist in the population.

ASCHEA gave very good results on eleven test cases from the literature. However, the present study is limited to a single penalty coefficient, and further work will extend the adaptive penalty to multiple penalty coefficients - one
per constraint. Moreover, more work is needed to tackle equality constraints. The first experiments indicate that, when using the standard way to turn the constraint \( h = 0 \) into the double inequality constraint \(-\varepsilon < h < \varepsilon\), these two inequalities should be handled independently. Further, the value of \( \varepsilon \) should be adjusted along evolution by making it adaptive.

References

Test-Case Generator TCG-2 for Nonlinear Parameter Optimisation

Martin Schmidt
Bording Data A/S, Bredskifte Alle 5, DK-8210 Aarhus V, Denmark
marsch@daimi.au.dk

Zbigniew Michalewicz
Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA
zbyszek@uncc.edu

Abstract. Experimental results reported in many papers suggest that making an appropriate choice of a method and its parameters to solve a nonlinear parameter optimisation problem remains an open question. The most promising approach at this stage of research seems to be experimental, involving a design of a scalable test suite of constrained optimisation problems. Then it would be possible to evaluate merits and drawbacks of the available optimisation methods as well as test new methods efficiently. In this paper we discuss the new test-case generator TCG-2 for constrained parameter optimisation techniques. The TCG-2 is capable of creating various test problems with different characteristics, including the dimensionality of the problem, number of local optima, number of active constraints at the optimum, topology of the feasible search space, etc. This test-case generator is very useful for analysing and comparing different constraint-handling techniques and different nonlinear parameter optimisation techniques.

1 Introduction

The general nonlinear programming (NLP) problem is to find $\vec{x}$ so as to

$$\text{optimize } f(\vec{x}), \vec{x}=(x_1,\ldots,x_n) \in \mathbb{R}^n,$$

Equation 1

where $\vec{x} \in F \subseteq S$. The objective function $f$ is defined on the search space $S \subseteq \mathbb{R}^n$ and the set $F \subseteq S$ defines the feasible region. Usually, the search space $S$ is defined as an $n$-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds):

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n,$$

where the feasible region $F \subseteq S$ is defined by a set of $p$ additional constraints $p \geq 0$:

$$g_j(\vec{x}) \leq 0, \quad \text{for } j=1,\ldots,q, \text{ and } h_j(\vec{x}) = 0, \quad \text{for } j=q+1,\ldots,p.$$
At any point $\bar{x} \in F$, the constraints $g_j$ that satisfy $g_j(\bar{x}) = 0$ are called the *active* constraints at $\bar{x}$.

The NLP problem, in general, is intractable: It is impossible to develop a deterministic method for the NLP in the global optimisation category, which would be better than the exhaustive search [1]. This makes a room for evolutionary algorithms, extended by some constraint-handling methods. Indeed, during the last few years, several evolutionary algorithms that aim at complex objective functions (e.g., non-differentiable or discontinuous) have been proposed for the NLP; a recent survey paper [3] provides an overview of these algorithms.

It is not clear what characteristics of a constrained problem make it difficult for an optimisation technique. Any problem can be characterised by various parameters; these may e.g. include the number of linear constraints, the number of nonlinear constraints, the number of equality constraints, the number of active constraints, the ratio $\rho = |F|/|S|$ between the size of the feasible search space and the whole search space, the type of the objective function (the number of variables, the number of local optima, the existence of derivatives, etc).

In [3] eleven test cases for constrained numerical optimisation problems were proposed ($G1$–$G11$). The results of many tests did not provide meaningful conclusions, since no single parameter proved to be significant as a major measure of difficulty of the problem. In [3] the authors wrote: “It seems that the most promising approach at this stage of research is experimental, involving the design of a scalable test suite of constrained optimisation problems, in which many [...] features could be easily tuned. Then it should be possible to test new methods with respect to the corpus of all available methods.”

Clearly, there is a need for a parameterised test-case generator that can be used for analysing various methods in a systematic way (rather than testing them on a few selected test cases; moreover, it is not clear whether addition of a few extra test cases is of any help).

There were some attempts in the past to propose a test case generator for unconstrained parameter optimisation [5], [6]. We are also aware of one attempt to do so for constrained cases; in [2] the author proposed so-called stepping-stones problem defined as:

Objective: maximise $\sum_{i=1}^{n} (x_i / \pi + 1)$,

where $-\pi \leq x_i \leq \pi$ for $i=1,...,n$ and the following constraints are satisfied:

$e^{x_i / \pi} + \cos(2x_i) \leq 1 \quad \text{for } i=1,...,n$.

Note that the objective function is linear and that the feasible region is split into $2^n$ disjoint parts (called stepping-stones). As the number of dimensions $n$ grows, the problem becomes more complex. However, as the stepping-stones problem has one parameter only it cannot be used to investigate some aspects of a constraint-handling method.

In [4] we reported on experiments with a test case generator TCG. This generator was capable of creating various test cases with different characteristics:

- Problems with different values of $\rho$: The relative size of the feasible region in the search space.
- Problems with different number and types of constraints.
- Problems with convex or non-convex objective function, with multiple optima.
- Problems with non-convex constraints consisting of (possibly) disjoint regions.

However, there were some problems with the TCG, so it was necessary to develop the new version, the TCG-2, to address these issues.

This paper presents the test-case generator TCG-2 with the following organisation: The next section describes the new TCG-2, section 3 presents experimental results whereas section 4 concludes the paper and indicates directions for future research.

## 2 Test-Case Generator TCG-2

This section presents the new TCG-2 that is a parameterised test-case generator of the following form:

\[ \text{TCG-2}(n, m, \rho, c, a, p, \sigma, \alpha, d) \]

where the parameters control:

- \( n \) - the dimensionality of the test function,
- \( m \) - the number of feasible components,
- \( \rho \) - the feasibility of the search space,
- \( c \) - the complexity of the feasible search space,
- \( a \) - the number of active constraints at global optimum,
- \( p \) - the number of peaks of the objective function,
- \( \sigma \) - the width of the peaks,
- \( \alpha \) - the decay of height of the peaks,
- \( d \) - the minimal distance between different components.

The ranges and the types of the parameters are:

- \( 1 \leq n; \text{integer} \)
- \( 1 \leq m; \text{integer} \)
- \( 0 \leq \rho \leq 1; \text{float} \)
- \( 0 \leq c \leq 1; \text{float} \)
- \( 0 \leq a \leq n; \text{integer} \)
- \( 1 \leq p; \text{integer} \)
- \( 0 \leq \sigma; \text{float} \)
- \( 0 \leq \alpha \leq 1; \text{float} \)
- \( 0 \leq d \leq 1; \text{float} \)

There are five major components of the TCG-2 and we discuss them in turn:

1. The search space.
2. The feasible search space.
3. The constraint violation function.
4. The landscape of the objective function.
5. The fitness landscape using a static penalty approach.

The search space is defined as an \( n \)-dimensional cube with the ranges \([0,1]\).

The general idea behind the TCG-2 is to randomly create non-overlapping feasible areas (referred to as boxes) in the search space. Many boxes are attached to each other in order to form feasible components. All components are disjoint and hold a total feasible area of \( \rho \times 1 \times 1 \). The higher the complexity \( c \) is, the more (probably smaller) boxes will be created (hence the higher the complexity). The feasible part of the search space is generated with the following constraint generation algorithm:
Create_Initial_Components(m, ρ, c, d)
Enlarge_Initial_Components(m, ρ, c, d)
Add_Complexity(m, ρ, c, d)

The above procedures are explained below:

- **Create_Initial_Components(m, ρ, c, d):** This procedure creates \( m \) disconnected feasible components (boxes) at random locations such that each initial component occupies a very small part of the total feasible search space, i.e. each initial component has the size \( 1\% \cdot ρ \cdot (1 - c) \cdot |S|/m \).

- **Enlarge_Initial_Components(m, ρ, c, d):** This procedure enlarges the existing feasible components by choosing one side of a box at random and trying to move it a little bit (if a collision occurs the enlargement is undone). This is repeated until the total feasible search space becomes \( ρ \cdot (1 - c) \cdot |S| \). This procedure adds no new components but enlarges existing ones and makes sure that all components remain disconnected.

- **Add_Complexity(m, ρ, c, d):** This procedure generates the complexity of the feasible search space by randomly generating and attaching boxes to the existing components, i.e. every new box can be attached to any existing box. This procedure also assures, that no matter how many boxes that are added there will never be any feasible continuous path from one feasible component to any other. Hence, we will have \( m \) disconnected feasible components and every component consists of one or more boxes. This procedure stops once the total feasible search space becomes \( ρ \cdot |S| \), and hence this procedure generates new boxes with a total size of \( ρ \cdot c \cdot |S| \).

All procedures generate different components such that the minimal distance between them is \( d \). Note that the constraint generation algorithm is stochastic. Therefore each procedure is designed such that it restarts if it gets stuck in an invalid arrangement of boxes. In general, if \( m ≥ 4 \) and \( ρ \) is close to 1 then the constraint generation algorithm has a long runtime since it is difficult to arrange the boxes correctly. In the extreme case of \( ρ = 1 \) and \( m > 1 \) it is impossible to create disjoint components and hence a feasible arrangement of \( m \) components is impossible. Also in the extreme case of a large feasibility \( ρ \) and a large minimal distance \( d \) it becomes impossible to generate a valid arrangement of boxes.

The results of the constraint generation algorithm are:

1. The larger the number of components \( m \) is the more (disconnected) feasible components are generated.
2. The larger the feasibility \( ρ \) is the larger the initial components are (as \( ρ \cdot (1 - c) \cdot |S|/m \)) and the larger the final feasible search space becomes (as \( ρ \cdot |S| \)).
3. The larger the complexity \( c \) is the smaller the initial components are (as \( ρ \cdot (1 - c) \cdot |S|/m \)) and the more boxes are added to the components (as \( ρ \cdot c \cdot |S| \)). The more boxes that are added to the components the more difficult the feasible search space becomes. Hence, the larger the complexity \( c \) is the more complex and difficult the feasible search space becomes.
4. The larger the minimal distance \( d \) is the more separated the feasible components become, and hence they will form more and more separated feasible island.
The following figures present examples of generated constraints by the above algorithm.

Figure 1 present generated constraints for the TCG-2( n=2, m=2, ρ=0.5, c=0.5, a=2, p=3, σ=0.4, α=0.5, d=0.01 ) and one can see that the size of the two feasible components is different and that there are a different number of boxes attached to the two components. Further all boxes that belong to the same component are attached to each other but they do not overlap. Boxes that belong to different components are disconnected. The global optimum at (0.2,0.8) has two active constraints in agreement with a=2.

\[ CV(\vec{x}) = \begin{cases} 0 & \text{if } \vec{x} \text{ is inside a box} \\ |\vec{c}_{box} - \vec{x}| & \text{otherwise} \end{cases} \]

where \(\vec{c}_{box}\) is the closest centre of the feasible boxes and \(\vec{x}\) is the vector to evaluate.

After creation of the feasible components the objective function \(G(\vec{x})\) is defined as:

\[ G(\vec{x}) = g_i(\vec{x}) \text{ where } \forall j: \|\vec{x} - \vec{c}_j\| \leq \|\vec{x} - \vec{c}_i\| \]

and where

\[ g_k(\vec{x}) = h_k \exp\left(-\frac{|\vec{x} - \vec{c}_k|^2}{2\sigma^2}\right) \quad \alpha \leq h_k \leq 1 \quad k = 1, \ldots, p. \]

Equation 2 defines the objective function \(G(\vec{x})\) using a set of \(p\) randomly placed gaussians \(g_k(\vec{x})\) where \(h_k\) is the height of peak \(k\) and \(\vec{c}_k\) is the centre of peak \(k\). In order to evaluate \(G(\vec{x})\) the closest centre \(\vec{c}_i\) is found and then the function \(g_i(\vec{x})\) is

---

1 The centre of a feasible box is simply the middle of the box.
evaluated. All centres $\mathcal{C}_k$ are placed randomly in the search space with the exception of the global optimum that is placed such that there are exactly $a$ active constraints at the global optimum. All peaks heights $h_k$ are evenly distributed in the range $[a, 1]$ such that the global optimum has the highest peak $h_k = 1$ while the lowest peak has the height $h_k = a$. Note that the global optimum is always placed either inside a feasible area (if $a = 0$) or at the border of a box (if $a > 0$). Hence the feasible global optimum always has the value $G(\bar{x}) = 1.0$ and the constraint violation value $CV(\bar{x}) = 0$.

The very common static penalty approach uses the presented objective function $G(\bar{x})$ and constraint violation function $CV(\bar{x})$, and combines them as follows:

$$fitness(\bar{x}) = G(\bar{x}) - W \times CV(\bar{x}),$$

where the penalty constant $W > 0$.

The following three figures show an example of a generated objective function (Figure 2), a constraint violation function (Figure 3) and the resulting fitness landscape (Figure 4) for $TCG-2( n=1, m=3, \rho=0.5, \sigma=0.0, a=1, p=3, \alpha=0.5, d=0.05 )$ and $W = 10$. It is quite obvious that the resulting fitness landscape (Figure 4) is multimodal with three feasible island according to $m=3$. Note that the global peak (around 0.3) is located at the (right) edge of the left-most feasible box which is according to $a = 1$, i.e. one active constraint at the global optimum.

---

2 This creates a disruptive landscape for the objective function $G(\bar{x})$. If one instead had used the max over all $g_k(\bar{x})$ then the objective function $G(\bar{x})$ would be continuous.
3 Experimental Results

To test the usefulness of the TCG-2 a simple steady-state evolutionary algorithm was developed. We used a constant population size of 100 and each individual was a vector $\mathbf{x}$ of $n$ floating-point components. Parent selection was performed by a standard binary tournament selection. An offspring replaces the worse individual of a binary tournament. One of three operators was used in every generation (the selection of an operator was done accordingly to constant probabilities 0.5, 0.15 and 0.35 respectively):

- **Gaussian mutation:** $\mathbf{x} := \mathbf{x} + N(0, \sigma)$, where $\sigma = (\frac{1}{2\sqrt{n}}, \ldots, \frac{1}{2\sqrt{n}})$ for all experiments reported in this section.
- **Uniform crossover:** $z := (z_1, \ldots, z_n)$, where each $z_i$ is either $x_i$ or $y_i$ (with equal probability), where $\mathbf{x}$ and $\mathbf{y}$ are two selected parents and $\mathbf{z}$ is the child.
- **Heuristic crossover:** $z' := r * (\mathbf{x} - \mathbf{y}) + \mathbf{x}$, where $r$ is a uniform random number between 0 and 0.25, and the parent $\mathbf{x}$ is not worse than $\mathbf{y}$ ($\mathbf{z}$ is the child).

The termination condition was to quit the evolutionary loop if an improvement in the last $N = 10,000$ generations was smaller than a predefined $\epsilon = 0.001$. Now the TCG-2 can be used to investigate the merits of any constraint handling method, e.g. for the popular constraint-handling method based on static penalties.

The following results are averaged over 100 runs, and in each figure the continuous line shows the average fitness of the best final individual whereas the broken line shows how often the global optimum was reached. Since the TCG-2 has 9 parameters it is difficult to discuss their interactions in this short paper. Thus we have selected a single point from the TCG-2 parameter search space:

\[ n=30, m=3, \rho=0.1, c=0.5, a=1, p=30, \sigma=0.5, \alpha=0.1, d=0.05 \]

and have varied one parameter at the time. Note that the lowest peak has a height of 0.1 and that all peaks might be placed in the infeasible space (except of the global optimum).

Figure 5 shows TCG-2\( (n, m=3, \rho=0.1, c=0.5, a=1, p=30, \sigma=0.5, \alpha=0.1, d=0.05) \) where the dimensionality $n$ is varied between 1 and 110. As $n$ is increased a sudden decrease of the performance (continuous line) occurs around $n=70$. The probability for reaching the global optimum (broken line) drops significantly faster.

Figure 6 shows TCG-2\( (n=30, m=3, \rho, c=0.5, a=1, p=30, \sigma=0.5, \alpha=0.1, d=0.05) \) where the feasibility $\rho$ is varied between 0.0001 and 0.8. As $\rho$ is increased a seemingly strange effect becomes visible: The chance for finding the global optimum decreases as the feasibility $\rho$ increases. The reason for this is that the evolutionary algorithm becomes stuck in a feasible local optimum, since more and more peaks

---

3 See the previous section.

4 The best final solution is considered to have "reached" the global optimum if the fitness is larger that 0.99. (The fitness at the global optimum is always 1.0.)
happen to be placed in (or nearby) a feasible area. The overall effect of a varying $p$ is smaller than for a varying dimensionality $n$. 

Figure 5: Varying dimensionality $n$

Figure 6: Varying feasibility $p$

Figure 7 shows $TCG-2(n=30, m=3, p=0.1, c, a=1, p=30, \sigma=0.5, \alpha=0.1, d=0.05)$ where the complexity $c$ is varied between 0 and 0.8. As $c$ is increased both the performance and the chance for reaching the global optimum decrease, but the overall impact is relatively small and comparable to the impact of varying the feasibility $p$. 

Figure 8 shows $TCG-2(n=30, m=3, p=0.1, c=0.5, a, p=30, \sigma=0.5, \alpha=0.1, d=0.05)$ where the number of active constraints $a$ is varied between 1 and 30 (note that $a \leq n$ and $n = 30$). As $a$ is increased, the performance decreases slightly and the chance of reaching the global optimum drops to zero around $a = 10$. Hence, if $a$ is increased the evolutionary algorithm has greater difficulties in reaching the global optimum but it finds other peaks with relatively good performance.

Figure 7: Varying complexity $c$

Figure 8: Varying no. active constraints $a$

Figure 9 shows $TCG-2(n=30, m=3, p=0.1, c=0.5, a=1, p, \sigma=0.5, \alpha=0.1, d=0.05)$ where the number of peaks $p$ is varied between 1 and 100. The chance of reaching the global optimum decreases as $p$ increases. The performance decreases until $p$ reaches 25, then the performance increases and stabilises. The reasons for this behaviour are based on two situations:

- $p$ is small (in this case $p \leq 25$): When $p$ is increased, the evolutionary algorithm more easily becomes stuck in a poor local optimum.
- $p$ is large (in this case $p \geq 30$): When $p$ is increased, more relatively good peaks are located in or nearby the feasible areas, and hence the evolutionary algorithm
more frequently finds a better solution. Further, as \( p \) increases there are also more relatively good local peaks based on the linear decay of the peaks.

Figure 10 shows \( TCG-2( n=30, m=3, \rho=0.1, c=0.5, a=1, p=30, \sigma=0.1, d=0.05 ) \) where the width of the peaks \( \sigma \) is varied between 0.01 and 15. As one would expect, increasing \( \sigma \), when \( \sigma \) is relatively small, increases the performance and the chance of reaching the global optimum. Nevertheless, increasing \( \sigma \) even further continues to increase the performance and the chance of reaching the global optimum. This happens since individuals that get into the feasible area around the global peak are assigned an objective value close to 1.0, which is larger than any other peak. On the other hand, when \( \sigma \) is relatively small the individuals that are close to local optima can receive a better fitness than the individuals close to the global optimum.

An overall observation is that the evolutionary algorithm always found a feasible solution (even though the fitness might have been very close to zero). This will most probably not hold for very large values of the dimensionality \( n \) and extremely small values of the feasibility \( p \).

4 Conclusions

The new test-case generator \( TCG-2 \) was presented, and it was explained how the \( TCG-2 \) overcomes the limitations of earlier test-case generators. The new \( TCG-2 \) creates nonlinear constrained optimisation problems by means of easy adjustable parameters. Using these parameters is was explained how the \( TCG-2 \) creates and controls the dimensionality, the multimodality, the feasibility, the ruggedness, the constraintness, the feasible components, the complexity of the search space and the separation of feasible islands. With the gradual and intuitive control over these parameters the \( TCG-2 \) is a significant improvement over earlier test-case generators.

It was shown that the largest impact on the performance and the chance for reaching the global optimum was found by varying the following parameters (in this order): The dimensionality \( n \), the width of the peaks \( \sigma \) and the number of peaks \( p \).

In general, varying the feasibility \( p \), the complexity \( c \) or the number of active constraints \( a \) seemed to have relatively little effect on the performance and the chance
for reaching the global optimum. Only very small values of $p$ had a significant impact, resulting in a low performance and a low chance of reaching the global optimum. The complexity $c$ has little importance and the number of active constraints $a$ has only importance for reaching the global optimum. Three TCG-2 parameters remain to be tested: The number of feasible components $m$, the decay of the peaks $a$ and the minimal distance $d$. Furthermore, parameter interactions will be investigated.

The very strong importance of the dimensionality $n$ shows that applications that use an evolutionary algorithm with a static penalty approach should test the performance of the algorithm over a wide variety of dimensions (especially large number of dimensions). This must be done in order to assure a robust performance of this approach – other approaches might have the same “problem”.

Future experiments will involve the comparison of various penalty methods, different repair algorithms, decoders, etc. Once the knowledge about the performance of various methods is collected one can create new techniques, test new ideas and compare the results with the knowledge of the performance of other methods. This will greatly enhance the possibility to easily compare different methods with each other.

Further, a future step for improving the usability of the TCG-2 is to implement a mapping from real-world problems to TCG-2 parameter settings. Once this mapping is implemented, the knowledge about which optimiser is best for a given TCG-2 parameter setting can be used to suggest the appropriate optimiser for a given real-world problem.

Acknowledgements
The research reported in this paper was partially supported by the ESPRIT Project 20288 Cooperation Research in Information Technology (CRIT-2): “Evolutionary Real-time Optimization System for Ecological Power Control”.

References
Solving CSP Instances Beyond the Phase Transition Using Stochastic Search Algorithms

Luk Schoofs, Bart Naudts

Intelligent Systems Lab,
Department of Mathematics and Computer Science,
University of Antwerp, RUCA,
Groenenborgerlaan 171, B-2020 Antwerpen.
{lshools, bnaudts}@ruca.ua.ac.be

Abstract

When solving constraint satisfaction problems (CSPs) with stochastic search algorithms (SSAs) using the standard penalty function, it is not possible to show that there is no solution for a problem instance. In this paper we present a hybrid function that can be generally used in conjunction with SSAs to prove unsolvability without changing the search algorithms drastically. We use eight state-of-the-art algorithms to show the general usability of the new function. We compare the algorithms with and without the new penalty function and we test the scalability of the new function.

1 Introduction

Stochastic search algorithms (SSAs) are good heuristics for the NP-complete constraint satisfaction problems (CSP, e.g., [15], [11], [7], [10], [8]). The major drawback of using SSAs in conjunction with the standard penalty function is that they cannot prove that an instance has no solution. It is this drawback that we want to solve in the most general way. The new method has to be elegant and easily integratable into existing SSAs without changing the overall setting of the algorithm. We test it on state-of-the-art algorithms, like for instance different kinds of GAs [6], PBIL [1] and UMDA [9]. Although an earlier attempt has been done by Dozier in [2], only little work has been performed in this area of proving unsolvability with SSAs.

The structure of the paper is as follows. First we define constraint satisfaction problems, with the standard penalty function, and the concept of path consistency. The new hybrid penalty function is discussed in Section 3 and the empirical results can be found in Section 4. We conclude this paper with an overall conclusion in Section 5.

2 Constraint satisfaction problems

A binary constraint satisfaction problem consists of the following components:

- a set of variables $x_0, x_1, ..., x_{n-1}$;
- a domain $D_i$ corresponding to each variable;
- a subset $C_{ij} \subset D_i \times D_j$ for each pair of variables $(i, j)$, with $0 \leq i < j < n$, which represents a constraint when it differs from the Cartesian product $D_i \times D_j$.

It can be shown that every CSP can be transformed into an equivalent binary CSP [15]. The goal of a CSP is to assign to the variables $x_i$ a value from their domain $D_i$ in such a way that all constraints are satisfied. Formally, we say that a constraint $C_{ij}$ is satisfied if and only if $(x_i, x_j) \in C_{ij}$. The couple $(x_i, x_j)$ is then called a valid assignment. When $(x_i, x_j) \notin C_{ij}$ we say that the assignment $(x_i, x_j)$ violates the constraint $C_{ij}$.

Besides the number of variables and the number of elements in the domains $D_i$, there are two other parameters that have to be set to generate a random CSP instance. The first one is the connectivity: the probability that a constraint is created between two variables. The second parameter is the probability that a couple $(i, j)$ is removed from the set $D_i \times D_j$. This is called the tightness of the constraints in the CSP. A phase transition from solvable to unsolvable is observed when the tightness goes from 0 to 1, at a value depending on the connectivity, the number of variables and their domain sizes. The mushy region is the area around the phase transition where solvable and unsolvable CSP instances coexist in significant proportions.

With every CSP instance, a constraint graph $G = (V, E)$ can be associated in the following way:

- every variable $i$ is a node in $V$,
- every constraint between the variables $i$ and $j$ is an edge in $E$ between the nodes in $V$ that correspond to the variables $i$ and $j$.

The constraint graph of a CSP can be directed or undirected, in particular the couples in $E$ can be ordered or unordered.

2.1 The standard penalty function

The difficulty with using black-box algorithms like evolutionary algorithms for finding solutions of a CSP is that the problem has no explicit objective function to be optimized. It is clear that one has to be constructed. The most obvious choice is the standard penalty function, which simply counts the number of constraints violated by one assignment.

An assignment $x$ is a solution of the CSP if and only if the penalty function becomes zero. It is clear that assignments with an equal number of constraint violations share the same fitness value, and that no distinction is made between the constraints. In particular, no distinction is made between what one intuitively experiences as easy and hard constraints.

2.2 Path consistency

To make the search space of the search problem smaller, we can eliminate the values from the domains which will certainly not lead to a solution. To do this we need the notion of path consistency [15].
Suppose $P = \{v_0, \ldots, v_{n-1}\}$ is a path in constraint graph $G = (V, E)$, with $v_i \in V$. If for every edge $(v_i, v_j) \in P$ the following two conditions hold

1. we can assign a value to the corresponding variables $x_i, x_j$ from their respective domains $D_i, D_j$ in such a way that no constraint $C_{ij}$ between the two variables is violated, and
2. we can assign to the corresponding variable $x_k$ of every vertex $v_k \in V \setminus \{v_i, v_j\}$ a value from its domain $D_k$ in such a way that no constraint between every two adjacent variables on the path $P$ is violated,

then the path $P$ is said to be path consistent.

A CSP is said to be path consistent if and only if every path in the corresponding constraint graph $G$ is path consistent. Intuitively, path consistency means that every value in the domains can lead to a solution of the CSP. In general, randomly generated instances are not path consistent.

Path consistency can be enforced by eliminating values from the domains that do not follow the conditions stated above. When during the elimination phase a domain becomes empty, the CSP instance has no solution, because an assignment to that variable is impossible. In [15] one finds deterministic algorithms to make CSP instances path consistent.

3 The hybrid penalty function

The new penalty function we present is a hybrid between the standard penalty function and a path consistency algorithm. The SSA in conjunction with the hybrid penalty function stops its search when a domain becomes empty. In this way it is used to efficiently prove that there is no solution for typical instances beyond the mushy region. As the empirical results show, the path consistency algorithm is not able to decide solvability in the mushy region.

Decreasing the domain size of a variable makes the search space smaller. When the function detects that a certain value, for a certain position, cannot lead to a solution it is eliminated and the SSA does not have to search that region of the search space anymore.

3.1 The algorithm

During the evaluation of a candidate solution, couples $(x_i, x_j)$ are eliminated from the set $P_{ij} = D_i \times D_j$ when a conflict between the variables $i$ and $j$ with these values is detected. If the couple was already eliminated in a previous cycle, the elimination has no effect and the evaluation continues. At the same time the penalty is increased with the appropriate value.

When a value $x_i$ for a variable $i$ has no more couples in $P_{ij}$, the value can be eliminated from the domain $D_i$. Because $x_i$ is eliminated from the domain it cannot be used in any other combination containing variable $i$ and all couples with $x_i$ are eliminated from every other set $P_{kl}$, $0 \leq k, l < n$. 
When $x_i$ is eliminated from $D_i$, all couples containing value $x_i$ are eliminated from the set $P_{ij}$. When this is done, it is possible that value $x_j$ is no more represented in $P_{jk}$ (for $k \neq i$). If this is the case, the value is also eliminated from $D_j$. This recursive elimination is done until no values can be deleted anymore. When a domain turns out to be empty, the evaluation is stopped and a "no solution" token is returned to the SSA. After processing a variable of the candidate solution, the evaluation continues with the next variable until all are checked. When this is done and no empty sets are detected, the function returns the fitness value for this candidate solution and the SSA can continue its search. The hybrid penalty function is detailed in Figure 1.

```plaintext
eliminate_values(i, j, x_i, x_j)
    eliminate the tuple $(x_i, x_j)$ from $P_{ij}$ (if possible)
    if $\exists$ empty domain
        return 'no solution'
    if no tuples $(x_i, *)$ in $P_{ij}$
        then
            eliminate $x_i$ from $D_i$
            $\forall k$ such that $(x_i, x_k) \in C_{ik}$
                eliminate_values(i, k, x_i, x_k)
            $\forall k$ such that $(x_k, x_i) \in C_{ki}$
                eliminate_values(k, i, x_k, x_i)
        if no tuples $(*, x_j)$ in $P_{ij}$
            then
                eliminate $x_j$ from $D_j$
                $\forall k$ such that $(x_j, x_k) \in C_{jk}$
                    eliminate_values(j, k, x_j, x_k)
                $\forall k$ such that $(x_k, x_j) \in C_{kj}$
                    eliminate_values(k, j, x_k, x_j)

penalty_function(X = \{x_0, x_1, ..., x_{n-1}\})
    for every variable $i$
        for every variable $j$ other than $i$
            if conflict between $i = x_i$ and $j = x_j$
                then
                    if eliminate_values(i, j, x_i, x_j)
                        = 'no solution'
                        then
                            return 'no solution'
                    increment the penalty $pen$
                return the penalty value $pen$
```

Figure 1: The hybrid penalty function in pseudo code. It is composed of two parts: the first part implements the path consistency algorithm. The second part is the main body of the penalty function as described in Section 3.1. The expression $(x_i, \ast)$ is the set of all couple with as first variable $i$ with the value $x_i$. The value in the second position does not matter. We assume that the sets $P_{ij}$ are already created.
3.2 Remarks

The SSA works with the univariate information of the hybrid penalty function to generate new candidate solutions. The interaction information, however, is not used. This means that bad combinations of values can be used several times during the search. *Tabu search* [4] can prevent this, but experiments along this line have failed up to now.

The algorithms that use the hybrid penalty function have to generate a high diversity of values for the path consistency algorithm to be efficient. This is not the case for a few algorithms like UMDA, as we will see later on.

We want to reduce the search space as quickly as possible. Therefore the hybrid penalty function eliminates as many values as possible from as many domains as possible in one evaluation of a candidate solution. This is why most of the evaluation time is spent at making an instance path consistent.

The effect of eliminating an element from a domain is only noticeable when new values are introduced. GA-like algorithms only generate new values through the mutation operator. Algorithms like MIMIC, PBIL, ANT, etc. construct completely new populations and for this reason introduce a lot more new values than GA-like algorithms.

4 Experimental results

To study the effectiveness of the new penalty function, we applied it to several SSAs. This also shows that the hybrid penalty function can be generally used with a great variety of algorithms.

- random hill climbing, HC
- population-based incremental learning, PBIL [1]
- mutual-information-maximizing input clustering, MIMIC [3]
- simple genetic algorithm, SGA [6], [5]
- genetic algorithm with bit-based simulated crossover, GA-BSC [14]
- adaptive genetic algorithm, AGA [12]
- univariate marginal distribution algorithm, UMDA [9]
- ant colony algorithms, ANT [13]

The experiments were set up to show the difference between the search with the standard penalty function and with the hybrid penalty function. The algorithms are run with optimal values for their parameters. These optimal values are found experimentally, as one can see in [13]. The runs are first done for small problem instances: 10 variables, 4 elements in the domains, a connectivity of 1.0. The tightness is changed between 0 and 1 with 0.05 as step size. This can be summarized as (10, 4, 1.0). As a second experiment we use larger problem
instances to show that the hybrid penalty function is also effective for larger problems. The larger CSP model is presented as \((30, 5, 0.5)\).

In what follows the figures are always shown in pairs, for the same algorithm, so one can clearly see what happens when going from a smaller to a larger problem. A figure always shows the median number of evaluations over 30 runs for every tightness value. A distinction is made between the runs with the standard penalty function (lines with the stars) and those with the hybrid penalty function (lines with the boxes). Because of space limitations only the most relevant figures are shown.

In Figure 2 we used a deterministic backjumper (cf. [15]) to solve random instances of the two previously described problem classes. With this deterministic algorithm we can determine the location of the mushy region. For the problem with 10 variables it is found between tightness values 0.18 and 0.31. In the larger problem this is between 0.12 and 0.18.

As we can see in Figures 3-7, the performance of the algorithms before the mushy region is not affected by the hybrid penalty function. It can be clearly seen that the two curves are located on top of each other. Also instances in the mushy region cannot be solved or proved unsolvable.

None of the algorithms was able to solve instances in the mushy region in an efficient way, as expected. They even perform badly near the mushy region. The peak around the mushy region reaches the maximum number of evaluations, which means that the algorithms cannot solve the instances in a proper way. This unsolvability can be seen for solvable instances before of the mushy region.
as well as for unsolvable instances beyond the mushy region. For instance the SGA (Figure 5) and the GA-BSC perform so badly that even for very small tightness values no solution can be found. This is due to the fact that a very large population size had to be used and as a result the maximum number of evaluations is reached in fewer generations.

![Figure 3: Two plots for a HC. The left plot describes the median number of evaluations needed to solve a (10, 4, 1.0) CSP class. On the right side the same is shown for a (30, 5, 0.5) CSP class. The x-axis presents the tightness and the y-axis gives the median number of evaluations in a logarithmic scale over 30 experiments. On both plots the curve with the stars represents the runs with the standard penalty function and the lines with the boxes indicate the runs with the hybrid penalty function.](image)

![Figure 4: The plots for PBIL. Same explanation as for Figure 3](image)
Figure 5: The plots for SGA. Same explanation as for Figure 3

Figure 6: The plots for AGA. Same explanation as for Figure 3

Figure 7: The plots for ANT. Same explanation as for Figure 3
Even though the same unsolvability checker is applied, the area around the mushy region is not the same for every algorithm. This can be seen on Figures 5 and 6 where the bump for SGA is larger than the one for AGA. It can also be seen that beyond the phase transition all algorithms find proof that there is no solution on the same tightness value. The bump is larger before the phase transition where all instances are solvable. As a result, the shape around the mushy region is determined by the algorithm itself and not by the path consistency algorithm.

In the case of the deterministic algorithm used to locate the mushy region, we observe that beyond the mushy region the median number of evaluations is decreasing slowly. All SSAs using the standard penalty function cannot prove there is no solution. They keep searching for a solution forever (which means 100,000 evaluations). Once the curve reaches the maximum, it never decreases again. When the SSAs use the hybrid penalty function, the curve suddenly falls down, meaning that is very easy to prove there is no solution. It is obvious to see that this first population carries enough information to prove there is no solution. This drop is for every algorithm very sudden and very steep, but it does not coincide with the edge of the mushy region. We notice that in the region where the deterministic algorithm needs a lot of evaluations to prove there is no solution, the SSAs cannot find the proof.

5 Conclusions

In this paper we presented a hybrid method for determining the unsolvability of CSP instances. The method is based on the standard penalty function which is hybridized with a path consistency algorithm. The new penalty function can easily be added to every SSA solving CSPs without influencing the whole setting of the algorithm. The new hybrid penalty function is not usable for instances in the mushy region itself and on average it does not affect the performance of the algorithms on instances before the mushy region.

Acknowledgments

The second author is a post-doctoral researcher of the Fund for Scientific Research – Flanders (Belgium) (F.W.O.).

References


Noisy and Non-stationary Environments
Abstract. Recently, there has been an increasing interest in applying evolutionary computation to path planning [15]. To date, these evolutionary path planners have been single agent planners. In real-world environments where the knowledge of obstacles is naturally distributed, it is possible for single agent path planners to become overwhelmed by the volume of information needed to be processed in order to develop accurate paths quickly in non-stationary environments. In this paper, a new adaptive replacement strategy (ARS) is presented that allows steady-state evolutionary path planners to search efficiently in non-stationary environments. We compare this new ARS with another ARS using a test suite of 5 non-stationary path planning problems. Both of replacement strategies compared in this paper work by allowing an influx of diversity rather than increasing mutation rates. We refer to this influx of diversity as hyper-diversity.

1 Introduction

The Distributed Path Planning Problem can be stated as follows. Given an environment $(R, B, G, A, O)$ where $R$ represents a “client” point agent, $B$ represents the starting (or beginning) point, $G$ represents the destination point, and $A$ represents a set of distributed “planning” agents, where each $a_i \in A$ has knowledge of a subset of obstacles, in $O$, located near the region that $a_i$ occupies: discover a collision free path from $B$ to $G$ using the “planning” agents in $A$ that some “client” agent, $R$, can traverse.

In this paper, we compare a number of distributed neuro-evolutionary computations (DNECs) for path planning in non-stationary environments. The DNECs evolve candidate paths (CPs) that are represented as radial basis function networks (RBFNs) [8]. By evolving, RBFNs the DNECs are able to dramatically reduce the number of nodes (or via points) needed to represent smooth candidate paths. The major contribution of this paper is the comparison of two adaptive replacement strategies (ARS) that are based on hyper-diversity.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of distributed EC, RBFNs, and steady-state ECs for non-stationary problems. In Section 3, the DNECs and the ARSs are described in detail. In Section 4, the test suite of five randomly generated path planning problems
is described and in Section 5, our preliminary results are presented. Section 6 provides a brief discussion and our conclusions are presented in Section 7.

2 Background

2.1 Distributed ECs

Distributed Evolutionary Computations (DECs) typically fall into one of three categories: function-based [1], domain-based [11], and variable-based [10]. Function-based DECs distribute tasks (or functions) of the evolutionary process (selection, procreation, evaluation) among \( k \) processors in order to speed-up the processing time of a single EC.

Domain-based DECs (DBDECs) distribute a population of \( P \) candidate solutions (CSs) among \( k \) processors. There are two types of DBDECs: coarse-grained and fine-grained. In coarse-grained DBDECs, \( k \) populations of \( \frac{P}{k} \) CSs are maintained. Thus, \( k \) ECs are executed in parallel. Periodically, selected CSs are allowed to migrate to other populations. Fine-grained DBDECs usually assign one CS to each processor. The populations or demes overlap as CSs are only allowed to mate and compete for survival within their geographical neighborhoods.

In variable-based DECs (VBDECs), the structure representing CSs is distributed. Let \( V \) represent the variables that form the structure of the CSs of a problem. VBDECs, distribute \( \frac{V}{k} \) variables among \( k \) processors. Each processor uses an EC to evolve a population of \( P \) partial CSs.

2.2 Steady-State Search in Non-Stationary Environments

Research in the area of evolutionary search in non-stationary environments to date has focused primarily on two approaches: memory-based [6] and mutation-based [2, 3, 7, 11, 13]. Memory-based approaches enhance ECs with additional structures that enable them to remember past solutions while mutation-based approaches focus on using "higher than normal" mutation rates in an effort to track non-stationary optima. In this paper we investigate a third approach that can be used by steady-state evolutionary search in non-stationary environments.

In [14, 12], the authors introduce a simple modification to steady-state search that allows it to effectively track optima in non-stationary environments. Rather than replacing the worst individual each iteration they replaced the oldest individual.

Adaptive replacement [4] is a diversity-based strategy that is a generalization of the "replace the oldest" strategy. This type of replacement strategy operates by allowing an influx of diverse individuals into the population for a specified amount of time. This influx of diversity is referred to as hyper-diversity.

To better illustrate the concept and sufficient conditions for achieving hyper-diversity, let \( \rho_s \) represent the selection pressure as the ratio of the average fitness of the population to the fitness of an individual that is selected to be a parent\(^1\). Similarly, let \( \rho_r \) represent the replacement (deletion) pressure as the ratio of the

\(^1\) In this paper we consider minimization only.
average fitness of the population to the fitness of an individual selected to be replaced.

Given \( \rho_s \) and \( \rho_r \), hyper-diversity results when \( \rho_s = \rho_r \). As long as \( \rho_s > \rho_r \), the population will converge towards a desired solution and when \( \rho_s < \rho_r \) the population will experience 'reverse' evolution and diverge.

In adaptive replacement, the parameter that is adapted during search is, \( \gamma \), the number of fitness-based replacements before one age-based replacement. Thus, the "replace the oldest" strategy can be seen as \( \gamma = 0 \). The value assigned to \( \gamma \) is based on the amount "stress"[2] the EC is under as the environment changes. Prior research has shown that adaptive replacement is superior to static replacement strategies[4].

2.3 RBFNs and Path Construction

The construction of CPs can be viewed as an interpolation problem where the objective is to develop a continuous function that passes through \( n \) specified nodes. Consider a sequence of \( n \) nodes, \([(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]\). An interpolation RBFN representing this sequence can be expressed as \( f(x) = \sum_{i=1}^{n} \phi(x, x_i)w_i \), where \( \phi(x, x_i) = \exp\left(\frac{(x-x_i)^2}{2\sigma^2}\right) \) and where \( w_i \) are weights. In order to properly pass through the nodes, a set of values must to be discovered for the weights. This can be accomplished by solving the following linear equations for the set of weights (where \( \sigma = \frac{1}{\sqrt{2n}} \), and where \( \phi \) must be non-singular) [8]:

\[
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n 
\end{bmatrix} =
\begin{bmatrix}
\phi(x_1, x_1) & \phi(x_1, x_2) & \cdots & \phi(x_1, x_n) \\
\phi(x_2, x_1) & \phi(x_2, x_2) & \cdots & \phi(x_2, x_n) \\
\vdots & \vdots & \ddots & \vdots \\
\phi(x_n, x_1) & \phi(x_n, x_2) & \cdots & \phi(x_n, x_n)
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
\vdots \\
w_n
\end{bmatrix}
\]

3 The DNECs

3.1 An Evolutionary Agent Protocol

The DNECs were developed for evolving CPs through an environment composed of one-dimensional, non-stationary obstacles. The obstacles are represented by a vertical line segment with an upper and lower bound. Associated with each obstacle in the environment is an evolutionary agent located at the midpoint of the obstacle. Each agent has an \( x \) coordinate that remains constant and uses an EC to evolve a population of offsets that are added to the upper or lower bounds of their associated obstacle to form feasible \( y \) coordinates. Thus, each offset of an EC represents one node of a distributed CP. The values on the \( x \) and \( y \) axes range from 0.0 to 1.0.

The protocol used by an evolutionary agent is as follows. Each agent uses two random number generators \( rnd_1 \) and \( rnd_2 \). The random number generator, \( rnd_1 \), is used for synchronizing the tournament selection algorithm [9]. If the seed for each evolutionary agent is identical then the selection of parents will be synchronized (i.e., all the offsets corresponding to a distributed parent will be selected by the evolutionary agents). If the seeds of the agents are different then an asynchronous form of tournament selection will be used (i.e., each agent may
select offsets corresponding to different distributed CPs to be crossed and/or mutated).

After the seed has been given to an evolutionary agent, the protocol proceeds as follows. An initial population of \( P \) offsets is randomly generated\(^2\). Each offset is then evaluated by sending it to the 'Evaluate' procedure which takes as arguments the address of some agent that will be evaluating the length of the distributed CP, the population of offsets, the index of the offset to be evaluated, and the time of birth of the offset (tob). This procedure converts the offset into a \( y \) value and then sends a message to the evaluator agent containing the candidate node and the index of the distributed CP that the node belongs to. The evaluator agent replies with a message which contains the index of the distributed CP as well as the overall path length associated with the CP. This path length is assigned to the offset as its fitness\(^3\). After a CP is evaluated, the position of the evolutionary agent (along with the position of its associated obstacle) is changed.

After the initial generation of offsets has been generated and evaluated, the evolutionary agent begins its iterative loop of selection, procreation and replacement. In this loop, the timer \( t \) is incremented and two parents are selected via (synchronous or asynchronous) tournament selection.

After two parent offsets have been selected, one offset in the population must be selected to be replaced. This is an important part of the protocol. One cannot focus on selecting the worst fit individual [14] which we refer to as fitness-based replacement. Instead a balance must be negotiated between fitness-based replacement and age-based replacement. In age-based replacement, the oldest individual is selected to to be replaced. We define the fitness-based replacement rate, \( \gamma \), to be the number of consecutive fitness-based replacements executed before a single age-based replacement is executed\(^4\).

Once an offset is selected to be replaced, the parent offsets then create a child offset, \( \delta_{ic} \), through the use of three operators. The first operator is \( BLX-0.0 \) crossover [5] and works as follows: \( \delta_{ic} = \text{rnd2}(\delta_{ip}, \delta_{iq}) \), where \( \text{rnd2} \) returns a uniform random number within the interval \([\delta_{ip}, \delta_{iq}]\). The second and third operators use only the first parent selected and are as follows: \( \delta_{ic} = \text{rnd2}(\delta_{ip}, 0.0) \) and \( \delta_{ic} = \delta_{ip} + \text{rnd2}(\delta_{ip}, 0.0) \). Each offspring undergoes a Gaussian disturbance, \( N(0, s) \). The child offset is then mutated using sign mutation which flips the sign of the offset. After sign mutation, the child replaces the offset selected to die and is evaluated. After each offspring is evaluated the coordinates of the evolutionary agent are changed and the population of offsets is updated so that the distributed paths will remain stationary.

### 3.2 Two Adaptive Replacement Strategies

It has been shown that adapting \( \gamma \) during evolution results in greater search efficiency [4]. The reasons for this stem from the fact that the EC's perception

\(^2\) Offset are always checked to make sure that their associated \( y \) value is within \([0.0,1.0]\).

\(^3\) Since the associated path length of an offset is assigned as its fitness, the lower the fitness the better the offset.

\(^4\) In [14], the authors only investigate the use of \( \gamma = 0 \).
of environmental change varies during evolution. If the environment remains relatively stationary, then a large $\gamma$ would be more effective than a smaller one. However, as the rate of change of the environment increases the $\gamma$ value should decrease to prevent the selection algorithm from selecting individuals based on old, out-dated fitnesses.

The two adaptive replacement strategies (ARSs) attempt to balance the convergence of the population with the (perceived) rate of change of the environment. This ratio can then be used as the $\gamma$ value for a specified amount of time. These two strategies differ in the method used for calculating population convergence. The first ARS, ARS-1 [4], considers only the best individual of the population. The second ARS, ARS-2, takes into consideration the entire population.

ARS-1 can be derived using the following notation. Let $x_k$, denote the best individual in the population at time $k$, $\varepsilon(x_k, m)$, denote the fitness associated with $x_k$ at time $m$, where $m \geq k$, and $\frac{df_p}{dt}$, denote the change in the best fitness of the population with respect to time $\frac{\varepsilon(x_k, k) - \varepsilon(x_k, k+\Delta t)}{\Delta t}$. Let $\frac{df_b}{dt}$, denote the change in of the previous best individual’s fitness over time $\frac{\varepsilon(x_{k-1}, k-\Delta t) - \varepsilon(x_{k-1}, k)}{\Delta t}$, and $\frac{df_p}{df_b}$, denote the ratio of the population convergence (with respect to the best fitness in the population) to the perceived change in the environment. Therefore, ARS-1 is as follows:

$$
\gamma = \begin{cases} 
\Delta t & \text{if } df_b \geq 0 \\
\frac{|df_p|}{df_b} & \text{if } df_p \geq 0 \text{ and } df_b < 0 \\
0 & \text{otherwise}
\end{cases} 
$$

In Equation 1, three triggers [2, 3] are shown. The first trigger causes the EC to use fitness-based replacement exclusively. Any time the DNEC using ARS-1 perceives that the environment is stationary or that the previous best individual’s path length decreases as the environment changes then $\gamma = \Delta t$.

The second trigger allows $\gamma$ to be assigned intermediate values within the interval $[0, \Delta t]$. As long as the rate of population convergence is greater than the perceived rate of change of the environment then a non-zero value will be assigned to $\gamma$. If the case arises where $\frac{df_p}{df_b} \geq \Delta t$ then $\gamma$ is set equal to $\Delta t$.

The third trigger is used to invoke the “replace the oldest” strategy. Our hypothesis is that when this strategy (as well as ARS-2) is used with smaller population sizes (< 100) hyper-diversity will result. Finally, notice that the best individual is re-evaluated every $\Delta t$ iterations. Therefore, ARS-1 (and ARS-2) is said to be a 'conservative' replacement strategy [12].

ARS-2, is similar to ARS-1. The only difference between ARS-1 and ARS-2 is that the population convergence is measured relative to the average fitness of the population rather than the best individual. Thus, the three triggers are also similar.

---

5 In this paper one time unit is equivalent to one CP evaluation or one iteration of the distributed steady-state EC.

6 This can happen when by updating an offset the value of the offset goes from non-negative to negative or vice versa.
4 The Test Suite

Our test suite, shown in Figure 1, consisted of five randomly generated non-stationary path planning problems. For each problem, the number of equally spaced obstacles was randomly selected from the interval, [4,8]. Problems 1 and 5 consisted of 4 obstacles while Problems 2-4 consisted of 8 obstacles. The length of each obstacle was randomly selected from the interval [0.15,0.35]. For each of the five problems the start and destination points were (0.0,0.5) and (1.0,0.5).

Each obstacle, \( o \in O \), slides parallel to the y-axis. The obstacles move at a constant rate with the y-coordinate of each obstacle changing every time a distributed CP is evaluated. The rate at which an obstacle moves is determined by \( \Delta_o = \frac{2(1-\omega_o)}{g-2} \), where \( \omega_o \) represents the length of \( o \) and \( g \) represents the maximum number of path evaluations allowed. The obstacles initially move upwards until they reach the edge of the top boundary of the environment and then move downward until the reach they bottom boundary of the environment. At this point they being moving upwards once more. After \( g \) path evaluations all obstacles will have returned to their original starting positions.

5 Results

A total of 12 DNECs were compared. These DNECs were distinguished by two attributes: type of selection and replacement method, which was taken from the set \( \{a,A,s,S\} \) (where \( a \) denotes asynchronous selection with ARS-1, \( A \) denotes asynchronous selection with ARS-2, \( s \) denotes synchronous selection with ARS-1, and \( S \) denotes synchronous selection with ARS-2,) and population size, which was taken from the set \( \{5,10,20\} \).

The other parameter settings for the 12 DNECs were as follows. The usage rates for the three operators were 0.5, 0.25, and 0.25. The standard deviation of the Gaussian disturbance of the DNECs was set to 0.03 (\( s = 0.03 \)), and the sign mutation rate was set to 0.05 (\( \mu = 0.05 \)). The length, \( l \), of a CP was determined as follows: \( l = \Sigma_{i=0}^{19} dst(\frac{i}{20}, f(\frac{i}{20}), \frac{i+1}{20}, f(\frac{i+1}{20})) \), where \( dst(b,c,e,f) \) returns the
distance between points \((b, c)\) and \((e, f)\). Finally, \(\Delta t\) was set to 5 and at \(t = 0\) \(\gamma\) was set to 1.

In our study, the 12 DNECs were divided into classes based on the selection method and the population size used. Each of the DNECs was run a total of 31 times on each of the problems with \(g = 2000\). In order to determine the better performing ARS, a Student's t-test was used. The ARS that had the greater number of statistically significant mean off-line performances was considered to be the better overall performer. Any \(|t| \geq 1.7\) signaled a statistically significant difference in average performance.

In Table 1, six DNECs with synchronous selection are compared. In each cell of the table the mean off-line performance along with the standard deviation of the off-line performance is presented. Every third row contains the \(t\)-value of a \(t\)-test that was applied to the algorithms. In Table 1, when the population size of the synchronous DNECs is five, ARS-1 has statistically significant better performance than ARS-2 on the two easier problems in the test suite (Problems 1 and 5). This pattern can also be seen when the population is increased to 10 and 20. ARS-2 has the better performance on two of the three harder instances when the population size is 5 and 20.

Table 2 presents the results of the asynchronous DNECs. Once again the results show that ARS-1 performs better on Problems 1 and 5 (although the performance on Problem 1 is not statistically significant) while ARS-2 performs better on Problems 3 and 4 when the population size is 5 and 10. When the population size is 20, no statistically significant performances were found.

6 Discussion

In order to better understand the behavior of the ARSs we plotted \(\gamma\) along with \(q_k\). In Figures 2 and 3, \(q_k\) represents the mean difference in selection a replacement pressure for the \(k^{th}\) update of \(\gamma\). Let \(f_p(i)\) represent the average fitness of the population after the \(i^{th}\) iteration. Let \(\rho_s(i) = \frac{f_p(i)}{s(i)}\) and \(\rho_r(i) = \frac{f_p(i)}{w(i)}\) where \(s(i)\) represents the fitness of the first parent selected in iteration \(i\).
Table 2. The Performances of the Asynchronous DNECs

<table>
<thead>
<tr>
<th>Algs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.129 (0.017)</td>
<td>1.510 (0.066)</td>
<td>1.582 (0.060)</td>
<td>1.541 (0.054)</td>
<td>1.180 (0.018)</td>
</tr>
<tr>
<td>(a,5)</td>
<td>(A,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.136 (0.018)</td>
<td>1.535 (0.052)</td>
<td>1.577 (0.064)</td>
<td>1.493 (0.052)</td>
<td>1.193 (0.017)</td>
</tr>
<tr>
<td></td>
<td>t = -1.62</td>
<td>t = -1.64</td>
<td>t = 0.29</td>
<td>t = 3.70</td>
<td>t = -2.95</td>
</tr>
<tr>
<td></td>
<td>(a,10)</td>
<td>(A,10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.142 (0.013)</td>
<td>1.541 (0.056)</td>
<td>1.584 (0.051)</td>
<td>1.483 (0.043)</td>
<td>1.201 (0.018)</td>
</tr>
<tr>
<td></td>
<td>1.156 (0.016)</td>
<td>1.520 (0.056)</td>
<td>1.557 (0.062)</td>
<td>1.459 (0.046)</td>
<td>1.211 (0.017)</td>
</tr>
<tr>
<td></td>
<td>t = -3.66</td>
<td>t = 1.38</td>
<td>t = 1.72</td>
<td>t = 2.60</td>
<td>t = -2.21</td>
</tr>
<tr>
<td></td>
<td>(a,20)</td>
<td>(A,20)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.175 (0.021)</td>
<td>1.515 (0.076)</td>
<td>1.560 (0.051)</td>
<td>1.471 (0.036)</td>
<td>1.241 (0.027)</td>
</tr>
<tr>
<td></td>
<td>1.177 (0.017)</td>
<td>1.510 (0.057)</td>
<td>1.546 (0.057)</td>
<td>1.463 (0.039)</td>
<td>1.247 (0.023)</td>
</tr>
<tr>
<td></td>
<td>t = -0.57</td>
<td>t = 0.26</td>
<td>t = 1.13</td>
<td>t = 0.91</td>
<td>t = -1.10</td>
</tr>
</tbody>
</table>

and where \( w(i) \) represents the fitness of the individual that is to be replace in iteration \( i \). Given the above definitions, \( q_k = \frac{1}{\Delta t} \sum_{i=k}^{k+\Delta t-1} \rho_s(i) - \rho_r(i) \), where \( k = 0, 5, 10, \ldots, g - \Delta t \). Therefore, when \( q_k > 0 \), the ARS causes the population to converge, when \( q_k = 0 \), the ARS causes hyper-diversity in the population, and when \( q_k < 0 \), the ARS causes the population to diverge.

Figure 2 shows plots of \( \gamma \) and \( q_k \) for one run of \((s, 20)\) on the first and second problems (we are not able to show all five plots due to space constraints). One will immediately notice the varying \( \gamma \) values throughout each run. When one views the plot of \( q_k \) closely, it can be seen that whenever \( \gamma = 5 \) the value of \( q_k \) is always positive. This shows that during these time periods that ARS-1 is causing the algorithm to converge. However, whenever \( \gamma = 0 \) the value of \( q_k \) will occasionally drop below zero. One can also see the change in \( q_k \) over these periods is more erratic.

The reason for this behavior (when \( \gamma = 0 \)) is because the ARS-1 only considers the convergence of the best individual. When \( \gamma = 0 \) and aged-based replacement is used exclusively, the fitnesses of the individuals are more evenly distributed with respect to age. This is primarily because the population may not have fully converged. Therefore above and below average individuals will be equally replaced with respect to age.

Figure 3 shows plots of \( \gamma \) and \( q_k \) for one run of \((S, 20)\) on the first and second problems. One observes immediately that ARS-2 causes \( \gamma = 5 \) for longer periods of time. This is because ARS-2 considers the average fitness of the population in determining population convergence.

Notice once again that when \( \gamma > 0 \) the value of \( q_k > 0 \). Now, notice that whenever \( \gamma = 0 \) the value of \( q_k < 0 \). This is not always the case but it does occur frequently. This means that ARS-2 reverts to ‘reverse’ evolution in an effort to add diversity to the population.

The behavior of ARS-2 can be explained (once again) by the fact that population convergence is determined by average fitness. After long periods where \( \gamma = 5 \), the fitnesses of the individuals will be much more closer than if only the fitness of the best individual were considered. When \( \gamma = 0 \), the individu-
als being replaced will have fitnesses that are closer to the average fitness of the population. Since, the selective pressure of the tournament selection algorithm is relatively low, aged-based replacement will remove good individuals much faster than the EC can replace them. Since the replacement pressure dominates the selective pressure, the net effect is 'reverse' evolution.

7 Conclusions
In this paper, we have presented and compared two adaptive replacement strategies that are triggered by the convergence of the EC as well as the perceived change in the environment. Our results show that ARS-1 performs better on easier problems while ARS-2 performs better on harder problems. In our discussion, we showed how ARS-2 alternates between convergence ($g_k > 0$) and 'reverse' evolution ($g_k < 0$). In the future, we plan to investigate possibility of balancing the selective and replacement pressure in ARS-2 when $\gamma = 0$.

References


Optimization of Noisy Fitness Functions by Means of Genetic Algorithms Using History of Search

Yasuhito SANO\(^1\) and Hajime KITA\(^2\)

\(^1\) Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, Yokohama 226-8502, JAPAN
\(^2\) National Institution for Academic Degrees, 3-29-1 Otsuka, Bunkyo, Tokyo 110-0012, JAPAN

Abstract. In the present paper, optimization of functions with uncertainty by means of Genetic Algorithms (GA) is discussed. For such problems, there have been proposed methods of sampling fitness function several times and taking average of them for evaluation of each individual. However, important applications having uncertain fitness functions are online adaptation of real systems and complicated computer simulation using random variables. In such applications, possible number of fitness evaluation is quite limited. Hence, methods achieving optimization with less number of fitness evaluation is needed. In the present paper, the authors propose a GA for optimization of continuous fitness functions with observation noise utilizing history of search so as to reduce number of fitness evaluation. In the proposed method, value of fitness function at a novel search point is estimated not only by the sampled fitness value at that point but also by utilizing the fitness values of individuals stored in the history of search. Computer experiments using quadric fitness functions show that the proposed method outperforms the conventional GA of sampling fitness values several times at each search point in noisy environment.

1 Introduction

Genetic Algorithms (GAs) are optimization technique widely applicable to various problems because of their characteristics such as optimization only using function values, global search ability and flexibility of the framework. Online adaptation through experiment-base evaluation and optimization based on complex simulation using random numbers are important applications of the GAs.

For more successful applications of GA to such problems, we must enhance search ability of GAs considering both the uncertainty of fitness values and limit of allowable number of fitness evaluation. In the present paper, the authors propose a novel GA achieving quicker and more precise convergence in optimization of noisy fitness function utilizing history of search so as to reduce the noise.
2 Optimization of Noisy Fitness Functions

2.1 Online Optimization

As purpose of this study, we discuss online optimization problems for real systems. In such problems, the following points must be considered: (1) uncertainty of environment, (2) limitation of the possible number of fitness evaluation and (3) limitation of evaluation of individuals that degrade the performance of the system extremely. One of the authors has considered the third requirement introducing a three layer controller called the Bio Control Architecture[1]. In this paper, we focus our study on the issues of (1) and (2).

2.2 Formulation of Problem

Real systems have some uncertainty, and performance of a system will fluctuate because of uncertainty. Such fluctuation will be categorized into the following three types:

1. Input to the system, or realization of the design may not be sufficiently precise. Hence the real performance of the system deviates from the ideal one.
2. Measurement of the performance of the system may not be accurate, and involves some observation noise.
3. Environment may change, and the characteristics of the system may deviate from those identified in advance due to the environmental change.

In the case of (1), input should be decided taking the unpredictable fluctuation into account, and therefore to find an input robust to such fluctuation is needed. Application of GAs to such problems is discussed by e.g. [3][8][7]. When the standard GA is applied to a system under the unreliable environment, it becomes a problem that a really excellent individual may not by taken over in the next generation because selection is carried out based on sample fitness values.

In case of (3), if environmental change is slow, adaptation to such environmental change will be an effective strategy. GAs to cope with environmental change are discussed by e.g. [9][10][11][12][13].

In this paper, the authors focus out discussion on the second type of uncertainty. In such case, optimization of the system must be carried out under randomly fluctuating fitness function, and hence, some techniques to cancel such fluctuation are needed. The problem in this category is formulated as follows:

\[
\min_{x \in \mathcal{X}} \langle F(x) \rangle \\
F(x) = f(x) + \delta \quad \delta \sim N(0, \sigma^2)
\]

where \(f\) and \(\mathcal{X}\) represent the fitness function and the search space, respectively, \(\langle \rangle\) represents expectation, and random fluctuation \(\delta\) is modeled as probabilistic
variable following a normal distribution. In this paper, we call \( f(x) \) 'the true fitness value', and \( F(x) \) 'the sampled fitness value'.

In aforesaid studies of the problem categorized into (1), robust solutions are obtained through optimization of expectation of fitness under additive noise to the decision variables. Hence, methods for the categories (1) and (2) are closely related.

2.3 Genetic Algorithms for Noisy Fitness Functions

Given sufficiently large population size, GAs find optimal solution of noisy fitness function because of self-averaging nature of the population based search[8]. However, in practical applications having noisy fitness functions, naive GAs face difficulty of slow convergence and poor solution quality. So as to enhance search ability of GAs, several methods have been proposed. They are categorized into the following two approaches.

The first one is to sample fitness values several times for each individual, and to use the mean of the sampled values for evaluation of the individual. Fitzpatrick et al. have proposed a method which samples fitness values more than once for each individual[2]. Branke has proposed a method that restricts multiple sampling only to the best or nearly best solutions so as to reduce fitness evaluations[3]. Furthermore, Stagge has used a technique in which sampling times is decided using t-test for significance of good solutions[4]. This approach achieves reduction of the variance of fitness values without any assumptions on fitness function. However, it requires large number of evaluations to reduce the variance because variance reduces in inverse proportion to sampling times. Considering practical applications such as optimization through experiment or simulation, it is a serious drawback in this approach.

Another approach is to evaluate an individual not only by its sampled fitness values but also by other individuals near it. This approach requires some assumptions on fitness functions so as to predict fitness value of a point using these sampled other points. Tamaki et al. have proposed a technique which refer to not only the sampled value of an individual but also the fitness values of its parent[6][7]. However, since the parent is a survivor in selection, its fitness values have some bias toward apparently better solution. Hence the estimated value involves systematic error. Branke also has proposed a method which refer to the sampled value of the nearby individual in current and previous generation[3]. In this method, the number of the samples used for estimation is limited in number.

3 Estimation of Fitness Value Using History of Search

In this paper, a genetic algorithm for noisy fitness function is proposed. The authors take the second approach in the previous section. That is, an individ-

---

1 Another formulation is multi-objective optimization that minimizes both the mean value and the variance of the fitness function simultaneously. A typical example requiring such formulation is optimization of investment so as to achieve 'high return' and 'low risk'.
ual is evaluated using the fitness values of the other individuals as well as the sampled fitness values for it. So as to increase number of individuals used for evaluation, we use not only the current generation but also whole the history of search. To utilize history of search, a stochastic model of the fitness function is introduced, and the maximum likelihood technique is used for estimation of the fitness functions.

3.1 A Stochastic Model of Fitness Functions

Suppose it requires to estimate the fitness of an individual using other individuals in the history of search. Since individuals are located different places in the search space, we need same model of the fitness function to utilize information of other individuals. In this paper, we adopt a simple model that fitness values of individuals distributed randomly around the fitness value of the individual of interest. We assume the variance of the fitness value depends only on the distance from the individual of interest.

Let $\mathbf{x}$ be an individual we want to estimate its fitness value, and let $\mathbf{h}$ be an individual in the history of search whose distance from $\mathbf{x}$ is $d$ and a sampled fitness value for $\mathbf{h}$ is $F(h)$. We assume the following model:

$$f(h) \sim N(f(x), kd)$$

$$F(h) = f(h) + \delta \sim N(f(x), kd + \sigma^2)$$

where $f(h)$ is the true fitness of individual $\mathbf{x}$, and $k$ is a positive parameter. As shown in Fig.1, Eqs. (3) and (4) mean that the true fitness $f(h)$ distributed randomly around the $f(x)$ following normal distribution of a variance proportional to the distance $d$, and hence observation of the fitness $F(h)$ follows the normal distribution of a variance $kd + \sigma^2$ considering additive observation noise $\delta$. The smaller the parameter $k$ is, the flatter fitness landscape it means, and the more information in the history can be used to estimate $f(x)$.

![Fig. 1. Model of search space](image)

3.2 Estimation of Fitness by the Maximum Likelihood Method

Assuming that parameters $k$ and $\sigma^2$ are known in advance, the fitness value $f(x)$ can be estimated by the maximum likelihood method using the search history
based on the above model. Let \( H_l, l = 1, \ldots, H \) be the \( H \) individuals in the search history and \( F(h_l) \) and \( d_l, l = 1, 2, \ldots, H \) be their sampled fitness values and distances from the individual of interest, respectively. The probability of obtaining \( F(h_1), \ldots, F(h_H) \) is represented by

\[
\prod_{i=1}^{H} p(F(h_i), d_i)
\]  

(5)

where \( p(F(h_i), d_i) \) is the probability density function of \( F(h_i) \) given by the normal distribution:

\[
p(F(h_i), d_i) = \frac{1}{\sqrt{2\pi(kd_i + \sigma^2)}} \exp\left(-\frac{1}{2} \frac{(F(h_i) - f(x))^2}{kd_i + \sigma^2}\right)
\]  

(6)

Treating Eq. (5) as the likelihood w.r.t. \( f(x) \), estimation of \( f(x) \) can be obtained by maximizing Eq. (5) for \( f(x) \). A necessary condition of the maximum likelihood estimate can be obtained by differentiating the logarithms of Eq. (5), and let it be equal to 0:

\[
\sum_{i=1}^{H} \frac{F(h_i) - f(x)}{kd_i + \sigma^2} = 0
\]  

(7)

From Eq. (7), the maximum likelihood estimation \( \tilde{f}(x) \) is obtained as follows:

\[
\tilde{f}(x) = \frac{\sum_{i=1}^{H} \frac{F(h_i)}{kd_i + \sigma^2}}{1 + \sum_{i=2}^{H} \frac{\sigma^2}{kd_i + \sigma^2}} = \frac{\sum_{i=1}^{H} \frac{F(h_i)}{kd_i + \sigma^2}}{1 + \sum_{i=2}^{H} \frac{\sigma^2}{kd_i + \sigma^2}}
\]  

(8)

Eq. (8) shows that the estimate is a weighted average of the sampled fitness values. The nearer the individual in the history to the individual of interest, the more weight is put on its sampled fitness in estimation. The RHS of Eq.(8) is obtained by assuming the \( h_1 \) be the sampled value at \( x \) itself.

### 3.3 Estimation of the Model Parameters

In actual situation, the parameters \( k \) and \( \sigma^2 \) in the Eq. (8) are not known in advance. Furthermore, the history of search changes together with the generation. Hence it is needed to estimate these parameters adaptively to change of the history. We also employ the maximum likelihood technique for estimation of \( k \) and \( \sigma^2 \). Taking Eq. (5) as the likelihood of the parameters \( k \) and \( \sigma^2 \), and a logarithm likelihood of the parameters is calculated from Eq. (5) as follows:

\[
\log L(k, \sigma^2) = -\frac{1}{2} \left( H \log 2\pi + \sum_{l=1}^{H} (kd_l + \sigma^2) + \sum_{l} \frac{(F(h_l) - f(x))^2}{kd_l + \sigma^2} \right)
\]  

(9)
Since this equation also includes unknown variable \( f(x) \) and \( d_i \), we set \( x \) at the individual which has the minimum sampled fitness considering that the usage of the model is optimization of \( f(x) \). The fitness value of \( f(x) \) is simply estimated by the average of the sampled fitness values of five individuals near by. Distance \( d_i \) is easily calculated if \( x \) is decided.

Considering positiveness of the parameters, a numerical hill climb method w.r.t. logarithms of \( k \) and \( \sigma^2 \) is used for maximization of the likelihood.

### 3.4 Prototype Algorithm of the Proposed Method

The following is a prototype algorithm of the proposed method. For selection operation we employ a steady state type one that picks a pair of parents from the population, apply a crossover operator \( C \) times, and substitute parents by the best two individuals among the union of parents and children. For crossover operator, we use the unimodal normal distribution crossover (UNDX) proposed by Ono et al. [5] considering application to optimization is continuous search space. The UNDX yields children normally distributed on a line connecting parents. Numerical experiment shows that it achieves excellent search ability both on multimodal and non-separable fitness functions. Since the search ability of the UNDX is high, we don’t employ mutation.

1. Initialize the population of \( M \) individuals \( x_1, \ldots, x_M \) randomly.
2. Let evaluation counter \( e = 0 \). Set maximal number of evaluations to \( E \).
3. Let history \( H = \phi \).
4. Choose two individuals \( x_{p1} \) and \( x_{p2} \) from the population as parents.
5. Produce \( C \) children \( x_1^c, \ldots, x_C^c \) by applying the crossover to the parents.
6. Let \( y_1 = x_{p1}, y_2 = x_{p2} \) and \( y_{i+2} = x_i^c, i = 1, \ldots, C \).
7. Sample fitness value \( F \) for \( y_i, i = 1, \ldots, C + 2 \).
8. Let \( e = e + C + 2 \).
9. Store the sampled values into the history \( H \), i.e.,
   \[
   H = H \cup \{(y_i, F(y_i)) | i = 1, \ldots, C + 2 \}.
   \]
10. Select the individual \( h_{\text{min}} \) having the smallest sampled fitness value from \( H \).
11. Estimate \( k \) and \( \sigma^2 \) by maximization of Eq. (9).
12. Estimate \( f(y_i), i = 1, \ldots, C + 2 \) by Eq. (8).
13. Substitute the individual having two smallest \( \tilde{f}(y_i) \) among \( \{y_1, \ldots, y_{C+2}\} \) into \( x_{p1}, x_{p2} \).
14. If \( e \leq E \), go to Step 4, otherwise terminate the algorithm.

The above algorithm seems complex and computational load is higher than the standard GA. However, in the practical applications of online adaptation, the total possible number of fitness evaluation is quite limited, and evaluation of a single individual takes long time. Hence the proposed method is applicable from the viewpoint of computational complexity.
4 Experiments

4.1 Optimization of a Noisy Sphere Function

In this section, we compare the performance of the proposed method with conventional methods through numerical experiments. For the test function, the following 10 dimensional sphere function with additive noise is used so as to confirm fundamental convergence properties of the methods:

\[ F_1(x) = \sum_{j=1}^{10} x_j^2 + \delta, \quad \delta \sim N(0, \sigma^2_{F_1}), \quad \sigma^2_{F_1} = 1.0, \]  

(10)

where \( x \) is \( \{x_1, x_2, ..., x_{10}\}^T \), and \( \delta \) is an additive noise following the normal distribution. Initial population is sampled in \([-0.5, 0.5]^{10}\) randomly.

The following three algorithms are compared.

- Standard GA: A GA using a single fitness sample value for evaluation of an individual.
- Sample 10-GA: A GA using the mean of 10 fitness value samples for evaluation of each individual.
- Proposed Method.

Same operators and parameters are employed except for estimation techniques of the fitness value. The population size \( M \) is 30, and the children size \( C \) is 5. These parameters are chosen considering application of our method to the practical control problem[1] that allows several hundred fitness evaluations. However, so as to examine the convergence characteristics of the methods, we extend maximum generation to 10000 fitness evaluations.

Performance of the tested methods is evaluated from two points of view. That is, how closely each algorithm converges to the optimum, and how accurately fitness values are estimated in each algorithm. For the first and the second viewpoints, indexes \( A \) and \( D \) are introduced respectively:

\[ A = \frac{1}{M} \sum_{m=1}^{M} f(x_m), \quad D = \frac{1}{P+C} \sum_{i=1}^{P+C} |\hat{f}(x_i) - f(x_i)| \]  

(11)

where \( A \) is the mean of the true fitness values of the individuals in the population, and \( D \) is the average of the deviation of the estimated fitness values from true ones. The results of the experiment are shown in Figs. 2.(a) ~ (c). Figure 2.(a) shows the evolution of index \( A \). In the standard GA, reduction of index \( A \) stagnates after around 2000 evaluations, and the value of \( A \) fluctuates around 0.3. Contrary to this, in the sample 10-GA exhausts all the evaluations before convergence while it succeeded in finding better solutions. The proposed method converges quickly and finds more accurate solutions than the other two methods. In Fig. 2.(a) result of GA applied to the fitness function without noise (the noiseless-GA) is also plotted. The proposed method achieve a convergence speed about half of the noiseless GA in early stage of the search.
578

Figure 2.(b) shows evolution of the index $D$. We can confirm that $D$ of sample 10-GA achieves better estimation than the standard GA as predicted by the theory. Concerning the proposed method, it achieve estimation of almost same precision with Sample 10-GA up to 5000 fitness evolution and it gets better later. Figure 2.(c) shows evolution of estimated model parameters $k$ and $\sigma^2$. Concerning $\sigma^2$, the estimated value are stable and near to the actual value 1. Concerning $k$, while no adequate reference value is available, it shows that $k$ fluctuates 0.2 to 2 continuously.

4.2 Optimization a Noisy Ellipsoid Function

Since the proposed method employs a stochastic model of fitness function using distance between individuals, performance of the algorithm depends on scaling of the coordinate system. Therefore, it is important to assess the sensitivity of the proposed method to the scaling of coordinate system. In the next experiment, sensitivity to the scaling is examined using the following noisy ellipsoid function as a fitness function:

$$F_2(\mathbf{x}) = \sum_{j=1}^{10} a_j x_j^2 + \delta, \quad \delta \sim N(0, \sigma_{F_2}^2), \quad \sigma_{F_2}^2 = 1.0$$

$$a_1, a_2, a_3 = 0.1, \quad a_4, a_5, a_6, a_7 = 1.0, \quad a_8, a_9, a_{10} = 10.0$$

where $\mathbf{x} = \{x_1, x_2, ..., x_{10}\}^T$ and $\delta$ is an additive noise. Initial population is sampled in $[-0.5, 0.5]^{10}$. Other conditions are same with the previous experiment.

The results of the experiment are shown in Figs. 2.(d) ~ (f). Figure 2.(d) shows the evolution of $A$. Concerning precision of convergence to optimal solution, results quite similar to the previous experiment are obtained. The proposed method achieves much quicker and more precise convergence than the other methods also for the noisy ellipsoid function. Figure 2.(e) shows evolution of index $D$. For ellipsoid function, precision of fitness estimation by the proposed method is intermediate of the other two methods up to 5000 evaluations. While $F_2$ is a separable function, it should be noted that the adopted crossover operator UNDX is an invariant operator under orthogonal transformation of the coordinate system, and therefore it works also well for non-separable functions.

Figure 2.(f) shows evolution of estimated $k$ and $\sigma^2$ for $F_2$. Compared with Fig. 2.(c), $\sigma^2$ is also estimated well for $F_2$. Concerning $k$, larger values are obtained. It is consistent with larger values of $D$ shown in Fig. 2.(e) twice larger $k$ implies less usage of the history as shown in Eq. (8).

5 Conclusion

In this paper, optimization of noisy fitness functions by GAs is discussed. Importance of reduction of function evaluations as well as precision of solution is pointed out considering applications of GAs for noisy fitness functions such as
Numerical experiments show that the proposed method outperforms the conventional methods both in convergence speed and precision of solution.

The following are subjects of future study:
1. Improvement of performance of estimation in early stage of search by introducing Bayesian technique instead of the maximum likelihood are. The Bayesian technique will accelerate parameters estimation by introducing a priori knowledge of the system.

2. To develop crossover operators which take tradeoff between the identification of the fitness function and its optimization into account.

3. Application of the proposed method to the problem of online adaptation of engine controller[1].

References


Evolvable Hardware
and Hardware Implementation of EAs
An Efficient Random Number Generation Architecture for Hardware Parallel Genetic Algorithms

Marc Bright and Brian Turton

Circuits and Systems Research Group, Cardiff University School of Engineering, Cardiff, CF24 0YF, UK

Abstract. A technique for random number generation for a specific hardware Parallel Genetic Algorithm (PGA) architecture is presented. The thermal noise properties of silicon semiconductor material are used to provide a single true-random white noise source. This is combined with a bit-serial propagation technique to produce multiple true-random parallel number streams for parallel genetic evolution. The results demonstrate that the proposed architecture produces spatially and temporally uncorrelated numbers while minimising the complexity of generation and propagation hardware.

1 Introduction

There has been increasing interest in the VLSI implementation of Parallel Genetic Algorithm’s (PGAs) in order to solve complex problems in real-time [1, 2, 3]. The PGA utilises genetic operators, such as crossover and mutation, to solve problems that typically cannot be solved with conventional solvers. The VLSI PGA architecture presented by Turton et al uses a fine grain VLSI PGA to produce solutions for image registration [1] and disc scheduling [4]. The architecture is illustrated in figure 1; it consists of a number of identical processors with communication paths to near neighbours. Each processor contains a single solution; all solutions are evolved in parallel through application of genetic operators. Each processor requires random numbers during each generation to determine which type of genetic operator to apply and how it is subsequently applied. The parallel evolution approach therefore requires parallel random number streams for each processor with temporal and spatial independence qualities. These qualities are essential to ensure that the selection of operator is not biased, as this will compromise exploration.

Bland et al [5] proposed a technique for parallel random number generation based on the use of a number of Pseudo-Random Number Generators (PRNGs). Replication of PRNGs across the device, each seeded by the preceding PRNG, enables parallel generation. Such PRNGs suffer from dependence on the initial seed and the careful selection of operating parameters to ensure a sufficiently long PRN, which will repeat (and in precisely the same sequence) once the end of period conditions are met [6]. In addition, the replication of PRNGs increases the complexity and area of the VLSI device. Similar approaches have been proposed Anguita et al [7], who takes
advantage of dependence on the seed to create reproducible PRN streams, and Al-
spector et al [8] who uses a unique ‘mask’ for each neuron in a neural network to
reduce correlations. The mask is applied to a PRN that is the same for all neurons,
thus requiring only a single PRNG. However, the generation of masks to produce
apparently independent numbers is a non-trivial task and requires multi-layer XOR
networks that increase complexity. A common feature of these techniques is the use
of PRNGs that produce pseudo-random numbers with some degree of correlation.
This letter presents a technique for the parallel generation of true-random number
streams while minimising complexity.

2 Implementation

The technique is based on the amplification of thermal noise in VLSI devices, which
has been demonstrated as a viable source of random bit streams in [9] and [10].
Thermal noise has good white-noise qualities within a specified frequency range and
can produce a stream of independent bits at demonstrated rates of up to 1MHz [9].
This generator requires considerable area so the generation scheme should be limited
to using one RNG.

Fig. 1. The hardware GA architecture [1].
Figure 1 presents an overview of the architecture on top of the current PGA architecture. A single noise based RNG provides input to a bit-serial data path. The bit-serial data path reduces the area and power of the architecture. The path consists of an out and back segment. Each segment passes every chromosome once, through N-bit windows as illustrated in figure 2. The out and back random data in these windows is XOR'ed to produce a unique random number for each chromosome in parallel, ensuring the PGA can operate at its maximal rate. Spatial independence in the same generation is ensured as the thermal noise source produces a true-random bit stream. The back segment is essential to remove near-neighbour temporal correlation i.e. on successive generations, as a neighbour along the direction of the out segment would receive some bits of the previous chromosomes random number. By using the back segment as a random mask for the out segment such near-neighbour temporal correlation is removed. An important advantage of the presented technique is this use of a continuously varying random mask to eliminate correlations.

![Fig. 2. Generation of random number for each processor](image)

During a generation each chromosome requires 4 random numbers; one each to determine the probabilities of mutation and crossover, and two to determine the positions of two-point crossover. With a bit rate of 1MHz each 8-bit random number is generated in 8 microseconds, limiting the PGA to processing a generation every 32 microseconds. However the genetic mutation and crossover operations of each processor typically take at least 2-3 milliseconds [1] so generating the random numbers in parallel with these processes does not limit the processing rate.
3 Results and Analysis

The key aspects of the RNG architecture are its correlation properties in the three cases of:

A. Correlation between successive random numbers of a chromosome (temporal)
B. Correlation between the random numbers of chromosome X at generation a and chromosome Y at generation a+1: Chromosome Y is a neighbour to chromosome X along the direction of the out segment (temporal and spatial).
C. Correlation between the random numbers of neighbouring chromosomes in the same generation (spatial).

The architecture's principal parameters are the random number bit-length (the size of the XOR window) and the number of random-bits to be generated for each evaluation i.e. the number of required bit-shifts. A low number of shifts is advantageous in terms of speed and power consumption, but the effect on correlation, particularly of type A and B, needs to be analysed. The size of the random number is dictated by the required accuracy of the PGA. Typical PGA implementations specify mutation operations to 0.5% accuracy, thus requiring 200 levels. This dictates a minimum size of 8 bits for the random number (256 levels).

The system was simulated to produce correlation statistics and assess the optimum bit-shift length. Each analysis consists of data obtained from 50,000 PGA generations. For simulation purposes a software random number generator was selected [6] with an analysed autocorrelation of 0.02%. Table 1 summarises the correlations determined with the hardware parallel random number generator with 1-bit and 4-bits generated for each genetic evaluation. These results were produced by simulating the system ten times to reduce the stochastic effects of the software random number generation process; therefore, the results are an average of the ten simulations.

Table 1 clearly illustrates the advantage of the back segment. Without this segment, when only generating 1-bit per evaluation, an individual chromosome will have a random number that contains seven bits of the previous random number (correlation type A). All other correlation's are low as is expected given the correlation statistics of the random number generator itself. The inclusion of the back segment results in a dramatic reduction in type A correlation. This is due to the XOR operation ensuring that the random number generated is the result of two random numbers, moving in opposite directions. When the number of bits generated per evaluation is increased to 4-bits the table illustrates that correlation, particularly of type A, is further reduced. This is due to the out and back segment windows now contain fewer bits (compared to 1-bit results) of the previous timestep's random number. Without the back segment, correlation type A still decreases but with an associated increase in type B due to the chromosome along the out segment containing an increasing portion of its direct predecessor's previous random number. The use of the back segment ensures that both A and B are decreased as the bit-shift length is increased.

The absolute bit-shift length selected must be chosen in accordance with the implementation requirements. Large bit-shifts significantly reduce correlation. However, even small shifts produce low levels of correlation with the added advantages of speed and less power consumption per evaluation.
This analysis illustrates that the back segment prevents near-neighbour temporal correlation and also preserves the properties of same chromosome temporal correlation. Simulations and analyses of type B and C correlation produced values of <0.5\% and <0.35\% respectively.

4 Conclusions

In contrast to other techniques that require dedicated generation circuitry across the VLSI device [5,9] the architecture utilises a single true-random source with an efficient bit-serial propagation architecture, reducing the VLSI area and power requirements. The registers required to hold and shift the random numbers are part of the present hardware architecture, therefore the proposed architecture only requires extra control and exclusive-or circuitry. The initial PGA architecture was based on a 1-micron CMOS process [1], with the use of a smaller feature size the propagation architecture could be incorporated without any significant area penalty.

Bit-serial architectures are inherently low power and the use of a clocked shift scheme minimises the propagation of power consuming glitches [11]. An analysis of correlation in the spatial and temporal domains illustrates the effectiveness of the architecture in generating independent parallel streams of true-random numbers. The architecture is scalable for hardware PGAs with different numbers of processors.

<table>
<thead>
<tr>
<th>Type</th>
<th>Correlation with back segment bit-shift=1 (%)</th>
<th>Correlation without back segment bit-shift=1 (%)</th>
<th>Correlation with back segment bit-shift=4 (%)</th>
<th>Correlation without back segment bit-shift=4 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.1</td>
<td>50.2</td>
<td>0.6</td>
<td>6.5%</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>0.6</td>
<td>0.01</td>
<td>5.9%</td>
</tr>
<tr>
<td>C</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

References


An Integrated On-Line Learning System for Evolving Programmable Logic Array Controllers

Yong Liu¹, Masaya Iwata², Tetsuya Higuchi², and Didier Keymeulen²

¹ The University of Aizu
Aizu-Wakamatsu, Fukushima 965-8580, Japan
yliu@u-aizu.ac.jp
http://www.u-aizu.ac.jp/yliu/
² Evolvable Systems Laboratory
Computer Science Division, mbox 1501
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305-8568, Japan
{iwata,higuchi}@etl.go.jp
http://www.etl.go.jp/~ehw/

Abstract. This paper presents an integrated on-line learning system to evolve programmable logic array (PLA) controllers for navigating an autonomous robot in a two-dimensional environment. The integrated on-line learning system consists of two learning modules: one is the module of reinforcement learning based on temporal-difference learning methods, and the other is the module of evolutionary learning based on genetic algorithms. The control rules extracted from the module of reinforcement learning can be used as input to the module of evolutionary learning, and quickly implemented by the PLA through on-line evolution. The on-line evolution has shown promise as a method of learning systems in complex environment. The evolved PLA controllers can successfully navigate the robot to a target in the two-dimensional environment while avoiding collisions with randomly positioned obstacles.

1 Introduction

There are two main approaches to the learning problem of the robot in an unknown and/or dynamic environment [1]. The first approach is based on evolutionary algorithms, such as genetic algorithms [2], evolutionary programming [3], and evolution strategies [4]. It searches directly in the space of possible policies for one performing well in the environment. The second approach is based on reinforcement learning [5], which uses statistical techniques and dynamic methods to estimate the utility of taking actions in states of the environment. In robot learning, researchers in both evolutionary learning and reinforcement learning often employ off-line learning, i.e. learn a control policy in simulation and then apply the learned control policy to the real system.

However, it was observed that off-line learned control policies sometimes could not cope well with the new environment that is different with the environment used in the training process [6, 7]. In such dynamic environment, on-line
learning will become more important, which allows the robot to respond to unexpected situations and learn while engaged in the problem-solving process. On-line learning implies the learning systems that learn as each experience arrives rather than store the experiences for use in a separate off-line learning phase. Learning from on-line experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behaviour. Unfortunately, on-line learning is often time-consuming. In order to speed up on-line learning, this paper presents an integrated on-line learning system to evolve programmable logic array (PLA) controllers for navigating an autonomous robot in a two-dimensional environment.

In robot learning, many of papers are concerned with neural networks as controllers [1]. In this paper, a PLA is used as a controller. The PLA device consists of a programmable AND array cascaded to a programmable OR array. This provides flexibility and maximum minimisation capability prior to embedding of the desired Boolean function into the PLA device. As indicated by Lund and Hallam [8], the Boolean function is sufficient for the adaptation to an environment required by the avoidance task. The main advantages of the PLA device over neural networks are that the architectures of the PLA device can be quickly reconfigured and easily implemented in the hardware chip. The reconfiguration of the PLA device can be autonomously obtained by evolvable hardware (EHW) [9]. EHW is based on the idea of combining reconfigurable hardware devices, such as PLA and field-programmable gate array (FPGA), with evolutionary algorithms to execute reconfiguration autonomously.

The integrated on-line learning system consists of two learning modules: one is the module of reinforcement learning based on temporal-difference learning methods, and the other is the module of evolutionary learning based on genetic algorithms. The function of the module of reinforcement learning is to extract control rules from interactions between the robot and the environment. The extracted control rules can therefore be used as input for the module of evolutionary learning. Since the knowledge has already been embedded within the extracted rules, evolutionary learning is to implement the existing rules using the PLA rather than to search the PLA controllers directly. In this way, the implementation of evolutionary learning can be made simple enough to be carried out together with the reconfigurable hardware within a single EHW chip [9].

The rest of this paper is organised as follows. Section 2 formulates the robot navigation problem as a Markov decision process. Section 3 studies how to deal with robot navigation by reinforcement learning. Section 4 gives the ideas on the design of the PLA controllers by evolutionary learning. Section 5 describes the integrated on-line learning system for evolving the PLA controllers. Finally, Section 6 concludes with a summary of the paper and a few remarks.

2 Robot Navigation System

The robot navigation system described in this paper is a computer simulation. A real robot named Evolver has been built based on this simulated system [6]. The
shape of the robot in simulation is circular with a diameter of of 25 cm. It has 10 infrared sensors equally distributed around its periphery. The 10 infrared sensors indicate the presence of objects at distance of less than 30 cm. It is equipped with four bumping sensors to detect when the robot collides with the obstacles at either the front or the back. Two cameras are mounted on the top of the robot, which are able to recognise and track a colored object. The cameras locate the target within the four directions around the robot. The robot is moved by two independent motor wheels, which are controlled in terms of speeds. The robot can be allowed to perform eight preprogrammed actions: forward, backward, left, right, and four combinations of rotations with forward and backward actions.

The task assigned to the robot is to track a colored object within a limited number of actions while avoiding obstacles and walls in a two-dimensional environment. There are obstacles of different shapes in the environment enclosed by the walls. To perform its task, the robot must learn the basic behaviour of obstacle avoidance and moving to the target. It must also learn to co-ordinate the behaviour of obstacle avoidance and the behaviour of moving to the target to avoid becoming stuck due to repetition of an identical sensor-motion sequence.

In its simplest form, the tracking and avoiding problem can be formulated as a Markov decision process (MDP) [10]. In the MDP, the robot can perceive a set $S$ of possible states of its environment and has a set $A$ of actions that it can perform. At each discrete time $t$, the robot senses the current state $s_t$, chooses a current action $a_t$, and performs it. The environment responds by giving the robot a reward $r_t = r(s_t, a_t)$ and by producing the succeeding state $s_{t+1} = T(s_t, a_t)$. Here the transition function $T$ and the reward function $r$ are part of the environment, and are not necessarily known to the robot. In a MDP, the functions $T$ and $r$ depend only on the current state and action, and not on earlier states or actions.

The goal of a MDP is to determine the expected reward for each state-action pair. That is, it is to learn a policy, $f : S \rightarrow A$, which is a mapping from perceived states of the environment to actions to be taken when in those states. Two different representations of policy $f$ have been used in evolutionary algorithms and reinforcement learning. In evolutionary algorithms, the policy is normally represented as a mapping function that maps the next action $a$ from the current observed state $s$. For example, in robot navigation, a decision policy may map sensor values to wheel speeds. In reinforcement learning, the policy is represented as value functions, such as action-value function, denoted as $Q(s, a)$, that is the expected return of starting from state $s$, taking the action $a$, and thereafter following policy $f$. The following sections will discuss how to learn the policy $f$.

3 Reinforcement Learning

3.1 Temporal-Difference Learning

A central and novel idea of reinforcement learning is called temporal-difference (TD) learning [5]. TD learning is a combination of Monte Carlo ideas and dy-
namic programming ideas. Like Monte Carlo methods, TD methods can learn directly from the raw experience without a model of environment’s dynamics. Like dynamic programming methods, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome. TD learning works because it is possible to make local improvements. At every point in the state space, the Markov property allows actions to be chosen based only on knowledge about the current state and the states reachable by taking the actions available at that state.

TD methods fall into two classes: on-policy and off-policy [5]. One of the most important breakthroughs in reinforcement learning was development of an off-policy TD control algorithm known as Q-learning [11]. The learned action-value function, \( Q(s, a) \), directly approximates the optimal action-value function, independent of the policy being followed. Sarsa learning algorithm is an on-policy TD learning method in which the action-value function \( Q \) is updated after every transition from a nonterminal state. Sutton and Barto compared Sarsa learning with Q-learning on a gridworld example [5]. The results showed that the on-line performance of Q-learning is worse than that of Sarsa learning. In this section we discuss how to design a control algorithm based on Sarsa learning. The application of Q-learning will be studied in the integrated on-line learning system in Section 5.

3.2 Sarsa Learning

The major steps of Sarsa learning are given as follows [5]:

1. Initialise \( Q(s, a) \) values arbitrarily.
2. Initialise the environment.
3. Choose action \( a \) using the policy derived from \( Q(s, a) \) (e.g., \( \epsilon \)-greedy).
4. Take action \( a \); Observe reward \( r \) and the next state \( s' \); Choose the next action \( a' \) using the policy derived from \( Q \) (e.g., \( \epsilon \)-greedy).
5. Update the \( Q(s, a) \) as follows:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]
\]  

6. Let \( s \leftarrow s' \) and \( a \leftarrow a' \). Go to the next step if the state \( s \) is a terminal state. Otherwise, go to Step 3
7. Repeat Steps 2 to 6 for a certain number of episodes.

Figures in Fig. 1 show the simulated environment. Each environment is a 2000 x 2000 environment enclosed by the walls. There are a number of obstacles of different shapes in the environment. The left figure in Fig. 1 shows the result of applying Sarsa learning with \( \epsilon = 0.1, \alpha = 0.1, \gamma = 0.9 \), and the initial values \( Q(s, a) = 0 \) for all \( s \) and \( a \), on a training set with 50 starting points. A look-up table was used to store \( Q(s, a) \). In Sarsa learning, the robot starts from one point in the training set. It moves by selecting from a discrete set of actions according to \( Q \) with \( \epsilon \)-greedy policy by setting \( \epsilon = 0.1 \). Sarsa learning gives each action \( a \) with immediate reward \( r \). The function \( Q \) is then updated according
to Eq.(1). The robot moves until it arrives at the target position or reaches a
certain number of steps. One episode then ends and Sarsa learning starts a new
episode from the same starting point until Sarsa learning learns how to reach
the target or reaches a certain number of episodes. Sarsa learning repeats the
above learning process for each point in the training set. After Sarsa learning,
the learned control policy $Q$ could successfully navigated the robot to the target
position from all 50 starting points.

The learned control policy $Q$ was tested on a testing set with another ten
starting points in the same environment. The control policy also successfully
navigated from all the points in the testing set. However, when new obstacles
were added in the environment, the learned $Q$ successfully navigated from only
four starting points in the same testing set. The robot driven by the learned $Q$
was stuck in the loops from other six starting points. The result is shown in the
middle figure in Fig. 1. In another test of the learned policy $Q$ showed in the
right figure in Fig. 1, the robot was placed in a new environment to navigate
from ten different points. The robot controlled by the learned policy $Q$ failed to
reach the target five times.

Two main reasons cause the failure of the learned $Q$ policy on the robot
navigation in the changed environment. The $Q$ policy is learned off-line. It cannot
cope well with the changed environment. Another reason is that the control
policy $Q(s, a)$ is represented by a look-up table in Sarsa learning. When dealing
with an environment that can be broken into discrete states, the lookup table
provides the most obvious method. However, such representation completely
separates the information learned among states, and there is no generalisation
on the similar states.

![Fig. 1. Traces of the robot controlled by the $Q$ policy obtained from Sarsa learning. The left figure shows the traces starting from 50 points in the training set. The middle figure shows the traces starting from ten points in the testing set in a changed environment. The right figure shows the traces of the robot from ten points in a new environment. The smallest square denotes the target.](image-url)
4 Evolving Programmable Logic Array

An evolutionary approach normally searches directly in the space of policies without ever appealing to value functions. In this section, an off-line evolutionary learning is presented to evolve the PLA controllers. The architectures of the PLA are evolved by a steady-state genetic algorithm with uniform crossover and bit-by-bit mutation. There are several characteristics, as indicated by Kajitani et al. [12], which make the steady-state genetic algorithm better choice for hardware implementation.

4.1 Policy Representation

In evolutionary learning, the control policy $f$ is represented by $m$ Boolean formula in $k$-term disjunctive normal form (DNF), which consists of a disjunctive of a maximum of $k$ terms. Each term is the conjunction of Boolean variables or their complement so that the policy $f$ can easily be implemented by the PLA that has a limited number of logic gates. Besides the PLA is an efficient way to implement a set of logic functions, the architecture of the PLA is readily expressed in binary string which can be evolved in evolutionary algorithms. In the simulated robot, the $n = 12$ input variables of the Boolean functions are the binary representation of the sensors, in which 10 bits for the infrared sensors and 2 bits for the vision sensors. The $m = 3$ output values of the function $f$ are the possible actions.

The PLA for executing the policy $f$ consists of a programmable AND array cascaded to a programmable OR array. Each row on the AND array calculates the product of the input connected to it, and each column of the OR array calculates the sum of the results from the output of the AND gates if connected. With $n = 12$ and $m = 3$, the AND-OR array has $2 \times n + m = 27$ columns and needs a maximum number of $2^n = 4096$ rows to represent any Boolean functions with 12 input and 3 output. However, in order to force the Boolean function to generalise, the number of rows can be reduced by merging rows with the same output. Because of the limitation of number of gates available in the PLA, the number of rows is set up to $k = 32$ in the simulation. The architecture bits of the PLA with 32-term DNF can be encoded by $27 \times 32 = 864$ bits.

4.2 Fitness Evaluation

The fitness of each PLA is obtained by executing the PLA to navigate the robot in the environment from the different starting points. For each starting point, the PLA execution continues until the robot arrives at the target, or collides with an obstacle, or reaches the maximum number of steps allowed to perform in the environment. The PLA receives a payoff based on the robot’s final position. The payoff is 1 if the robot arrives at the target, otherwise it is 0. There are 50 starting points in the training set. 25 points are predetermined and the rest points are randomly selected in the environment. This forces the robot to encounter many different environment states. The value of fitness for each PLA is between 0 and 50.
4.3 Experimental Results

To compare evolutionary learning with Sarsa learning, the experiments have been conducted in the same environment used in Section 3.2. The population size (32) and the maximum number of generations (2000) were set up in evolutionary learning. At the beginning of the evolution, the 864 bits for each individual PLA were initialised at random.

The left figure in Fig. 2 shows the result of the best PLA controller in the population at last generation. The evolved PLA was able to successfully navigate from the 50 initial points without bumping any obstacles or being stuck after 8130 actions. It also shows that the PLA controller coordinated both the obstacle avoidance behaviour and the target tracking behaviour. For example, the PLA controlled the robot to follow the walls or the obstacles until there was no obstacle in the direction of the target. The off-line evolved PLA was tested on the same testing set used in Sarsa learning in the same environment. The PLA had no trouble to control the robot to the target from the ten testing points. When the new obstacles were added in the environment. The evolved PLA was able to generalise in the changed environment and navigated the robot successfully to the target. The results are shown in the middle figure in Fig. 2.

It is interesting to compare the traces of the robot controlled by the PLA policy and the Q policy in Figs. 1 and 2. For the same environment and the same starting points, the left figures in Figs. 1 and 2 show that the Q policy leads to shorter traces and takes 6129 actions to navigate from the 50 starting points while the PLA policy needs 8130 actions for the same task. However, the PLA policy showed better generalisation than the Q policy represented by look-up table in the changed environment. The right figure in Fig. 2 shows the result of the PLA policy in a new environment. It is clear that neither the PLA policy nor the Q policy could successfully navigate the robot for all ten starting points in the new environment. The PLA policy failed three times while the Q policy failed five times. The results indicate that off-line learned policies in both reinforcement learning and evolutionary learning are not robust enough to deal with the new environment because the configurations of the sensors had not been experienced before and could not be anticipated through generalisation.

5 Integrated On-line Learning System

On-line evolution is more adaptive to changes in the environment since such changes are immediately reflected in the fitness evaluation of policies. However, in robot navigation, on-line evolution is often time-consuming since the large number of interactions with the environment are needed to learn effective behaviour [13].

To learn with fewer interactions with the physical environment and still maintain good on-line performance, an integrated on-line learning system was developed to evolve the PLA controllers. There are two modules in the integrated learning system. The module of reinforcement learning is used to extract control
rules. The module of evolutionary learning then uses the extracted control rules as training patterns so that the fitness evaluation of the PLA can be measured on how many rules the PLA maps correctly. Since the fitness evaluation of the PLA is not required to be executed in the environment, the number of direct interactions between the PLA and the environment can be dramatically reduced. It is especially important for applications where real-world experience is costly while computation would be considered as cheaper.

The module of reinforcement learning is based on Q-learning [11]. Q-learning continually updates the policy $Q$ throughout its lifetime. Its simplest form, one-step Q-learning, is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

(2)

The module of evolutionary learning is implemented periodically after the robot executes a certain number of actions or the robot collides with an obstacles. Different with the off-line evolutionary learning in Section 4, this module is to evolve the PLA to implement the extracted control rules from the $Q$ function rather than to directly search the space of the PLA policies. A steady-state genetic algorithm with uniform crossover, bit-by-bit mutation, and a gene replacement mutation [12] was used in the module of evolutionary learning. The idea of the gene replacement mutation is to replace a part of a chromosome with a bit string called as “chromosome candidate segment” [12]. The chromosome candidate segment can be generated from a training input-output pattern used for the evaluation of the PLA. The reason to adopt this genetic algorithm is that it can be carried out together with the reconfigurable hardware within a single EHW chip [9].
5.1 Simulation Experiments

Compared with the off-line evolutionary learning, the integrated on-line learning system was tested on the same environment. Observed from the traces in the right figure in Fig. 2, the failure of the off-line learned policy was caused either by a collision or by a loop of an identical sensor-motion sequence. In order to deal with these two problems, a strategy of combining the off-line evolved PLA and the on-line evolved PLA was adopted. The basic principle of the strategy is to make use of the off-line evolved PLA when possible and to execute the on-line evolved PLA otherwise. The purpose of execution of the on-line evolved PLA is to either avoid the collision or move to a location from which the off-line evolved PLA can break out of such a loop and again makes progress towards accomplishing the task.

The combined policies are more robust than the off-line evolved PLA and cope well with the new environment. The robot could reach the target from all ten testing starting points. The result is shown in the left figure in Fig. 3. Sarsa learning, the off-line evolutionary learning, and the integrated on-line learning system were further tested in another two new environments. The results showed that only the integrated on-line learning system can successfully navigate the robot without failure. The results of the the integrated on-line learning system are shown in the middle figure and the right figure in Fig. 3.

![Fig. 3. Traces of the robot controlled by the strategy of combining the off-line evolved PLA and the on-line evolved PLA. The smallest square denotes the target.](image)

6 Conclusions

This paper proposes an integrated on-line learning system to evolve the PLA controllers for an autonomous robot with limited sensory input. The on-line
evolution has shown promise as a method of learning systems in complex environment. The learned PLA controllers can successfully navigate the robot to a target in the two-dimensional environment while avoiding collisions with randomly positioned obstacles.

From the simulation results, it would appear that the integrated on-line learning system provides one of the most efficient and reliable learning methods. However, the integrated on-line learning system needs to be tested on a real robot. One of our current research projects is to implement the integrated on-line learning system to design EHW chips for a real robot Evolver [6].

References

Combinatorial Optimisation
Selection and Reinforcement Learning for Combinatorial Optimization

A. Berny
IRIN, Université de Nantes, France
berny@irin.univ-nantes.fr

Abstract. Improving on a previous paper, we explicitly relate reinforcement and selection learning (PBIL) algorithms for combinatorial optimization, which is understood as the task of finding a fixed-length binary string maximizing an arbitrary function. We show the equivalence of searching for an optimal string and searching for a probability distribution over strings maximizing the function expectation. In this paper however, we will only consider the family of Bernoulli distributions. Next, we introduce two gradient dynamical systems acting on probability vectors. The first one maximizes the expectation of the function and leads to reinforcement learning algorithms whereas the second one maximizes the logarithm of the expectation of the function and leads to selection learning algorithms. We finally give a stability analysis of solutions.

1 Introduction

Statistical approaches for optimization and learning have been used for many years. Only in the restricted case of Bernoulli distributions, one can cite Stochastic Learning Automata from Narendra and Thathachar [10], Associative Reward Penalty from Barto and Anandan [2], REINFORCE from Williams [14], BSC from Syswerda [13], Population-Based Incremental Learning from Baluja and Caruana [1], Relative Payoff Procedure from Dayan and Hinton [6] and Univariate Marginal Distribution from Mülhenbein [9]. All these algorithms update a probability vector to increase the expected reward, that is the expectation of the function to be maximized relatively to the Bernoulli distribution given by the probability vector. Algorithms using more complex probability distributions, such as those exhibiting chain or tree-like dependency graphs, have been proposed to overcome the limitations of those based on Bernoulli distributions (see [8] for a review). However, those limitations are not clearly identified. What is known is that Bernoulli algorithms perform better than hill climbers at a relatively low computational cost.

In this paper, we analytically study Bernoulli algorithms seen as gradient dynamical systems, putting the stress on their local behavior. We extend the work presented in [4] and apply some of the ideas of [3] in the context of binary string search.

In Section 2, we introduce a statistical framework for optimization over fixed-length binary strings. We show that maximizing the expectation of the function
to be maximized is equivalent to maximizing the function itself. In Section 3, we introduce two equivalent gradient dynamical systems acting on probability vectors, leading respectively to reinforcement and selection learning (PBIL). Section 4 gives a Liapunov stability analysis of exact strings which relies on combinatorial techniques. Section 5 is an introduction to other stability issues.

2 Statistical Framework

2.1 Introduction and Notations

The problem is to maximize an arbitrary function over the set of fixed-length binary strings. We recast this problem as the problem of finding a distribution over binary strings maximizing some statistical criterion. The criterion is the expectation of the function relatively to the distribution. We will show that both problems are equivalent.

Let \( S = \{0,1\}^n \) be the set of binary strings of length \( n \). We will also refer to \( S \) as the \( n \)-dimensional hypercube. Let \( f : S \to \mathbb{R} \) be the function to maximize. We require that \( f \) be positive, that is \( f \geq a > 0 \). Let \( M(S) \) be the set of distributions over \( S \) and \( J : M(S) \to \mathbb{R} \) the statistical criterion defined for each \( \mu \in M(S) \) as 
\[
J(\mu) = E_\mu(f) = \sum_{x \in S} \mu(x)f(x).
\]
From the positivity of \( f \), we have \( J \geq a > 0 \).

Let \( U = \{ x \in S : f(x) = f^* \} \), where \( f^* = \sup_{S} f \), and \( V = \{ \mu \in M(S) : J(\mu) = \sup_{\mu \in M(S)} J \} \) be the optimal sets for \( f \) and \( J \) respectively. Let \( \delta_a \) be the Dirac distribution charging point \( a \in S \), that is \( \delta_a(x) = 1 \) if \( x = a \) and \( \delta_a(x) = 0 \) otherwise.

2.2 Equivalence of Problems

We first observe that \( J(\mu) \leq f^* \) and that \( J(\delta_a) = f^* \) for all \( a \in U \). Consequently, \( \{ \delta_x : x \in U \} \subset V \). We have the following theorem:

**Theorem 1.** \( \mu \in V \) if and only if \( \mu(x) = 0 \) for all \( x \in S \setminus U \) or, put another way, \( \mu \) is a mixture of Dirac distributions charging points in \( U \).

**Proof.** Let \( \mu \in V \). Then
\[
J(\mu) = f^* \\
= \sum_{x \in U} \mu(x)f(x) + \sum_{x \in S \setminus U} \mu(x)f(x) \\
= f^* \sum_{x \in U} \mu(x) + \sum_{x \in S \setminus U} \mu(x)f(x).
\]

Hence,
\[
f^* \left(1 - \sum_{x \in U} \mu(x)\right) = \sum_{x \in S \setminus U} \mu(x)f(x) \\
f^* \sum_{x \in S \setminus U} \mu(x) = \sum_{x \in S \setminus U} \mu(x)f(x).
\]
Suppose there exists an $x \in S \setminus U$ such that $\mu(x) > 0$. We divide both sides of the above equation by $\sum_{x \in S \setminus U} \mu(x)$ and obtain

$$f^* = \sum_{x \in S \setminus U} \left( \frac{\mu(x)}{\sum_{x \in S \setminus U} \mu(x)} \right) f(x).$$

$f$ is upper bounded over $S \setminus U$ by $\sup_{S \setminus U} f < f^*$. We finally reach the contradiction $f^* \leq \sup_{S \setminus U} f < f^*$ and $\mu(x) = 0$ for all $x \in S \setminus U$. Conversely, if $\mu(x) = 0$ for all $x \in S \setminus U$ then $J(\mu) = \sum_{x \in U} \mu(x) f^* = f^*$.

2.3 Bernoulli Distributions

We restrict the search for an optimal distribution to a simple family of distributions, the Bernoulli distributions. More precisely, we make the assumption of independent random variables for the components of binary strings.

In order to express Bernoulli distributions, we introduce the function $\phi : \{0,1\} \times [0,1] \to [0,1]$ defined by

$$\phi(x_i, v_i) = \begin{cases} v_i & \text{if } x_i = 1 \\ 1 - v_i & \text{if } x_i = 0 \end{cases}$$

which can be interpreted as the probability of the $i$th component $x_i$ of binary string $x = (x_1, \ldots, x_n) \in S$. Let $\Delta$ be the box $[0,1]^n$ whose vertices are the points of $S$. $v = (v_1, \ldots, v_n) \in \Delta$ is called a probability vector. We define the Bernoulli distribution $\mu_v$ associated to the probability vector $v$ as

$$\mu_v(x) = \prod_{i=1}^{n} \phi(x_i, v_i).$$

Observe that if $v \in S$ then $\mu_v = \delta_v$.

3 Dynamical System Approach

3.1 Reinforcement Learning

We search for dynamical systems acting on probability vectors of Bernoulli distributions in order to maximize the statistical criterion $J$. We first turn to gradient techniques and introduce the dynamical system

$$v' = \grad J, 
\tag{1}$$

where $v'$ is the time derivative of the solution $v(t)$. However, (1) does not constrain $v$ to stay in $\Delta$, which is the condition to interpret $v$ as a probability vector. Thus, we stabilize (1) with the poles 0 and 1, by setting, if $v' = \varphi(v)$ and $\varphi = (\varphi_1, \ldots, \varphi_n)$,

$$\varphi_i = v_i (1 - v_i) \frac{\partial J}{\partial v_i}. \tag{2}$$
Let \( \Omega \) be the interior of \( \Delta \), that is \( \Omega = (0, 1)^n \). By continuity arguments, if \( v(0) \) is in \( \Omega \), so is \( v(t) \) for all \( t \in \mathbb{R} \). In most cases, one chooses \( v(0) = (1/2, \ldots, 1/2) \), which reflects the lack of a priori information on the problem.

We relate the polynomial \( v_i(1 - v_i) \) to the derivative of the logistic function which is a sigmoid function. Let \( g : \mathbb{R} \rightarrow (0, 1) \) be the logistic function defined as

\[
g(x) = \frac{1}{1 + e^{-x}}.
\]

We have the relation \( g' = g(1 - g) \). This suggests the introduction of a vector \( \theta \in \mathbb{R}^n \), with \( \theta = (\theta_1, \ldots, \theta_n) \), whose components are defined by \( \theta_i = g^{-1}(v_i) \). The dynamical system acting on \( \theta \) is simply

\[
\theta'_i = \frac{\partial J}{\partial v_i} \Big|_{g(\theta_i)}.
\]

In order to calculate \( \partial J/\partial v_i \), we first exchange the derivative and the sum over \( S \), which gives

\[
\frac{\partial J}{\partial v_i} = \sum_{x \in S} \frac{\partial \mu_x}{\partial v_i} f(x). \tag{3}
\]

We write

\[
\frac{\partial \mu_x}{\partial v_i} = \frac{\partial \log \mu_x}{\partial v_i} \times \mu_x(x)
\]

which enables us to interpret the sum over \( S \) in (3) as an expectation. We finally write the gradient of \( J \) like

\[
E(e(x_i, v_i)f(x)), \tag{4}
\]

where

\[
e(x_i, v_i) = \frac{\partial \log \mu_x(x)}{\partial v_i} = \begin{cases} -1/1 - v_i & \text{if } x_i = 0 \\ 1/v_i & \text{if } x_i = 1 \end{cases}
\]

is called the eligibility in the context of reinforcement learning.

Using a stochastic approximation with a comparison scheme, we obtain the following classical REINFORCE algorithm [14]

\[
\theta_i(t + 1) = \theta_i(t) + \alpha e(x_i, v_i) \left( f(x(t)) - E(f) \right),
\]

where \( \alpha > 0 \) is the learning rate and \( x(t) \) is the sample generated at time \( t \) from the distribution \( \mu_x \). The algorithm we have presented in [4] is slightly different.
3.2 Selection Learning

*Proportional selection* Instead of a gradient dynamical system maximizing \( J \), we consider a gradient dynamical system maximizing \( \log J \) whose vector field is given by

\[
\varphi_i = v_i (1 - v_i) \frac{\partial \log J}{\partial v_i}.
\]

Using expression (4) for the gradient of \( J \) and the simple relation

\[
\frac{\partial \log J}{\partial v_i} = \frac{1}{J} \frac{\partial J}{\partial v_i},
\]

we find

\[
v'_i = E\left( (x_i - v_i) \times \frac{f(x)}{E(f)} \right).
\]

Expressions (2) and (5) define the same field of directions and the resulting dynamical systems are thus equivalent.

We interpret the right-hand side of (6) as the effect of a fitness proportional selection operator, which is allowed by the fact that \( f \) is positive. Let us introduce the distribution \( \nu \) which is the image of \( \mu_v \) under the transformation

\[
\mu_v(x) \rightarrow \nu(x) = \frac{f(x) \mu_v(x)}{E(f)}.
\]

We can think of \( \nu \) as an infinite population resulting from the application a fitness proportional selection operator to the population \( \mu_v \). The operator \( E(\cdot \times f/E(f)) \) is the expectation operator \( E_{\nu}(\cdot) \) relatively to the distribution \( \nu \). (6) can be rewritten using the operator \( E_{\nu}(\cdot) \) as

\[
v'_i = E_{\nu}(x_i) - v_i.
\]

The intuitive interpretation of the above equation is that \( v \) is moved towards vertices with high fitness.

*Boltzmann selection* We come closer to PBIL in changing the criterion and optimizing \( J = \log E(e^{\beta f}) \), with \( \beta > 0 \), instead of \( J = \log(f) \). The dynamical system can still be written like (7) but the distribution resulting from selection is obtained through the transformation

\[
\mu_v(x) \rightarrow \nu(x) = \frac{e^{\beta f(x)} \mu_v(x)}{E(e^{\beta f})}.
\]

We write the dynamical system as \( v' = \varphi_\beta(v) \) and we can prove, using a Lagrange technique, that if \( v \in \Omega \) then

\[
\lim_{\beta \to \infty} \varphi_\beta(v) = \left( \sum_{x \in U} \frac{\mu_v(x)}{\sum_{x \in U} \mu_v(x)} x \right) - v.
\]
If $U = \{x^*\}$, then the only singular point of the ideal system is $x^*$ itself, which motivates the search for a schedule $\beta : \mathbb{N} \rightarrow \mathbb{R}^+$, with $\beta$ increasing, $\beta(0) = 0$ and $\beta(\infty) = \infty$, in order to insure convergence to the global optimum $x^*$. Such a schedule (or annealing) has been proposed for genetic algorithms by Cerf [5].

**Stochastic approximation** At each iteration, $N$ individuals $(x^i), i = 1, \ldots, N$, are sampled from $\mu_v$ which results in the empirical distribution

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \delta_{x^i},$$

which is then re-weighted through Boltzmann selection, yielding the new empirical distribution

$$\tilde{\nu} = \frac{N}{\sum_{j=1}^{N} e^{\beta f(x^j)}} \sum_{i=1}^{N} e^{\beta f(x^i)} \delta_{x^i}.$$  

If we consider the limit case $\beta \rightarrow \infty$ then $\tilde{\nu}$ is uniform over the best individuals in the finite population $(x^i), i = 1, \ldots, N$. If we select only one such individual $x^*$ then $\nu$ is updated with the discrete time recurrent equation

$$\nu(t+1) = \nu(t) + \alpha(x^* - \nu),$$

where $0 < \alpha < 1$. This is very close to the basic PBIL algorithm (in [1] Baluja and Caruana use the best two vectors instead of just the best one).

### 4 Stability Analysis of Vertices

#### 4.1 Introduction and Notations

In this section, we analyze the stability of points of $S$ or vertices of the box $\Delta$ which are, by construction, singular points of (2). To do so, we study the sign of a Liapunov function [7] in the neighborhood of a vertex. We implicitly use an induced topology on $\Omega$.

Let $y$ be a point of $S$. The Liapunov function $h : \Omega \rightarrow \mathbb{R}$ is simply $h(v) = J(\nu_v) - f(y)$. Since $y$ is in $S$, we have $\mu_y = \delta_y$. Then $J(\mu_y) = f(y)$ and $h(y) = 0$. The time derivative of $h$ is

$$h' = \sum_{i=1}^{n} \frac{\partial J}{\partial v_i} v'_i = \sum_{i=1}^{n} v_i (1 - v_i) \left( \frac{\partial J}{\partial v_i} \right)^2.$$

$h'$ vanishes on $y$ since $y$ is a vertex. If the gradient of $J$ does not vanish on $y$ \(^1\), we can find a neighborhood $W_1 \subset \Omega$ of $y$ on which $h' > 0$. We will be able to conclude in two cases:

\(^1\) Otherwise, in case of plateau or when $(f \circ \sigma_i)(y) = f(y)$ for all $i \in L$, we only get simple stability.
1. If we can find another neighborhood $W_2 \subset \Omega$ of $y$ on which $h < 0$ then $y$ is asymptotically stable;
2. If we can find another neighborhood $W_2 \subset \Omega$ of $y$ on which $h$ takes both signs then $y$ is unstable.

In order to study the sign of $h$, we develop $J$ in the neighborhood of $y$ until we reach an order at which we can conclude. We first make a change of variables to handle only positive coordinates in the neighborhood of $0$. Let $v = (v_1, \ldots, v_n)$ be in the neighborhood of $y$ in $\Omega$. We define the positive vector $w = (w_1, \ldots, w_n)$ such that if $y_i = 0$ then $v_i = w_i$, and if $y_i = 1$ then $v_i = 1 - w_i$. For each $x \in S$, we define the set of integers $I(x)$ such that the corresponding components of $x$ and $y$ are equal, that is $I(x) = \{i \in L : x_i = y_i\}$ where $L = \{1, \ldots, n\}$. We also define its complementary $J(x)$. For each integer $i$ in $I(x)$ we have $\phi(x_i, v_i) = 1 - w_i$, and for each integer $i$ in $J(x)$ we have $\phi(x_i, v_i) = w_i$. With these notations, we have

$$\mu_v(x) = \prod_{i \in I(x)} (1 - w_i) \times \prod_{i \in J(x)} w_i.$$  

We introduce the family of bit reversal transformations, $\sigma_i : S \rightarrow S, i \in L$, such that if $z = \sigma_i(y)$ then $z_i = 1 - y_i$ and $z_j = y_j$ for $j \neq i$. We use the fact that those transformations commute to write any $x \in S$ as the product $(\prod_{i \in J(x)} \sigma_i)(y)$. Summations over $S$ can then be replaced by summations over subsets of $L$. We finally introduce two additional notations to facilitate reading, $w_K = \prod_{i \in K} w_i$ and $f_K = (f \circ \prod_{i \in K} \sigma_i)(y)$, where $K \subseteq L$.

4.2 Criterion Expansion

With the above notations, the criterion can be written

$$J(\mu_v) = \sum_{J \subseteq L} f_J w_J \prod_{i \in L \setminus J} (1 - w_i).$$

We expand the inner product using the symmetrical polynomials

$$\chi_i(I) = \sum_{\substack{J \subseteq I \\ |J| = i}} w_J,$$

and obtain

$$J(\mu_v) = \sum_{J \subseteq L} f_J w_J \sum_{i=0}^{n-|J|} (-1)^i \chi_i(L \setminus J),$$

$$\chi_i(I) = \sum_{\substack{J \subseteq I \\ |J| = i}} w_J,$$
where \(|J|\) is the cardinal of \(J\) which we use to split the first sum. Then,

\[
J(\mu_v) = \sum_{k=0}^{n} \sum_{l=0}^{n-k} (-1)^i \sum_{J \subseteq L \atop |J| = k} f_J w_J \chi_i(L \setminus J)
\]

\[
= \sum_{k=0}^{n} \sum_{l=k}^{n} (-1)^{l-k} \sum_{J \subseteq L \atop |J| = k} f_J w_J \chi_{l-k}(L \setminus J) \quad \text{(where } l = i + k)\]

\[
= \sum_{l=0}^{n} (-1)^l \sum_{J \subseteq L \atop |J| \leq l} (-1)^{|J|} f_J w_J \chi_{l-|J|}(L \setminus J). \quad \text{(merge)}
\]

We isolate the product of components and see that

\[
w_J \chi_{l-|J|}(L \setminus J) = \sum_{K \subseteq L \atop |K| = l} w_K.
\]

We make the substitution into the criterion and get

\[
J(\mu_v) = \sum_{l=0}^{n} (-1)^l \sum_{J \subseteq L \atop |J| \leq l} \sum_{K \subseteq L \atop |K| = l} (-1)^{|J|} f_J w_K
\]

\[
= \sum_{l=0}^{n} (-1)^l \sum_{K \subseteq L \atop |K| = l} w_K \sum_{J \subseteq K} (-1)^{|J|} f_J \quad \text{(exchange)}
\]

\[
= \sum_{K \subseteq L} \alpha_K w_K, \quad \text{(merge)}
\]

where \(\alpha_K = (-1)^{|K|} \sum_{J \subseteq K} (-1)^{|J|} f_J\).

### 4.3 Sign of the Liapunov Function

**First order** We consider first order terms in the expansion of \(h\) which we write

\[
\sum_{K \subseteq L \atop |K| = 1} w_K (f_K - f(y)).
\]

If there exists a subset \(K\) such that \(|K| = 1\) and \(f_K > f(y)\) then \(y\) is unstable since \(h\) and \(h'\) can take values of the same sign the neighborhood of 0.

We now consider terms of order \(l \geq 2\) in the expansion of \(h,\)

\[
\sum_{K \subseteq L \atop |K| = l} \alpha_K w_K.
\]
We say that a subset $K$ of $L$ has the property $A_i$ if for each subset $J \subseteq K$ such that $|J| \leq i$, $f_J = f(y)$. We split the first sum between subsets having $A_1$ and those having $\neg A_1$. For subsets $K$ having $\neg A_1$, we can find a subset $J \subseteq K$ such that $|J| = 1$ and $f_J \neq f(y)$, which we reduce to $f_J < f(y)$ since we have already seen that the reverse relation led to instability. Then $J$ must appear in the first order terms and since $w_K$ is an $O(w_J)$ we will neglect it and simply write $O(w_J, |J| = 1, f_J < f(y))$. The expansion can now be written

$$
\sum_{K \subseteq L} \sum_{|K| = 1, \neg A_1} w_K (f_K - f(y)) + O(w_K, |K| = 1, f_K < f(y)) + \sum_{K \subseteq L} \sum_{|K| \geq 2, A_1} \alpha_K w_K.
$$

Higher orders Assume that the expansion of $h$ can be written

$$
\sum_{l=1}^p \left( \sum_{K \subseteq L} \sum_{|K| = l, A_{l-1} \wedge \neg A_l} w_K (f_K - f(y)) + O(w_K, |K| = l, f_K < f(y)) \right) + \sum_{K \subseteq L} \sum_{|K| \geq p+1, \neg A_p} \alpha_K w_K
$$

We isolate subsets for which $|K| = p + 1$ in the remainder and simplify the corresponding term. Since those subsets have the property $A_p$, it is easy to show that $\alpha_K = f_K - f(y)$, which gives the first term of the expansion for the order $p + 1$. Just as we have done for the first order, we split the remainder of order at least $p + 2$ between subsets having $A_{p+1}$ and those having $\neg A_{p+1}$. The latter can be neglected since they contain a subset which must appear in the first term. We have thus completed the expansion of $h$ for the order $p + 1$. We have to decide when to stop the expansion:

1. if there exists a subset $K$ with $|K| = p$ and $A_{p-1} \wedge \neg A_p$ such that $f_K > f(y)$ then the vertex is unstable;
2. if there exists no subset $K$ with $|K| = p + 1$ and $A_p$ then the vertex is asymptotically stable.

5 Other Stability Issues

Singular points of the criterion $J$ are such that $\partial J/\partial v_i = 0$ for all $i \in L$. Let us consider the derivative $D \varphi$ of the vector field $\varphi$, which is the matrix of general element $\partial \varphi_i/\partial v_j$. Since $J(v_i)$ is polynomial of degree 1 in each $v_i$, diagonal entries in $D \varphi$ are equal to zero at singular points of $J$. The trace of $D \varphi$ is thus equal to zero. The trace of $D \varphi$ is also equal to the sum of its eigenvalues, which implies that if $D \varphi$ does not vanish, it has both positive and negative eigenvalues, making any singular point of $J$ an unstable point. We have not dealt with the issue of the existence of such singular points, in particular inside the hypercube.

We have neither covered the case of compound singular points such that $\partial J/\partial v_i = 0$ for some $i$ and $v_i = 0$ or $v_i = 1$ for others nor the case of vertices which are at the same time singular points of the criterion $J$. 
6 Conclusion

We have presented a statistical framework for combinatorial optimisation over fixed-length binary strings and shown that both selection and reinforcement learning algorithms can be derived from gradient dynamical systems acting on Bernoulli probability vectors. It should be clear that such algorithms perform as well as hill climbing, since they can only converge to locally optimal solutions. We can state that from the stability analysis of vertices, which deals with the local behavior of the system. It remains to find a criterion, geometric or algebraic, to decide whether the system will converge to the global optimum of the function. This should ultimately be related to the regularity of the function and the notion of fitness landscape [12, 11].

References

Ant Colony Optimization for the Total Weighted Tardiness Problem

Matthijs den Besten\textsuperscript{1,2}, Thomas Stützle\textsuperscript{2,3}, and Marco Dorigo\textsuperscript{2}

\textsuperscript{1} University of Amsterdam, Department of Computer Science
Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands
\textsuperscript{2} Université Libre de Bruxelles, IRIDIA,
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
\textsuperscript{3} Darmstadt University of Technology, Intellectics Group,
Alexanderstr. 10, 64283 Darmstadt, Germany

Abstract. In this article we present an application of the Ant Colony Optimization (ACO) metaheuristic to the single machine total weighted tardiness problem. First, we briefly discuss the constructive phase of ACO in which a colony of artificial ants generates a set of feasible solutions. Then, we introduce some simple but very effective local search. Last, we combine the constructive phase with local search obtaining a novel ACO algorithm that uses a heterogeneous colony of ants and is highly effective in finding the best-known solutions on all instances of a widely used set of benchmark problems.

1 Introduction

In this paper we study the single machine total weighted tardiness problem (SMTWTP) [21], a scheduling problem that is known to be \mathcal{NP}-hard [16] and for which instances with more than 50 jobs can often not be solved to optimality with state-of-the-art branch & bound algorithms [1, 5].

In the SMTWTP \( n \) jobs have to be sequentially processed on a single machine. Each job \( j \) has a processing time \( p_j \), a weight \( w_j \), and a due date \( d_j \) associated, and the jobs become available for processing at time zero. The tardiness of a job \( j \) is defined as \( T_j = \max\{0, C_j - d_j\} \), where \( C_j \) is the completion time of job \( j \) in the current job sequence. The goal is to find a job sequence which minimizes the sum of the weighted tardiness given by \( \sum_{j=1}^{n} w_j \cdot T_j \).

Because the SMTWTP is \mathcal{NP}-hard, several heuristic methods have been proposed for its solution. These include construction heuristics like the Earliest Due Date or the Apparent Urgency heuristics (see [22] for an overview) and metaheuristics like Simulated Annealing [18, 22], Tabu Search and Genetic Algorithms [5].

In this paper we present the application of the Ant Colony Optimization (ACO) metaheuristic [8, 9] to the SMTWTP. ACO has been introduced by Dorigo and colleagues as a new optimization paradigm which is inspired by the trail following behavior of real ant colonies. Algorithmic implementations of this metaheuristic have shown very promising results for the well known Traveling Salesman Problem [10, 25] and are currently among the best available algorithms for other hard problems like the Quadratic Assignment Problem [13, 17, 26], the Sequential Ordering Problem [11], Vehicle Routing Problems [12], and routing problems in highly dynamic environments [7].
procedure ACO metaheuristic for static combinatorial problems
    Set parameters, initialize pheromone trails
    while (termination condition not met) do
        ConstructSolutions
        ApplyLocalSearch % optional
        UpdateTrails
    end

Fig. 1. Algorithmic skeleton for the ACO metaheuristic applied to static problems.

The paper is structured as follows. In Section 2 we introduce ACO and Ant Colony System, the particular ACO algorithm applied here. Section 3 presents the local search algorithm we applied and Section 4 investigates the proposed ACO algorithm and gives final performance results. We conclude with some final remarks in Section 5.

2 Ant Colony Optimization

In ACO algorithms a colony of (artificial) ants iteratively constructs solutions to the problem under consideration using (artificial) pheromone trails which are associated with appropriately defined solution components and heuristic information. The ants only communicate indirectly by modifying the pheromone trails during the algorithm’s execution. Because the constructed solutions need not be locally optimal with respect to small modifications, in many of the best performing ACO algorithms the ants additionally improve their solutions by applying a local search algorithm. Hence, most ACO algorithms for static combinatorial optimization problems\(^4\) follow the particular algorithmic scheme given in Figure 1.

2.1 Solution construction

We apply Ant Colony System (ACS) [10], a particular ACO algorithm, to the SMTWTP. A feasible solution for the SMTWTP, also called a sequence, consists of a permutation of the jobs. When applied to the SMTWTP, each ant starts with an empty sequence and then iteratively appends an unscheduled job to the partial sequence constructed so far. In ACS jobs are scheduled as follows.

- With probability \(q\), the unscheduled job \(j\) that maximizes \(\tau_{ij}(t) \cdot \eta_{ij}^\beta\) is put in the current sequence at position \(i\). Here, \(\tau_{ij}(t)\) is the pheromone trail associated to the assignment of job \(j\) to position \(i\), \(t\) indicates the dependence of the pheromone trail on the time, \(\eta_{ij}\) is the heuristic desirability of assigning job \(j\) to position \(i\), and \(\beta\) is a parameter which determines the influence of the heuristic information.

\(^4\) We call static those problems whose topology and costs do not change while they are being solved. The ACO metaheuristic can be applied also to dynamic problems in which topology and costs can change while solutions are built. An example is the AntNet algorithm that was applied to a routing problem for Internet-like networks [7].
With probability $1 - q$, job $j$ is chosen randomly with a probability given by

$$p_{ij} = \begin{cases} \frac{\tau_{ij}(t) \cdot \eta_{ij}^q}{\sum_{i \text{ not scheduled}} \tau_{ii}(t) \cdot \eta_{ii}^q} & \text{if job } j \text{ is not yet scheduled} \\ 0 & \text{otherwise} \end{cases}$$

Hence, with a probability $q$ the ant does the best decision as indicated by the pheromone trails and the heuristic information (exploitation), while with probability $1 - q$ it performs a biased exploration. The three following standard heuristics have been used to compute the heuristic information $\eta_{ij}$ used by the ants:

- Earliest Due Date (EDD). This heuristic puts the jobs in non-decreasing order of the due dates $d_j$. In this case $\eta_{ij} = 1/d_j$.
- Modified Due Date (MDD). This heuristic puts the jobs in non-decreasing order of the modified due dates $\text{mdd}_j$ [2] given by $\text{mdd}_j = \max\{C + p_j, d_j\}$, where $C$ is the sum of the processing times of the already sequenced jobs. In this case $\eta_{ij} = 1/\text{mdd}_j$.
- Apparent Urgency (AU). This heuristic puts the jobs in non-decreasing order of the apparent urgency [20], given by $\text{au}_j = (w_j/p_j) \cdot \exp(-\max\{d_j - C_j, 0\}) / k\bar{p}$. Here, $\bar{p}$ is the average processing time of the remaining jobs, $k$ is a parameter set as proposed in [22], and $\eta_{ij} = 1/\text{au}_j$.

### 2.2 Pheromone update

In ACS, two forms of pheromone update are applied. In the delayed pheromone trail update after each iteration pheromone trail is added to the components of the global-best solution, that is, the best sequence found so far. If in the global-best solution at iteration $t$ job $j$ is put on position $i$, then $\tau_{ij}(t + 1) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta \tau_{ij}(t)$, where $\rho, 0 < \rho \leq 1$, is a parameter representing the pheromone evaporation and $\Delta \tau_{ij}(t) = 1/T^*$, where $T^*$ is the total weighted tardiness of the global-best solution.

Additionally, ACS applies a step-by-step pheromone update rule immediately after an ant has added a new job to the partial sequence. Here, pheromone trails are modified by the update rule $\tau_{ij} = (1 - \xi) \cdot \tau_{ij} + \xi \cdot \tau_0$, where $\xi, 0 < \xi \leq 1$, and $\tau_0$, which is chosen as a small value, are two parameters. The effect of the local updating rule is to make the decision of putting job $j$ on position $i$ less desirable for the other ants so that the exploration of different sequences is favored.

### 2.3 Other ACO approaches to scheduling problems

An application of ACO to the unweighted single machine total tardiness problem was presented in [2]. Although for this problem heuristic approaches appear not to be very interesting, because the total (unweighted) tardiness problem can be rather efficiently solved by enumerative methods [6], we verified that our approach substantially outperforms this early application. In the literature only few other applications of ACO algorithms to scheduling problems are reported. In [3] an application of Ant System, the first ACO algorithm, to the Job Shop Scheduling problem is reported. Yet, this approach obtained relatively poor computational results, probably because no local search was used to improve solutions. Much better performance is reported for the application of MAX-MIN Ant System [25, 27] to the permutation Flow Shop Problem [24].
3 Local search for the SMTWTP

Local search for the SMTWTP starts from some initial sequence and repeatedly tries to improve the current sequence by replacing it with neighboring solutions. If in the neighborhood of the current sequence $\pi$ a better sequence is found, it replaces $\pi$ and the local search is continued from the new solution. The simplest local search algorithm, iterative descent, repeatedly applies these steps until no better neighboring sequence can be found and stops at the first local minimum encountered. Critical for the performance of the local search algorithm is the neighborhood structure chosen. For the SMTWTP we considered the following two neighborhood structures:

1. exchanges of jobs placed at the $i$th and the $j$th position, $i \neq j$ (interchange)
2. removal of the job at the $i$th position and insertion in the $j$th position (insert)

The implementation of these local search algorithms was sped-up using the techniques described in [4]. To achieve further improvements of the solution quality, we considered the concatenation of the iterative descent algorithms using the two different neighborhoods. This may be reasonable because a local optimum with respect to one neighborhood structure need not be a local optimum for the other one. In fact, the recent Variable Neighborhood Search metaheuristic [19] systematically applies the idea of changing neighborhoods in the search. The concatenation of two local search algorithms has also been applied in [23] for the permutation Flow Shop Problem. The concatenated local search algorithms, called Variable Neighborhood Descent (VND) in the following, will be denoted as interchange+insert and insert+interchange, depending on which neighborhood is searched first.

To evaluate local search we used a benchmark set of randomly generated instances, available via ORLIB at http://www.ms.ic.ac.uk/info.html. The benchmark set comprises instances with 40, 50, and 100 jobs. For the 40 and 50 job instances the optimal solutions are known, while for the 100 job instances only the best known solutions are given. The instances are generated randomly by drawing the processing time $p_j$ for each job $j$ randomly according to a uniform distribution of integers between 1 and 100 and assigning it a weight $w_j$ randomly drawn from a uniform distribution over the integers between 1 and 10. The due dates are randomly drawn integers from the interval $[(1-TF-RDD/2)\cdot\sum p_i, (1-TF+RDD/2)\cdot\sum p_i]$, where TF, the tardiness factor, and RDD, the relative due date, are two parameters. There are five instances for each pair of TF and RDD from the set {0.2, 0.4, 0.6, 0.8, 1.0}. This makes three sets of 125 instances each. The tardiness factor and the relative due dates determine critically the difficulty of solving the instances. For example, we found that most of the instances with TF = 0.2 are solved after one single application of the local search procedure, while for larger TF, the instances were much harder to solve. All the experiments were run using a 450MHz Pentium III PC with 256 MB RAM. Programs were written in C++ and run under Red Hat Linux 6.1.

In Table 1 we give computational results for the three construction heuristics EDD, MDD, and AU, and for the proposed local search algorithms when starting from initial solutions generated by the three construction heuristics without local search. As indicated before, some of the instances are very easily solved. This can be noted in Table 1 observing that a large number of best known solutions are found by the construction
Table 1. Comparison of the local search effectiveness for the SMTWTP. Results on the 100 job instances without local search and using the interchange, the insert, the interchange+insert, and the insert+interchange local search. We give the average percentage deviation from the best known solutions ($\Delta_{avg}$), the number of best-known solutions found ($n_{opt}$), and the average CPU time in seconds ($t_{avg}$) averaged over the 125 benchmark instances.

<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>no local search</th>
<th>interchange</th>
<th>insert</th>
<th>inter+insert</th>
<th>insert+inter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\Delta_{avg}$</td>
<td>$n_{opt}$</td>
<td>$t_{avg}$</td>
<td>$\Delta_{avg}$</td>
<td>$n_{opt}$</td>
<td>$t_{avg}$</td>
</tr>
<tr>
<td>EDD</td>
<td>0.001</td>
<td>0.62</td>
<td>33</td>
<td>0.64</td>
<td>0.24</td>
<td>46</td>
</tr>
<tr>
<td>MDD</td>
<td>0.002</td>
<td>0.65</td>
<td>38</td>
<td>0.078</td>
<td>1.31</td>
<td>36</td>
</tr>
<tr>
<td>AU</td>
<td>0.008</td>
<td>0.92</td>
<td>28</td>
<td>0.040</td>
<td>0.56</td>
<td>42</td>
</tr>
</tbody>
</table>

heuristics without local search. Yet, when averaged over the whole benchmark set, the construction heuristics alone perform rather poorly and the application of a local search algorithm increases very strongly the average solution quality with the VND showing a significant improvement over the single neighborhood local search. Table 1 also shows that the initial solutions generated by AU are best combined with insert local search, while initial solutions from the EDD and MDD heuristics are best combined with interchange local search. Regarding computation time, the interchange local search is significantly faster than the insert local search. Yet, interestingly, the computation time of the VND is only marginally higher than the one taken by its first local search.

4 Combining ACS with local search

In ACO, as in many other metaheuristics, the performance critically depends on the appropriate combination of local search with the solution construction mechanism. We performed therefore a detailed analysis of the effectiveness of the coupling of ACS with the four local search algorithms on 10 hard SMTWTP instances from the 100 job benchmark set. For the experiments we used the following default parameter settings: 20 ants, $\beta = 2$, $\xi = 0.1$, $\rho = 0.1$, $q = 0.9$, and $\tau_0 = 1/(n \cdot T_{MDD})$, where $T_{MDD}$ is the total weighted tardiness of the solution generated by the MDD heuristic and $n$ is the number of jobs.\footnote{Analogous parameter settings were found to yield good performance in earlier studies with ACS on the Traveling Salesman Problem [10] and the Sequential Ordering Problem [11].}

4.1 Configuration of ACS for the SMTWTP

In the following we present the results using run-time distributions (RTDs) which give the cumulative empirically observed probability of finding an optimal solution as a function of the CPU time [15, 25]. Here, for each instance 25 runs have been performed using the MDD heuristic information (the same experiments were also run using the AU heuristic). In Figure 2 are given the run-time distributions for the 14th (left plot in Figure 2) and the 67th instance (right plot in Figure 2) of the 125 instances of the 100 job benchmark set (on the other eight instances the relative performance of the local search algorithms was similar). From the RTDs we could make the following interesting observations. First, the VND algorithms interchange+insert and insert+interchange performed typically better than a single local search in the first neighborhood (interchange
Fig. 2. Comparison for combinations of the different local search algorithms with ACS based on the empirical cumulative probability distribution for finding the best-known solution measured over 25 independent runs. We give the run-time distributions for reaching the best known solution value on the 14th (left) and the 67th instance (right) of the 125 available 100 job instances.

and insert, respectively). Hence, the higher solution quality obtained by VND clearly compensates the small run-time disadvantage. Second, which of the two local search algorithms (interchange+insert or insert+interchange) should be run first is strongly instance dependent. For example, on the 14th instance the interchange+insert VND clearly dominates the insert+interchange VND, as noted from the fact that for all computation times it achieves a much higher probability of finding optimal solutions. On the other side, on the 67th instance the interchange+insert VND achieves larger solution probabilities only for short computation times; for longer run-times the ACS using insert+interchange VND is significantly better.

Having noticed the instance dependence of the appropriate choice for the local search algorithm, we addressed this problem considering an ACS algorithm using a heterogeneous colony of ants, in which half of the ants apply interchange+insert and the other half apply insert+interchange VND. Using the heterogeneous colony, we obtained the RTDs indicated with half/half in Figure 2 and, as can be observed, the heterogeneous colony for all CPU-times obtains a larger solution probability than the worse of the two homogeneous colonies using VND; interestingly, on the 67th instance, the heterogeneous ant colony performed even better than the best homogeneous colony. We have observed this phenomenon also on some of the other instances.

When closer examining the solution construction, we noted that if a job $j$ is not put on the position $i$ for which $\tau_{ij}(t) \cdot \eta_{ij}^3$ is maximal, often it is put at one of the last positions in the sequence. Because the absolute position of a job is important, we introduced additional candidate lists [24, 10]. In particular, the candidate set is defined dynamically during the solution construction and comprises the first still not sequenced cand jobs of the global-best solution $T^*$. The next job is then chosen among those of the candidate set in the case of a probabilistic choice according to Equation 1. We found that with cand = 20 a reasonably good performance is obtained.

Additionally, we analyzed the number of ants which should be used in ACS to achieve a good trade-off between the probability of finding best-known solutions and the necessary run-time to do so. In general, best behavior has been obtained with a num-
Fig. 3. Comparison on the influence of the heuristic information on ACS results. We give the RTDs for reaching the best known solution value on the 14th (left) and the 67th instance (right) of the 125 available 100 job instances.

Number of ants between 5 and 10. For a too small number of ants, some optimal solutions are found rather fast, but with increasing run-time often ACS with a larger number of ants achieves higher probabilities of finding optimal solutions. For the following experiments we used 10 ants.

As a next step, we analyzed the influence of the heuristic information on the final results. An important difference between static and dynamic heuristic information is that static information does not depend on the partial solution constructed by the ant while dynamic information does. When using static information, like the EDD-based heuristic information, the values of $\tau_{ij}(t) \cdot \eta_{ij}$ can be precomputed and need to be only occasionally updated. With dynamic heuristic information this is typically not possible and in this case the solution construction incurs a higher computational cost. Yet, this may be compensated by a higher accurateness of the dynamic heuristic information.

In Figure 3 we give RTDs for the two previously examined instances using the three different heuristics and not using heuristic information at all. The observed RTDs show that it is instance dependent which heuristic information yields best results. For example, while on the 14th instance the AU-based heuristic information showed to be even worse than not using any heuristic information at all, on the 67th instance the AU-based heuristic information performed best together with MDD. In general, we found that the advantage of the AU-based heuristic information increased with increasing range of due dates (RDD) and generally performed best for larger RDD. Only on instances with small RDD the MDD-based heuristic information gave significantly better results. Therefore, in ACS we use MDD-based heuristic information if $\text{RDD} \leq 0.3$ and otherwise the AU-based heuristic information.

4.2 Final Results

In final experiments we tested ACS on the SMTWTP benchmark instances from ORLIB. Because many of the instances could already be solved by one single local search we felt that the only real challenge would be to find the best known solutions on all instances. To account for this fact, we run ACS at least 25 times on each instance and we set the maximal CPU time to 1200 seconds, so that ACS could find for every instance...
Fig. 4. Final study of the performance of ACS on the 100 job instances. On the left is given the average computation time to find optimal solutions in dependence of the relative due date (RDD) and the tardiness factor (TF), while on the right is given the cumulative number of instances solved in dependence of the average computation time (measured over 25 runs on each single instance) to solve them.

Table 2. We give some basic statistics on the distribution of the average computation times to solve instances of the three problem sets. We indicate the number of jobs, the average time (averaged over the 125 instances) to solve the benchmark set ($t_{avg}$) and the standard deviation ($\sigma_t$), the average time to solve the easiest and the hardest instance ($t_{min}$ and $t_{max}$, respectively), and the quantiles of the average time to solve a given percentage of the instances. $Q_x$ indicates the average time to solve $x\%$ of the benchmark instances.

<table>
<thead>
<tr>
<th>No. jobs</th>
<th>$t_{avg}$</th>
<th>$\sigma_t$</th>
<th>$t_{min}$</th>
<th>$t_{max}$</th>
<th>$Q_{10}$</th>
<th>$Q_{25}$</th>
<th>$Q_{50}$</th>
<th>$Q_{75}$</th>
<th>$Q_{90}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.088</td>
<td>0.021</td>
<td>0.004</td>
<td>1.72</td>
<td>0.004</td>
<td>0.008</td>
<td>0.013</td>
<td>0.122</td>
<td>0.215</td>
</tr>
<tr>
<td>50</td>
<td>0.32</td>
<td>0.869</td>
<td>0.006</td>
<td>10.74</td>
<td>0.007</td>
<td>0.016</td>
<td>0.031</td>
<td>0.353</td>
<td>0.606</td>
</tr>
<tr>
<td>100</td>
<td>6.99</td>
<td>7.019</td>
<td>0.018</td>
<td>86.26</td>
<td>0.028</td>
<td>0.070</td>
<td>3.190</td>
<td>7.499</td>
<td>19.30</td>
</tr>
</tbody>
</table>

the best known solutions in each single run. In Figure 4 and Table 2 we give indicative statistics on the average time taken by ACS to solve the instances.

We found that a large part of the benchmark set was solved in short computation time. For example, on the 100 job instances 113 out of 125 instances could be solved in less than 20 seconds on average. Only a small part of the instances, most of them with a TF around 0.6 and large RDD, took somewhat more time. A closer examination showed that the hardest instances occur at a point where roughly half of the jobs are late and half of the jobs non-late. In part, the increase of the computation time on such instances is also due to the fact that the speed-up techniques applied in the local search are not anymore as effective as on instances with many late or non-late jobs.

ACS appears to perform significantly better than most other previously proposed algorithms for the SMTWTP. For example, ACS finds always the best-known solutions for the 100 job instances, while the best performing Tabu Search algorithm in [5] could only find 103 of the best-known solutions at that time (some of those instances were later improved). The only algorithm reaching a similar performance as ACS appears to be iterated dynasearch (ID) [4]. ID found on average 123.2 times the best-known solutions (the same best-known solutions to which the Tabu Search was compared). Regarding the computation time, ACS may take slightly larger computation time. However, with ACS we could solve all benchmark instances in every single run.
5 Conclusions

In this paper we have presented ACS-SMTWTP, an effective ACO algorithm for the single machine total weighted tardiness problem. Our algorithm could find, for all known benchmark instances available in the ORLIB, the optimal or best-known solutions within reasonable computation times. Moreover, it was able to find the optimal solution in every run if enough computation time was given. Our analysis also showed that the hardest instances occur at specific values of the tardiness factor and of the relative due date, the parameters which were used in the randomized instance generation suggesting the existence of a kind of phase transition for the SMTWTP like those observed in many other combinatorial optimization problems [14]. There are several important reasons for the very good performance of the ACO approach. Among them are that we use a very effective local search algorithm and additionally a heterogeneous colony of ants which increased the robustness of the algorithm.

In the future we will test our ACS algorithm on larger SMTWTP instances and will extend our successful algorithm to a wider range of single-machine scheduling problems. Additionally, our results using a heterogeneous colony of ants suggest that a similar approach may also yield improvements for ACO algorithms in other applications where a strong instance dependence of the best algorithm configuration has been noted. Similarly, a self-tuning ACO algorithm which uses search feedback to adjust its configuration to the particular instance currently being solved may be interesting.

References


Adaptive Fitness Functions
for the Satisfiability Problem

Jens Gottlieb and Nico Voss

Department of Computer Science, Technical University of Clausthal
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
{gottlieb, nvoss}@informatik.tu-clausthal.de

Abstract. Adaptive fitness functions have led to very successful evolutionary algorithms for the satisfiability problem. Although comparisons are available for benchmarks, a deeper understanding of the effects of adaptation is desirable. Therefore, we compare three approaches based on adapting weights. The dynamics of these weights motivate the use of decay factors, which significantly improve the success rate for two adaptation schemes. The most successful technique can be further improved by accelerating the adaptation process concerning difficult clauses.

1 Introduction

The satisfiability problem (SAT) is based on a set of boolean variables $x_1, \ldots, x_n$ and a set of clauses $c_1, \ldots, c_m$. A clause is a disjunction of literals, and a literal is a variable or its negation. Clauses can be interpreted as functions $c_i : \{0, 1\}^n \rightarrow \{0, 1\}$, and the goal is to determine a variable assignment $x \in \{0, 1\}^n$ satisfying all clauses, i.e. $c_1(x) \land \cdots \land c_m(x) = 1$. The class $k$-SAT contains all SAT instances where each clause contains exactly $k$ distinct literals. While 2-SAT is solvable in polynomial time, SAT and $k$-SAT for $k \geq 3$ are NP-complete [GJ79]. Due to its NP-completeness in the general case, several heuristic methods have been proposed for SAT, e.g. local search [Fra97, SLM92] and evolutionary algorithms (EAs). While classical genetic algorithms have difficulties to solve even instances of small size, recent enhancements like adaptive fitness functions, problem-specific variation operators and hybridization with local search [BEV98, GV98b, MR99] yielded a remarkable performance of EAs for hard 3-SAT instances. As adaptive fitness functions have been investigated in different EA configurations [BEV98, GV98b], it is difficult to assess the actual benefits of the proposed adaptation techniques. Therefore, we examine the effects of distinct adaptation mechanisms – including new variants – for a fixed EA setup.

Section 2 compares adaptive fitness functions, which are analysed w.r.t. decay factors in section 3. Accelerated adaptation is investigated in section 4, followed by a comparison with other EAs in section 5. Conclusions are given in section 6.

2 Adaptive Fitness Functions for SAT

We consider three basic adaptive fitness functions throughout this paper. These functions differ in the structure of the adapted fitness function $f$ and the adap-
tation mechanism employed. While the first approach (SAW) is based on clause weights, the remaining (AW1, AW2) use variable weights. In general, the adaptation takes place after a fixed number of fitness evaluations and exploits information concerning the fittest individual $x^*$ in the current population.

### 2.1 Three Adaptation Mechanisms

The mechanism of *stepwise adaptation of weights* (SAW) was introduced by Eiben et al. and applied successfully to 3-SAT [BEV98, EH97]. It is based on weights $w_i \in \mathbb{R}_+$ for the clauses $c_i$. The fitness function

$$f_1(x) = \sum_{i=1}^{m} w_i c_i(x)$$

is adapted by modifying the weights appropriately. In the beginning all weights are initialized by $w_i = 1$, and later the weights are adjusted according to

$$w_i \leftarrow w_i + 1 - c_i(x^*) \quad \text{for } i \in \{1, \ldots, m\} \quad \text{(SAW)}.$$

The weights represent a memory which helps to identify those clauses that have been difficult to satisfy in the preceding search process. A high weight $w_i$ indicates a great difficulty of clause $c_i$ and leads to a higher fitness of those solutions that satisfy $c_i$. Thus, the weights guide the search towards solutions satisfying the currently most difficult clauses.

Suggested in our previous study concerning 3-SAT [GV98b], refining functions form an alternative to guide the search by the adaptive fitness function

$$f_2(x) = \sum_{i=1}^{m} c_i(x) + \alpha \cdot r(x).$$

The fitness is calculated as a weighted sum of the number of satisfied clauses and the value of the refining function $r$. The factor $\alpha$ determines the influence of $r$, which is used to incorporate additional heuristic information into the fitness function. We analysed several refining functions but here we concentrate on

$$r(x) = \frac{1}{2} \left( 1 + \frac{\sum_{j=1}^{n} K(x_j) \cdot v_j}{1 + \sum_{j=1}^{n} |v_j|} \right)$$

with $K(1) = 1$ and $K(0) = -1$. This function is based on weights $v_j \in \mathbb{R}$ for the variables $x_j$, where positive weights indicate that the corresponding variables are favoured to be 1, while negative weights express a preference to 0. The weights are initially set to 0 and then adapted such that high absolute weights indicate high preferences. Thus, the weights determine the current search direction of the EA. We consider two adaptation schemes: the first scheme is

$$v_j \leftarrow v_j - K(x_j^*) \quad \text{for } j \in \{1, \ldots, n\} \quad \text{(AW1)},$$
which reflects a moderate adjustment of the weights towards the complement of the current best individual \(x^*\). The idea of this adaptation is to escape from local optima. While AW1 is problem-independent, the second scheme

\[
v_j \leftarrow v_j - K(x^*_j) \cdot |U_j(x^*)| \quad \text{for } j \in \{1, \ldots, n\} \quad (AW2)
\]

contains SAT-specific knowledge as \(U_j(x^*)\) is the set of unsatisfied clauses containing the corresponding variable. AW2 takes into account that it is necessary to flip variables in unsatisfied clauses in order to improve the current solution, and hence it guides the EA towards solutions satisfying yet unsatisfied clauses.

### 2.2 Results for a Steady-State Framework

The experiments are based on 3-SAT benchmarks described in table 1 and available at [http://www.in.tu-clausthal.de/~gottlieb/benchmarks/3sat](http://www.in.tu-clausthal.de/~gottlieb/benchmarks/3sat). All instances have been produced by the generator mkcnf.c, are satisfiable and lie in the phase transition \((m = 4.3n)\), which is known to contain difficult instances [MSL92]. While the final comparison in section 5 considers all three suites, the results in sections 2, 3 and 4 are based on the instances from suite 2 that have size \(n \in \{50, 100\}\). We employ the following EA setup: crossover is discarded according to previous studies [BEV98, EH97, GV98a, GV98b], and a simple mutation operator is used, which randomly selects an unsatisfied clause and then randomly flips one variable contained in that clause. One parent is selected by a tournament of size 2; the offspring is generated by applying mutation, and it replaces the worst individual in the current population if it is not already contained in the current population (otherwise it is rejected). The initial population is generated randomly, and the weights are adapted after \(|P|\) generations, where \(|P|\) is the population size in our experiments. All runs are terminated after \(T = 300\,000\) evaluations. The obtained success rate (SR), denoting the frequency of runs determining a solution, is shown in figure 1 for different \(|P|\). Generally, appropriate values of \(\alpha\) were selected for each parameter setup by a priori experiments.

While small population sizes exhibit the best performance for SAW and AW2, the best success rates for AW1 are obtained for large population sizes. On the one hand, the adaptation AW1 affects all variable weights whereas the mutation operator modifies only variables in unsatisfied clauses. Therefore, the number of variables that might be changed is extremely limited for small population sizes, and hence the optimization of \(r\) is too slow, resulting in worse results. For larger

<table>
<thead>
<tr>
<th>Suite</th>
<th>Problem sizes</th>
<th>Instances per size</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(n = 30, 40, 50, 100)</td>
<td>3</td>
<td>[BEV98, GV98b, MR99, RMK00]</td>
</tr>
<tr>
<td>2</td>
<td>(n = 50, 75, 100)</td>
<td>50</td>
<td>[GV98b, MR99, RMK00]</td>
</tr>
<tr>
<td>3</td>
<td>(n = 20, 40, 60, 80, 100)</td>
<td>100</td>
<td>[DJK98, MR99, RMK00]</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of the test suites used in our experiments
Fig. 1. Success rates of SAW, AW1 and AW2 for different population sizes $|P|$ (8 runs)

populations there is a higher potential of following the direction determined by $r$, due to the higher adaptation interval and the greater variety of unsatisfied clauses. On the other hand, the schemes SAW and AW2 are guided by unsatisfied clauses only. This directly matches the mutation operator that allows a rapid satisfaction of yet unsatisfied clauses. Thus, the adaptation and optimization of the fitness function are “synchronized”, enabling a fast reaction and hence good results for small population size and adaptation interval. In general, the best results are obtained for AW2, while AW1 yields a very poor SR for $n = 100$.

3 Absolute Weights and the Influence of Decay Factors

The difference in performance of AW1 and AW2 can be explained by the dynamics of the absolute weights, which are shown exemplarily for two instances in figure 2. AW2 yields a nearly constant average absolute weight, while AW1 yields an extreme increase of the absolute weights. High absolute weights reflect some kind of convergence since they cause small relative weight modifications by the adaptation mechanisms. As a change of the search direction is very time-consuming in this case, the effectivity of the adaptation process is strongly limited, which degrades the overall performance.

On the one hand, the scheme AW2 depends on the unsatisfied clauses only, and the remaining variables are not affected at all. Due to the mutation operator, it rarely happens that some clauses remain unsatisfied over several subsequent adaptations, i.e. an unbounded increase of weights is improbable. On the other hand, AW1 modifies all weights. The longer the current best solution remains unchanged, the higher become the corresponding variable weights. An increasing problem size causes a decreased probability of flipping a given bit, which leads to even higher absolute weights and partially explains the high weights for $n = 100$.

We do not explicitly present the weight dynamics of SAW as the average absolute weight is strictly monotonic increasing. Thus, SAW is susceptible to large weights and convergence of relative weights, too. Frank also observed this
problem for WGSAT, a local search algorithm using clause weights, and hence he suggested a decay factor [Fra97]. A decay factor yields a chance to reduce high absolute weights, which allows a correction of inappropriately adapted weights. Given the decay factor $\beta \in [0, 1]$, we consider the decayed adaptation schemes

$$w_i \leftarrow \beta w_i + 1 - c_i(x^*) \quad \text{(SAW\(_d\))}$$

$$v_j \leftarrow \beta v_j - K(x_j^*) \quad \text{(AW\(_1d\))}$$

$$v_j \leftarrow \beta v_j - K(x_j^*) \cdot |U_j(x^*)| \quad \text{(AW\(_2d\))}.\)

The absolute weights of SAW\(_d\) and AW\(_1d\) are bounded by $1/(1-\beta)$ for $\beta \in [0, 1)$, which prevents the convergence of relative weights. Figure 3 shows the success rate for the decayed adaptation mechanisms and several parameter choices $\beta < 1$. The results for the original variants ($\beta = 1$) are depicted by separate lines in order to allow a direct comparison. SAW\(_d\) exhibits a significant improvement for several values of $\beta$. As already observed for WGSAT in [Fra97], these values are very close to 1, especially for $n = 100$. Thus, Frank’s results can be extended to an EA employing clause weights, too. The highest improvement by a decay

Fig. 2. Averaged absolute weights $v_{avg} = \sum_{j=1}^{n} v_j/n$ for two representative instances

Fig. 3. Success rates for decayed adaptation schemes and several values of $\beta$ (8 runs)
factor is observed for $AW1_d$ and $n = 100$. In general, the decay factor is extremely effective for $AW1_d$ due to the enormous absolute weights produced by $AW1$ and the ability of the decayed version to bound and reduce weights. In contrast to $SAW$ and $AW1$, it seems to be impossible to improve $AW2$ by a decay factor.\(^1\) The reason for this is that $AW2$ does not produce high absolute weights and hence there is no need to bound or reduce the weights by some decay mechanism.

In general, $\beta$ must be selected carefully since a low decay factor destroys the information learnt during the search process. This is illustrated in figure 4 by a twodimensional projection of the weight dynamics for an instance with $n = 50$.\(^2\) Without decay ($\beta = 1$) the weights are changed continuously, which indicates the weights’ ability to act as long-term memory. An extreme decay factor ($\beta = 0.9$) destroys most information gained during the search process and hence the current weight does not exploit the previous weight trajectory.

![Fig. 4. Twodimensional projection of $AW2_d$'s weights $v$ in the first 3 000 generations](image)

It is remarkable that, despite the partly dramatic improvements for $AW1_d$ and $SAW_d$, the success rate of the original scheme $AW2$ is not reached. This indicates some natural superiority of $AW2$ concerning its ability of focusing on the most useful information and representing it in a compact way. In contrast to $AW2$, the benefits of explicit weight reductions by decay factors for $AW1$ and $SAW$ show that these schemes have some basic difficulties, which may stem from the representation of redundant, useless or misleading long-term information.

### 4 Accelerated Adaptation of Variable Weights

The results of the preceding sections indicate that $AW2$ achieves the best success rate, therefore it is interesting to check whether its performance can be further improved. An analysis of the weight dynamics of $AW2$ exhibits that there are always some variable weights that repeatedly are either increasing or decreasing. Only for $n = 100$ and $\beta = 0.999$ the SR was improved by approximately 0.75 percent.

\(^1\) Only for $n = 100$ and $\beta = 0.999$ the SR was improved by approximately 0.75 percent.

\(^2\) The vector $v \in \mathbb{R}^{50}$ is mapped to $\left(\sum_{j=1}^{25} v_j, \sum_{j=26}^{50} v_j\right) \in \mathbb{R}^2$. 
until the global search direction has changed. These weights often correspond to variables in unsatisfied clauses belonging to a slowly changing set of difficult clauses. Such a set often dominates the adaptation process over many consecutive iterations. This leads to frequent repetitions of similar adaptation steps, which represents a kind of "cycling". In order to prevent these cycles and to speed up the (slow) adaptation, we incorporate SAW to identify difficult clauses. The clause weights are used to accelerate the adaptation of the variable weights. Given the clause weights $w_i$ adapted according to SAW, we consider the scheme

$$v_j \leftarrow v_j - K(x_j^*) \cdot \sum_{i \in U_j(x^*)} w_i$$

for $j \in \{1, \ldots, n\}$ (AW2+) with $U_j(x^*)$ denoting the set of unsatisfied clauses of $x^*$ that contain the corresponding variable; note that AW2 is equivalent to AW2+ with constant clause weights $w_i = 1$. AW2+ causes a twofold adaptation as variable weights $v_j$ are adapted and the difficulty of clauses is learnt by the clause weights $w_i$. The higher the weights of unsatisfied clauses, the faster the associated variable weights are adapted. Viewed from a long-term perspective, difficult clauses have a stronger influence on the adaptation of variable weights. Further, this mechanism allows a rapid change of the current search direction, which makes the adaptation process more flexible. The results obtained in 50 runs for the benchmark instances are shown in table 2 and exhibit the superiority of AW2+. The slightly higher optimal values of $\alpha$ for AW2+ indicate a raised benefit of $r$.

<table>
<thead>
<tr>
<th>scheme</th>
<th>$n = 50$</th>
<th>$n = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>AW2</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>AW2+</td>
<td>4</td>
<td>45</td>
</tr>
</tbody>
</table>

| **Table 2. Results for AW2, AW2+, optimal population size $|P|$ and $\alpha$ (50 runs)** |

The key factor of AW2+'s superiority is the rapid change of the search direction, which can be verified by considering the maximum clause weight $w_{\text{max}}$ after the complete EA run. This weight $w_{\text{max}}$ identifies the difficulty of the most difficult clause. AW2+ identifies and satisfies harder clauses much faster since it yields on average a significantly smaller maximum clause weight than AW2.\(^3\) In the case of AW2, the weights of difficult clauses are increased more frequently as the adaptation needs more time to change the search direction appropriately.

Our experiments demonstrated the usefulness of incorporating SAW in order to accelerate the adaptation of variable weights. We also tested a decayed version of AW2+, where the clause weights are adapted according to SAW\(_d\) for some $\beta \in [0,1)$; however, the decay factors did not result in any improvement.

\(^3\) Although AW2 does not incorporate SAW, we adapted clause weights in parallel (without affecting AW2) in order to estimate the difficulty of the clauses.
5 Comparison with Other Evolutionary Algorithms

The effectiveness of EAs for SAT is measured by the success rate (SR) for selected benchmark problems. While its use is undisputed, the SR strongly depends on the computational efforts allowed by the termination criterion. As it is difficult to compare the complexity of EAs using different operators, the SR must be interpreted carefully. The SR of our steady-state EA (SSEA) is based on an evaluation limit of \( T = 300,000 \), where each child is produced by one bit flip. The evolutionary local search algorithms FlipGA [MR99] and ASAP [RMK00] generate \( T = 300,000 \) candidate solutions, where each solution is obtained by local optimization, which itself represents a (possibly) long sequence of bit flips. As the local search process is characterized by evaluated flips, it is reasonable to perceive each flip as one evaluation. Therefore, the computational efforts of FlipGA and ASAP can be measured by the total number of flips. Obviously, the limit of \( T = 300,000 \) candidate solutions induces a significantly larger number of allowed bit flips for FlipGA and ASAP. Therefore, all success rates for our EAs are obtained for lower magnitudes of bit flips compared to FlipGA and ASAP.

We measure the efficiency of an EA by the average number of flips to solution (AFS).\(^4\) In order to allow a comparison to the EA proposed in [GV98b] (RFEA), which employs a complex mutation operator causing full fitness evaluations, we consider the average number of evaluations to solution (AES), which is estimated for FlipGA and ASAP by the flip cost in terms of fitness evaluations to solution (AFES) [MR99]. Note that both AFS and AES refer to successful runs only, and that AFS is calculated for 3-SAT instances by multiplying AFES with \( n/3 \).

Table 3 presents the results for three instances of the first benchmark suite, which were proved to be difficult for Bäck et al.'s original SAWing EA [BEV98]. The perfect SR is achieved by FlipGA and ASAP, while particularly RFEA has difficulties to solve instance 10. Note that the considered EAs except RFEA routinely solved the remaining 9 instances in all runs. Table 4 shows the results for the second benchmark suite. The worst SR is observed for RFEA, while the remaining EAs exhibit a comparable SR. In general, the results for suites 1 and 2 show that RFEA is dominated by SSEA(AW2), which itself is slightly inferior to SSEA(AW2\(^+\)). This confirms the usefulness of accelerating the weight adap-

\(^4\) Although our EAs and ASAP contain adaptation mechanisms introducing additional costs, AFS is an accurate estimation of the overall efforts spent in the search process.

\(^5\) Here, we consider a slightly modified version of RFEA saving much CPU time.

<table>
<thead>
<tr>
<th>EA</th>
<th>No. 6, ( n = 40 )</th>
<th>No. 9, ( n = 50 )</th>
<th>No. 10, ( n = 100 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SR</td>
<td>AES</td>
<td>AFS</td>
</tr>
<tr>
<td>FlipGA</td>
<td>1.00</td>
<td>1514</td>
<td>20186</td>
</tr>
<tr>
<td>ASAP</td>
<td>1.00</td>
<td>1214</td>
<td>16186</td>
</tr>
<tr>
<td>RFEA</td>
<td>1.00</td>
<td>29256</td>
<td>n.k.</td>
</tr>
<tr>
<td>SSEA (AW2)</td>
<td>1.00</td>
<td>9027</td>
<td>9027</td>
</tr>
<tr>
<td>SSEA (AW2(^+))</td>
<td>1.00</td>
<td>8447</td>
<td>8447</td>
</tr>
</tbody>
</table>

Table 3. Results for three selected instances from suite 1 (50 runs)
<table>
<thead>
<tr>
<th>EA</th>
<th>$n = 50$</th>
<th></th>
<th>$n = 75$</th>
<th></th>
<th>$n = 100$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SR</td>
<td>AES</td>
<td>AFS</td>
<td>SR</td>
<td>AES</td>
<td>AFS</td>
</tr>
<tr>
<td>FlipGA</td>
<td>1.00</td>
<td>6,228</td>
<td>103,800</td>
<td>0.96</td>
<td>20,387</td>
<td>509,675</td>
</tr>
<tr>
<td>ASAP</td>
<td>1.00</td>
<td>5,843</td>
<td>97,383</td>
<td>not known</td>
<td>not known</td>
<td></td>
</tr>
<tr>
<td>RFEA</td>
<td>0.93</td>
<td>34,536</td>
<td>n.k.</td>
<td>0.66</td>
<td>33,080</td>
<td>n.k.</td>
</tr>
<tr>
<td>SSEA (AW2)</td>
<td>1.00</td>
<td>12,053</td>
<td>12,053</td>
<td>0.95</td>
<td>41,478</td>
<td>41,478</td>
</tr>
<tr>
<td>SSEA (AW2+)</td>
<td>1.00</td>
<td>11,350</td>
<td>11,350</td>
<td>0.96</td>
<td>39,396</td>
<td>39,396</td>
</tr>
</tbody>
</table>

Table 4. Results for benchmark suite 2 (50 runs)

tation. Further, we remark the higher magnitude of AFS for both FlipGA and ASAP, i.e. these EAs need much more flips to reach a solution than our SSEAs. As SSEA(AW2+) yields a higher SR for $n = 100$ on suite 1 but needs only a fraction of the flips needed by evolutionary local search, we expect SSEA(AW2+) to be superior to FlipGA and ASAP if the same number of total flips were allowed.

Figure 5 shows results for the benchmark suite 3. The algorithms investigated by De Jong and Kosters [DJK98] – including the original SAWing EA – yield a low SR. SSEA(AW2+) is superior to SSEA(AW2) but inferior to FlipGA. However, FlipGA actually consumes much more flips (multiply AES by $n/3$).

To sum up, the best SR is obtained by FlipGA, ASAP and our SSEA variants, which clearly dominate the remaining EAs suggested in [BEV98, GV98b, DJK98]. Further, the scheme AW2+ outperforms AW2. Our SSEA variants are more efficient than FlipGA and ASAP w.r.t. AFS. However, an accurate evaluation of the efficiency is difficult due to the structural differences of the EAs.

![Graph](image.png)

Fig. 5. Results for benchmark suite 3 (3 runs, except for FlipGA (5) and SSEA (4))

6 Conclusions

We compared several mechanisms to adapt fitness functions for the satisfiability problem. The adaptation scheme AW2 is superior to SAW and AW1, which suffer from large absolute weights leading to an undesired convergence of the adaptation process. Using appropriate decay factors helped to cope with this
problem, but the performance of AW2 was not reached. AW2 yields a highly compact representation of the information learnt during the adaptation process and hence decay factors were successful for SAW and AW1 only.

The scheme AW2 was further improved by using SAW to accelerate the adaptation, resulting in the scheme AW2+ that yields the most effective fitness adaptation. The comparison with two evolutionary local search algorithms exhibited that our EA yields comparable success rates but needs much fewer bit flips to determine a solution. Further, other EAs from literature were clearly outperformed w.r.t. their success rate. We believe a further performance increase can be reached by relying on both: local optimization and adaptive fitness functions.

References


Large-Scale Permutation Optimization with the Ordering Messy Genetic Algorithm

Dimitri Knjazew and David E. Goldberg

Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
{dimitri,deg}@illigal.ge.uiuc.edu
http://www-illigal.ge.uiuc.edu

Abstract. This paper presents a scaling analysis of the ordering messy genetic algorithm (OmeGA), a fast messy genetic algorithm that uses random keys to represent solutions. In experiments with hard permutation problems—so-called ordering deceptive problems—it is shown that the algorithm scales up as $O(l^{1.4})$ with the problem length $l$ ranging from 32 to 512. Moreover, the OmeGA performs efficiently with small populations thereby consuming little memory. Since the algorithm is independent of the structure of the building blocks, it outperforms the random key-based simple genetic algorithm (RKGA) for loosely coded problems.

1 Introduction

The growing interest in genetic-evolutionary algorithms (GEAs) for permutation problems is well reflected by the increasing number of approaches proposed in the literature. Research in this area is very interesting since there is a huge variety of permutation-based commercially important applications including scheduling, timetabling, and vehicle route planning problems. Unfortunately, many methods use either problem-specific or ad-hoc representation codings and operators. Also, the performance has not been sufficiently tested on hard problems, leaving the question of how these algorithms scale up unanswered.

Therefore, it would be interesting to apply genetic algorithms (GAs) that use efficient codings and operators to these problems. Furthermore, a more detailed and systematic analysis of their scale-up properties should be performed. We suggest that so-called competent genetic algorithms which solve hard problems quickly, reliably and accurately (Goldberg, 1993), would be good approaches for this undertaking. Much research has been done in this area and numerous competent GAs have been developed, including the fast messy genetic algorithm (Goldberg, Deb, Kargupta, & Harik, 1993), the linkage learning genetic algorithm (Harik, 1997) and the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999).

Another representative of competent GAs is the ordering messy genetic algorithm (OmeGA) proposed by Knjazew and Goldberg (2000). It is specialized
for solving permutation-based problems. The OmeGA uses the mechanics of
the fast messy genetic algorithm and represents the solutions by vectors of real
numbers (random keys). It has significantly outperformed the simple GA in pi-
lot experiments with ordering deceptive problems, hard permutation problems
introduced by Kargupta, Deb, and Goldberg (1992). In this paper we present
a scaling analysis of OmeGA and show that it scales up as $O(l^{1.4})$ with the
problem size $l$. We also demonstrate that the GA's performance is maximal for
small populations—an advantage that guarantees a low memory consumption,
even for large problems.

We start with a background description (Section 2) and explain the mechanics
of the OmeGA (Section 3). We then describe an ordering deceptive problem
(Section 4) that will be used for large-scale experiments (Section 5). In Section 6
we conclude the paper.

2 Background

This section provides the reader with background information necessary for un-
derstanding the forthcoming sections. We start with a short overview of some
permutation-oriented GEAs and discuss the notion of building blocks and de-
ceptive problems. We finally take a closer look at competent GAs.

Over the past few years, numerous genetic-evolutionary algorithms have been
designed for a wide variety of permutation-based problems. Many problems like
the traveling salesman problem (Goldberg & Lingle, 1985), scheduling (Davis,
1985), vehicle route planning (Blanton Jr. & Wainwright, 1993) or integrated
circuit design (Louis & Rawlins, 1991) have been tackled. Most of these tasks are
commercially important. Unfortunately, only a little research has been performed
to examine how permutation-solving GEAs scale up or, in other terms, how their
computational complexity increases with the underlying problem difficulty.

One method to perform a scaling analysis of a GA is by testing it on ar-
tificial problems where the solution is known beforehand and where the prob-
lem difficulty can be varied arbitrarily. For this undertaking, researchers in the
GA community very often used deceptive problems which are hard multimodal
optimization problems for binary strings introduced by Goldberg (1987). A de-
ceptive problem may be designed by combining a desired number of deceptive
subfunctions that mislead the genetic algorithm letting it converge to certain lo-
cal optimal points. No information about the subfunctions is passed to the GA.
To find the global optimum, partial solutions (or schemas) with above-average
fitness—the so-called building blocks (BBs)—must be identified and combined.
For our analysis we will focus on a so-called ordering deceptive problem de-
veloped by Kargupta, Deb, and Goldberg (1992) for permutations. Section 4 gives
a detailed description of this problem.

With increasing number and size (order) of the subfunctions the deceptive
problem's degree of difficulty grows. Particularly for the simple genetic algo-

rithm, the problem difficulty increases when a loose coding is chosen such that
the elements of the subfunctions are mapped to distant positions within the
problem representation. The distance between the outermost genes of a building
block is often denoted by *defining length*. BBs with large defining lengths are likely to be disrupted by traditional recombination operators such as one-point or $n$-point uniform crossover. In summary, no fixed recombination operators can ensure proper mixing with an arbitrary coding. This problem is usually referred to as the *linkage problem* in the literature. It was shown elsewhere (Thierens & Goldberg, 1993) that the required population size grows exponentially with the building-block defining length and number for simple genetic algorithms.

Hence, first-generation genetic algorithms need to be extended by some additional techniques to attack the linkage problem and to achieve a better scale-up behavior. This is the fundamental idea for the design of competent GAs. For example, the fast messy genetic algorithm (fmGA) uses a flexible representation of the chromosomes such that it becomes completely independent from the underlying problem coding and overcomes the linkage problem. Goldberg, Deb, Kargupta, and Harik (1993) showed in a large-scale analysis that the fmGA scales up no worse than $O(l^2)$ with the problem length $l$. The OmeGA is based on the fmGA and uses a robust model to represent permutations: the random key coding (Bean, 1994). Advantages of this representation are that no repair mechanisms are needed to maintain the feasibility of the solutions and that traditional crossover operators can be applied in the normal way.

### 3 Introduction to the Ordering Messy Genetic Algorithm

This section describes the design of the ordering messy genetic algorithm (OmeGA) and gives a short introduction to its main components: the fast messy genetic algorithm and random keys. The OmeGA is based on three key ideas:

- all basic mechanisms of the fast messy genetic algorithm are applied.
- the alleles are real or long integer numbers.
- the alleles are treated as random keys to encode permutations.

The "messy" genes are represented by the pair \((\text{genelocus}, \text{randomkey})\). All "messy" genetic operators are used in the usual way. This does not introduce any infeasibility problems to the robust random-key decoding of the permutations. By applying the mechanisms of the fmGA, we ensure a good building-block mixing and independence from the underlying problem coding. Therefore, one would expect the GA to scale up well.

In addition, the OmeGA's outer loop iterates over a desired number of *epochs*. An epoch starts with the first era and continues until the last era is completed. Afterwards, the best individual found so far is used as a competitive template for the succeeding epoch and so on. Multiple epochs are useful when the population size is not large enough but cannot be further increased because of memory reasons. Then, there is still a chance of finding the global optimal solution in a later epoch.

#### 3.1 The Fast Messy Genetic Algorithm

We now explain the key features of the fast messy GA: the messy representation, messy operators and organization. For a more detailed description, refer to Goldberg et al. (1993).
Unlike the simple genetic algorithms that use a fixed chromosome coding, the fast messy GA represents the genes by the pair \((\text{allele locus}, \text{allele value})\) in chromosomes of variable length. Thus, a string of messy genes may be over-specified when multiple versions of the same gene exist or under-specified when certain genes are missing. To evaluate over-specified chromosomes the genes are scanned from left to right with a first-come-first-serve precedence rule. For evaluating under-specified chromosomes, a competitive template, which is a completely specified fixed-bit string, is used in the fmGA. Before evaluation, the chromosome’s missing genes are filled with the corresponding alleles from the template. Note that a chromosome can be both under- and over-specified.

Like in simple GAs, selection and recombination are used to create a new population, except that traditional crossover is replaced by cut and splice operators (Goldberg, Korb, & Deb, 1989). The cut operator breaks a messy chromosome into two parts with a cut probability \(p_c = p_k(\lambda - 1)\), where \(p_k\) is a specified bitwise cut probability and \(\lambda\) the length of the chromosome. The cut position is randomly chosen along \(\lambda\). The splice operator joins two chromosomes with a certain splice probability \(p_s\).

The fast messy GA is organized in two nested loops: the outer loop and inner loop. The outer loop iterates over the order \(k\) of the processed building blocks. Every cycle of the outer loop is denoted as an era. When a new era starts, the inner loop is invoked which is divided into the three phases:

- initialization phase
- building-block filtering phase
- juxtapositional phase

The initialization phase creates a population of random individuals. The population size has to be large enough and chromosomes have to be long enough to ensure the presence of all possible genic and allelic combinations—candidates for building blocks after the initialization phase is completed. Then, the building-block filtering phase is invoked that works like a filter: “bad” genes not belonging to building blocks are supposed to be filtered out such that afterwards the population contains a high proportion of short strings consisting of “good” genes. This is accomplished by repeatedly performing selection and deleting random genes in all chromosomes until the overall string length is reduced to a value near \(k\).

During the juxtapositional phase, the selection operator is used and the above described cut and splice operators are applied to combine the short strings together that hopefully form the global optimal solution. After the juxtapositional phase is finished, the inner loop of the fmGA terminates. The actual template is then replaced by the best individual found so far, which becomes the new template for the next level and so on. Thus, the set of local optimal points discovered on level \(k\) serves as a launch pad for level \(k + 1\). The whole procedure can be repeated until a maximum level (era) is reached.

Two important techniques increase the fmGA’s performance: thresholding and tie-breaking. Thresholding is a modified selection operator that compares
only those individuals having a certain amount of genes in common. When tie-breaking is used, the building-block filtering phase of an era $i$ creates also strings of length 1 to $i - 1$. Consequently, chromosomes of different length are mixed faster and better.

3.2 Using Random Keys for Representation

We briefly overview the concept of random keys and the random key-based simple genetic algorithm (RKGA). The random keys have been introduced by Bean (1994). Here, real or long integer random numbers are used as sort keys to decode a sequence. For example, the individual $(0.46, 0.91, 0.33, 0.75, 0.51)$ represents the permutation $(3\ 1\ 5\ 4\ 2)$. Traditional recombination operators like single or $n$-point uniform crossover always generate feasible offspring when used on random key vectors. A simple GA that uses the random key representation is denoted by random key genetic algorithm (RKGA) in the literature. A detailed description of the RKGA can be found in Bean (1994).

4 Order Deceptive Problems

This section describes a deceptive problem for permutations we will later use for the large-scale experiments.

Kargupta, Deb, and Goldberg (1992) introduced a relative ordering function over the set of order-four permutations. In this article we denote this function by $f_{rel}$. It is defined as follows:

$$
\begin{align*}
&f(1\ 2\ 3\ 4) = 4.0 \\
&f(1\ 2\ 4\ 3) = 1.1 \\
&f(1\ 3\ 2\ 4) = 1.1 \\
&f(1\ 4\ 3\ 2) = 1.1 \\
&f(2\ 1\ 3\ 4) = 1.1 \\
&f(3\ 2\ 1\ 4) = 1.1 \\
&f(4\ 2\ 3\ 1) = 1.1 \\
&f(2\ 4\ 3\ 1) = 1.2
\end{align*}
$$

Here, only the relative ordering of the permutation elements matters. The global optimal point is $(1\ 2\ 3\ 4)$ with a function value equal to 4.0 and the misleading attractor is $(3\ 4\ 2\ 1)$ with the second highest function value of 3.2. A relative ordering deceptive problems can be constructed by concatenating a desired number of copies of $f_{rel}$. The overall function value of the whole permutation is the sum of the subfunction values. For instance, in the permutation

$$(2\ 5\ 1\ 3\ 6\ 11\ 1\ 7\ 12\ 10\ 8\ 4\ 9)$$

the elements 5, 6, 7 and 8 are in the correct relative order, thus, the corresponding subfunction value is 4.0. The items $\{1, 2, 3, 4\}$ gain a score of 1.1 according to the definition above. The relative order of the underlined elements $\{9, 10, 11, 12\}$ corresponds to $(3, 4, 2, 1)$ which is the misleading attractor with the second highest function value 3.2.
These are large and difficult problems. Considering a problem of length 128 with 32 subfunctions there exist \((4!)^{32} = 1.47 \cdot 10^{14}\) different relative orderings of which only one is globally optimal. In contrast, the number of the misleading attractors is \(2^{32} - 1 = 4.29 \cdot 10^9\) or more than four billions.

5 Experimental Results

The scale-up behavior of an optimization algorithm can be described by the tradeoff between the problem size and the number of function evaluations it requires until convergence. In this section we investigate this tradeoff for the OmeGA in experiments with relative ordering deceptive problems of order four described in Section 4. The computational complexity is expected to be the same as in the fmGA, which is no worse than \(O(l^2)\) according to Goldberg, Deb, Kargupta, and Harik (1993).

First, a description of population-sizing experiments is given. Based on the results the OmeGA’s scale-up behavior is discussed and presented. Finally, the complexity of the RKGA is experimentally determined and compared to the OmeGA.

5.1 The Scale-up Behavior of the OmeGA

<table>
<thead>
<tr>
<th>subfunction No.</th>
<th>subfunction genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 9, 17, 25</td>
</tr>
<tr>
<td>2</td>
<td>2, 10, 18, 26</td>
</tr>
<tr>
<td>3</td>
<td>3, 11, 19, 27</td>
</tr>
<tr>
<td>4</td>
<td>4, 12, 20, 28</td>
</tr>
<tr>
<td>5</td>
<td>5, 13, 21, 29</td>
</tr>
<tr>
<td>6</td>
<td>6, 14, 22, 30</td>
</tr>
<tr>
<td>7</td>
<td>7, 15, 23, 31</td>
</tr>
<tr>
<td>8</td>
<td>8, 16, 24, 32</td>
</tr>
</tbody>
</table>

Table 1. This table shows the genes belonging to the subfunctions of a loosely coded problem of length \(l = 32\).

To find out how many fitness function calls the OmeGA requires at least for solving a problem, one need to perform a parameter adjustment at first. Therefore, we conducted population-sizing experiments for problem lengths 32, 64, 128, 192, 256, 384 and 512. For every problem length we tested the OmeGA with different population sizes and determined the most suitable size yielding maximum efficiency. The goal was to find at minimum 95 percent of all BBs. The test problems were composed of copies of the subfunction \(f_{rel}\). We coded the problems loosely, such that the sum of the building-block defining lengths was maximal. A sample coding is shown in Table 1 for the problem length \(l = 32\).

The other OmeGA parameters were determined experimentally so that a good performance and a proper mixing could be achieved. The maximum chromosome length in the fast messy GA was limited to \(l\). We did not use mutation and set the splice probability to one. For the cut probability \(p_\kappa\) we took the reciprocal value of the half of the problem length: \(p_\kappa = 2/l\). The maximum number of epochs was chosen large enough such that the algorithm could find 95 percent of the BBs in all runs. In every epoch the OmeGA iterated over four eras of 65 generations each, including the building-block filtering and the juxtapositional phase. The population sizes of the eras were set according to an empirical rule keeping a fixed proportion to the total population size \(n\): \([0.1n]\) individuals in
the first, \([0.1n]\) in the second, \([0.2n]\) in the third, and \([0.6n]\) in the fourth era. We used tiebreaking and thresholding in our experiments.

Figure 1 shows the population-sizing graphs for \(l = 64, 128,\) and 192, averaged over 80 independent runs. It is interesting that all three curves have a minimum for a relatively small population size. We call these population sizes economic and denote them by \(n_{eco}(t)\). The corresponding average number of function evaluations and iterated epochs is summarized in Table 2 for the three problem lengths.

The fact that with \(n < n_{eco}(t)\) the number of functions calls becomes very large is not surprising: Consider a population with only a few members. Then, only a few, if any, BBs are expected to be expressed during the filtering phase.

As a result, the OmeGA would behave similar to a random search over the solution space. In this case, a large number of epochs and function evaluations would be required until convergence. The interesting features of the curves are their monotonously increasing parts for \(n > n_{eco}(t)\). They indicate that the best performance is achieved for a rather small population size and numerous epoch cycles. The number of function calls versus the problem length ranging from 32
to 512 is plotted in Figure 2 on a logarithmic scale. The regression line has a slope slightly smaller than 1.38, which indicates a complexity of $O(l^{1.38})$. Thus, the OmeGA scales up subquadratically as we have expected.

5.2 Performance Comparison of the OmeGA and the RKGA

To investigate how the RKGA scales up for different codings, we performed population-sizing experiments on the same problems as in Subsection 5.1. The population size was increased in constant steps of 200 individuals until 95 percent of all BBs could be found in 80 independent runs. We obtained the best results with the following RKGA parameters. The crossover probability was chosen to be 1.0 and no mutation was used at all. The recombination and selection operators were one-point crossover and binary tournament selection without replacement. The GA iterated over 300 generations in every run.

We conducted experiments for the problem lengths 64, 128, 196 and 256 for two types of problem codings. In the first case the BBs were tight and in the second case the BBs had a defining length of 12.
We refer to the last coding type as "deflen12 coding". Figure 3 presents the minimum number of function evaluations versus the problem length for the RKGA (tight and deflen12 coding) and the OmeGA (loose coding). The results clearly show that the RKGA works well on tightly coded BBs. However, when the BBs are coded loosely, its performance significantly decreases and much more function evaluations are required to find the solution. In contrast, the OmeGA solves the problems independently from the underlying coding.

6 Conclusion

This paper investigated the complexity of the ordering messy GA in scale-up experiments with hard permutation problems of bounded difficulty. We obtained results showing that the OmeGA scales up subquadratically with the problem size. Furthermore, the algorithm is linkage-friendly and works efficiently with small populations. Thanks to its "messy" representation, it is independent from the problem coding and clearly outperforms the RKGA for difficult problems. For future research we recommend the application of the OmeGA to various combinatorial real-world problems such as scheduling or vehicle routing.

Acknowledgments

The authors would like to thank Franz Rothlauf, Martin Pelikan, Martin Butz, Kumara Sastry, and Laura Albert for their useful comments and suggestions. The first author was partially supported by the "Studienstiftung des Deutschen Volkes" (Germany) and the Sigma-Xi Research Society. The authors would like to give special thanks to Professor Hans-Paul Schwefel for encouraging the first author to spend a portion of his diploma thesis studies with the second author thus enabling this collaboration to take place. Professor Schwefel also provided important guidance and essential suggestions that greatly improved this work. The first author was partially supported by the "Studienstiftung des Deutschen Volkes" (Germany) and the Sigma-Xi Research Society.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-97-1-0050. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.
References


A Hybrid GA for the Edge-Biconnectivity Augmentation Problem

Ivana Ljubić, Günther R. Raidl, and Jozef Kratica

1 Institute for Computer Graphics, Vienna University of Technology, Favoritenstraße 9-11/186, A-1040 Vienna, Austria
{ljudic|raidl}@apm.tuwien.ac.at
2 Faculty of Mechanical Engineering Strumicka 92/5, Belgrade, Serbia
jkratica@matf.bg.ac.yu

Abstract. In the design of communication networks, robustness against failures in single links or nodes is an important issue. This paper proposes a new approach for the $NP$-complete edge-biconnectivity augmentation (E2AUG) problem, in which a given graph $G(V, E_0)$ needs to be augmented by the cheapest possible set of edges $AUG$ so that a single edge deletion does not disconnect $G_0$. The new approach is based on a preliminary reduction of the problem and a genetic algorithm (GA) using a binary vector to represent a set of augmenting edges and therefore a candidate solution. Two strategies are proposed to deal with infeasible solutions that do not lead to edge-biconnectivity. In the first, more traditional variant, infeasible solutions are detected and simply discarded. The second method is a hybrid approach that uses an effective heuristic to repair infeasible solutions by adding usually cheap edges to $AUG$ until the graph augmented with $AUG$ becomes edge-biconnected. The two GA-variants are empirically compared to each other and to another iterative heuristic for the E2AUG problem using instances involving up to 1270 edges.

1 Introduction

When designing communication networks, a minimum spanning tree is usually the cheapest network that will allow a given set of sites to communicate. However, such a network is not robust against failures, since it might not survive the break of even a single link or site. For many communication structures, an important issue besides the minimization of connection costs is reliability. The network should be robust against failures in connections or switching nodes in the sense that any two nodes do not lose connection in case of up to a certain maximum number of simultaneous failures. To accomplish this, redundant communication routes must exist for any pair of nodes.

In graph theory, the terms vertex-connectivity and edge-connectivity are used to describe this kind of robustness. A connected, undirected graph $G(V, E)$ has edge-connectivity $C_E(G)$ ($C_E(G) \geq 1$) if at least $C_E(G)$ edges need to be deleted
in order to separate $G$ into disconnected components. Similarly, the graph has vertex-connectivity $C_v(G)$ ($C_v(G) \geq 1$) if at least $C_v(G)$ vertices with their adjacent edges must be deleted for disconnecting $G$. Note that $C_v$ is always less than or equal to $C_e(G)$, since at most one incident vertex for any of $C_e(G)$ edges disconnecting $G$ need to be deleted [2]. Furthermore, $C_e(G)$ is always less than or equal to the minimum-degree of all vertices $V$ [20].

In this article, we concentrate on the edge-biconnectivity augmentation (E2AUG) problem, which is stated as follows. Given are a weighted, undirected graph $G(V,E)$ with edge-connectivity $C_e(G) \geq 2$ and a spanning subgraph $G_0(V,E_0)$, $E_0 \subset E$ with $C_e(G_0) = 1$. Each edge $e \in E$ has an associated weight $w(e) > 0$. The goal is to identify a set of augmenting edges $AUG \subset E \setminus E_0$ with minimum total weight

$$W(AUG) = \sum_{e \in AUG} w(e)$$

such that graph $G_{AUG}(V,E_0 \cup AUG)$ is edge-biconnected, i.e. $C_e(G_{AUG}) \geq 2$. In $G_0$, an edge $e \in E_0$ is called a bridge if its deletion disconnects $G_0$. $G_{AUG}$ may therefore have no bridges. Note that the E2AUG problem is also called bridge-connectivity augmentation problem [4].

Besides the design of communication networks, this problem is also important to VLSI floorplanning [19]: An electronic circuit can be interpreted as a graph whose vertices are the (rectangular) functional units and whose edges are the interconnections between the units. If the graph has a so-called rectangular dual (a planar embedding of a dual of the graph such that each face and the total graph enclosure are rectangles), the most area-efficient chip-layout of the units can be determined efficiently. It is known that a rectangular dual exists if the graph is maximal planar and does not have complex triangles [19]. In general, a graph representing an electronic circuit will not fulfill these requirements; the graph must be augmented by additional vertices and edges. The most complex part of this augmentation problem can be transformed into the E2AUG problem.

Eswaran and Tarjan [3] proposed a polynomial-time algorithm for the special case when the weights $w(e), e \in E$, are all equal and $G$ is a complete graph. However, for the general case with different weights, they showed that the problem is $NP$-complete, see also [5,8]. The problem even remains $NP$-complete if weights are chosen from set \{1,2\} only [4]. In general, it is computationally too expensive to solve larger problem instances to optimality using exact techniques like branch-and-bound. Therefore, heuristics which are able to find high-quality suboptimal solutions in polynomial time are of interest.

The next section gives an overview of previous work related to the E2AUG problem. A new genetic algorithm approach is proposed in Sect. 3 and hybridized by including a problem-specific repair and improvement heuristic in Sect. 4. Empirical results for both variants are presented in Sect. 5. Especially the new hybrid approach finds good solutions with high confidence that are usually better or as good as those of another recently proposed iterative heuristic. Finally, conclusions are drawn in Sect. 6.
2 Related Work

Frederickson and Jájá [4] proposed an approximative algorithm for the E2AUG problem which is based on the following steps:

Firstly, the problem is simplified by detecting all already edge-biconnected components in \( G_0 \) and shrinking their vertex sets in \( G \) and \( G_0 \) into single new vertices, see Fig. 1. Edges that connect vertices of the same component can be discarded. Furthermore, among the edges that connect the same pair of components only the minimum weight edge must be retained, i.e. in case of \( G_0 \) the bridges. Let \( G'(V', E') \) and \( G'_0(V', E'_0) \) be these reduced versions of \( G(V, E) \) and \( G_0(V, E_0) \), respectively. In this way, \( G'_0 \) will always be a spanning tree and many edges from \( E \setminus E_0 \) can usually be discarded from further consideration as augmenting edges. Note that for each edge in \( E' \), a reference to the corresponding edge in \( E \) is stored for being able to efficiently transform a final augmenting edge set \( AUG' \) for the reduced graphs to the corresponding set \( AUG \) for the original graphs.

In the next step, \( G'_0 \) is interpreted as a directed tree in such a way that every node has a path to a selected root node \( r \). Then, a minimum weight branching from node \( r \), i.e. a directed minimum spanning tree including paths from \( r \) to all other nodes, is determined. Using this minimum weight branching, a set \( AUG' \) (\( AUG \)) of edges augmenting \( G'_0 \) (\( G_0 \)) to become edge-biconnected can be derived.

In [4], it is also shown that the total weight of the edges added by this technique is no more than twice the weight of an optimal augmentation. Subsequently, this algorithm was improved by Khuller and Thurimella [9] with regards to the time-complexity.

Khuller and Vishkin [10] proposed a similar approach for the related problem of identifying a minimum weight edge-biconnected spanning subgraph when no starting graph \( G_0 \) is given. Khuller and Raghavachari [12] proposed another algorithm using similar basic ideas for the problem of identifying a (nonspanning)
subgraph with a given edge- or vertex-connectivity for a given graph $G$. A survey on several related problems and approximation algorithms is given by Khuller [11]. Algorithms with better worst-case approximation factors for such problems are sometimes known when the edge weights fulfill certain conditions such as the triangle inequation. Du et al. [1] presented an approach to the $k$-edge-connected Steiner Network problem in metric spaces.

Recently, Zhu et al. [21] proposed another algorithm for the E2AUG problem with arbitrary weights, called DROP, that is based on [4] and leads in practice to significantly better results than the previous approaches, although it has the same worst-case approximation factor of two. Instead of deriving all augmenting edges from a single minimum weight branching as in [4], an iterative process is used which fixes only one augmenting edge at a time based on some measurement of how useful a particular edge is. After fixing one edge, its weight is reduced to zero and a new minimum weight branching is derived according to this modification. This process continues until the minimum weight branching contains only zero-weight edges and a complete set of augmenting edges has been derived.

The first author presented in [15] and together with J. Kratica in [16] a genetic algorithm (GA) for the vertex-biconnectivity augmentation problem. This approach represents solutions by binary vectors. Infeasible solutions that are not vertex-biconnected are either discarded or repaired by a heuristic. We used this algorithm and the GANP framework [14] as a first basis for approaching the E2AUG problem. However, various far reaching modifications turned out to be necessary or important to obtain the efficient algorithm proposed in the next sections.

3 The Basic Genetic Algorithm

As proposed by Frederickson and Jájá [4], we perform a pre-processing which reduces the graphs $G_0$ to $G'_0$ and $G$ to $G'$ by shrinking all already edge-biconnected components in $G_0$ into single new vertices (Fig. 1). We remove all selfloops and multiple edges, and consider only the cheapest edges connecting two different edge-biconnected components. The set of edges that might act as augmenting edges is then $E' \setminus E'_0$.

The main structure of the proposed algorithm corresponds to a traditional generational GA with overlapping populations [6]. Its operators and properties are described in the following.

**Encoding:** A candidate solution is represented by a vector $x$ of $l = |E' \setminus E'_0|$ binary genes $x_i \in \{0, 1\}$, $i = 1, \ldots, l$. Each gene $x_i$ is associated with an edge from $E' \setminus E'_0$ and indicates whether the edge is included in the set of augmenting edges $AUG'$ ($x_i = 1$) or not ($x_i = 0$).

**Initialization:** Initial solutions are created randomly by setting each gene $x_i$ independently with probability $7/8$ to $1$. In this way, an initial augmented graph
is usually dense and the probability that the solution is feasible, i.e. the augmented graph is edge-biconnected, is high. But nevertheless, it is not guaranteed that only feasible solutions are generated.

**Objective function:** For each new chromosome, the set of augmenting edges $AUG'$ is determined, and an edge-biconnectivity check is performed on the augmented graph $G'_{AUG} = (V', E'_0 \cup AUG')$. This test involves a depth-first search (DFS) on the graph and is derived from Tarjan’s algorithm to check vertex-biconnectivity [18]:

During the DFS all vertices are numbered in the order they are reached. The edges which the DFS follows to get to yet unreached nodes form a directed tree (precisely a spanning arborescence with the starting node as root). For each node $j$, a so-called *low-value* is determined, which is the smallest node reachable from $j$ by traversing zero or more tree edges followed by at most one other edge (*back-edge*) [18]. If a node $j$ with a low-value greater than the number of its parent $p$ exists, the edge $(p, j)$ is a bridge, and the graph is therefore not edge-biconnected. Otherwise, if no such node exists, the graph is edge-biconnected and $AUG'$ represents a feasible solution. The time-complexity of the whole test is $O(|E'_0 \cup AUG'|)$. A similar algorithm for testing edge-biconnectivity can also be found in [11].

In our basic GA, any infeasible solution is discarded. The objective value of a feasible solution is the sum of the weights of all augmenting edges $W(AUG) = W(AUG')$, see Eq. 1.

**Selection and population replacement policy:** Classic fitness proportional selection with different variants of scaling and tournament selection were empirically tested. The tournament selection turned out to be more robust for this application. In the experiments documented in Sect. 5, the population size was $P = 150$, and tournament selection was applied with a group size of 5.

During each generation, the worst 1/3 of the population is replaced by new solutions generated by means of crossover and mutation. In order to avoid premature convergence, each new solution is checked if it is already contained in the population and discarded in that case.

**Crossover:** Uniform crossover according to [17] is applied with a certain probability ($p_c = 85\%$ in the examples of Sect. 5). The value of each gene of an offspring is determined independently by inheriting it from the first parent with a probability of 30\% and otherwise from the second parent. In this way, about 30\% of the genes are exchanged.

**Mutation:** Each gene of a newly created solution is mutated with a certain probability $p_m$. Usually, the diversity of the genetic material is large at the beginning of a run and decreases with the time. We adapt the mutation rate during a run to promote a fast convergence to good solutions during the first generations and to introduce more diversity for escaping from local optima during later stages. The mutation probability at generation $t$ is

$$p_m(t) = p_{m0} + (p_{m1} - p_{m0}) 2^{-t/\gamma}$$  \hspace{1cm} (2)
(1) procedure repair($V'$, $E'$, $E'_0$, var AUG');
(2) begin
(3) while (bridge ← FindBridge($V'$, $E'_0$ $\cup$ AUG')) do
(4)   {$C_1$, $C_2$} ← FindConnectedComponents($V'$, $E'_0$ $\cup$ AUG' $\setminus$ {bridge});
(5)   for $e \in E' \setminus (E'_0$ $\cup$ AUG') sorted according to increasing $w(e)$ do
(6)     if $e$ connects $C_1$ with $C_2$ then
(7)        AUG' ← AUG' $\cup$ {$e$};
(8)     continue at (3);
(9) end;

Fig. 2. Pseudo-code for the greedy repair heuristic

with $p_{m0}$ and $p_{m1}$ denoting the lower and upper mutation rates for the beginning and ending, respectively. The parameter $\gamma$ controls, how fast the mutation rate changes towards $p_{m1}$.

According to preliminary tests, we suggest to set the lower mutation rate to $p_{m0} = 1/(2i)$, where $i$ is the length of the genetic code, and the upper mutation rate to $p_{m1} = 3/(2i)$.

**Caching of solutions:** During a GA run, some solutions are often generated repeatedly in subsequent generations. A classical GA evaluates each solution irrespectively of its repetition. To increase the efficiency, we use a caching technique [13,14] which memorizes all newly generated solutions with their objective values. In case of a subsequent occurrence of a solution, the objective value can quickly be retrieved instead of performing a new evaluation, as long as the solution resides in the cache. The used caching technique applies a least-recently-used strategy with a hash-queue data structure [13].

### 4 The Hybrid Genetic Algorithm

Instead of discarding a newly generated solution that is infeasible, it can also be repaired by extending set AUG with additional edges by a heuristic until the solution becomes edge-biconnected. For this purpose, the greedy heuristic shown in Fig. 2 is used. This procedure is performed in a Lamarckian way, therefore, the actual genotype is also modified according to the changes in set AUG'.

First, a bridge is determined by performing a depth-first search as already described in Sect. 3. If no bridge exists, graph $G'_{AUG}$ is already edge-biconnected and the procedure terminates. Otherwise, the bridge is temporarily removed from AUG', which will disconnect the graph into two components $C_1$ and $C_2$. These components are identified by an additional depth-first search. Next, the cheapest edge from $E'$ which is not yet contained in the augmented graph (including the bridge) and which connects $C_1$ with $C_2$ is searched for. This edge is then included in AUG'. In this way, the originally found bridge – and eventually also others – are bypassed. The algorithm restarts with checking the graph for edge-biconnectivity and identifying a bridge until the solution becomes feasible.
Table 1. Categories of test problem instances: ranges for the number of vertices and edges of $G$, length $l$ of genetic code, and edge-weights $w(e)$

| Category | $|V|$ | $|E|$ | $l$ | $w(e)$ |
|----------|------|------|-----|--------|
| A        | [20, 40] | [43, 79] | [24, 40] | $[1, |V|(|V| - 1)/2]$ |
| B        | [20, 70] | [81, 150] | [55, 81] | $[1, |V|(|V| - 1)/2]$ |
| C        | [20, 100] | [205, 248] | [126, 182] | $[1, |V|(|V| - 1)/2]$ |
| D        | [40, 100] | [384, 497] | [315, 398] | $[1, |V|(|V| - 1)/2]$ |
| M        | [70, 90] | [312, 438] | [243, 349] | [10, 1000] |
| N        | [100, 110] | [1203, 1270] | [1104, 1161] | [10, 50] |

For time-efficiency, all edges $E' \setminus E'_0$ are sorted according to increasing weights only once in a pre-processing step at the beginning of a run. In the worst-case, $AUG'$ is the empty set, each edge of the augmented graph is a bridge, and $|E'_0| - 1$ additional edges having always the largest weights must be included. The time-complexity of this heuristic is then $O(|E'||V'|)$. However, in practice only few bridges exist in most solutions generated by the GA and a single additional edge often eliminates several bridges. Therefore, the time-demand seems to be acceptable also for relatively large problem instances, see Sect. 5. In the following, we denote this hybrid version of the GA including the repair heuristic as HGA and the basic GA of Sect. 3 as BGA.

In contrast to BGA, it is for HGA not meaningful to create large augmentation sets leading to dense graphs as initial random solutions. This would only slow down the convergence to light-weight high quality solutions. Instead, initial solutions should now contain only few augmenting edges since the repair heuristic will add necessary edges of usually small weights. Therefore, each gene is now set to 1 with probability 1/16 only and to 0 otherwise.

5 Empirical Results

In this section, some typical empirical results obtained by BGA, HGA, and the iterative DROP heuristic from Zhu et al. [21] (which has been applied several times with different nodes as root), are documented. Problem instances belonging to different categories were randomly created by a test data generator already used in [21].

Table 1 shows the most important characteristics of these problem categories. The graphs $G$ of all problem instances were created randomly in such a way that they are guaranteed to be edge-biconnected. The graphs $G_0$ to be augmented are always randomly generated spanning trees of $G$. In this way, no preliminary problem reduction by shrinking already edge-biconnected components is possible and $G'$ and $G'_0$ always correspond to $G$ and $G_0$, respectively. Furthermore, the length of the genetic code is always $l = |E| - |E_0| = |E| - |V| + 1$.

For BGA and HGA, all strategy parameters were set as described in the previous sections. Each GA run was terminated when no better solution had been found within the last $G_{conv}$ generations with $G_{conv} = 5000$ for BGA and
Table 2. Results for DROP, BGA, and HGA: Average objective values of final solutions $W(AUG)$ with CPU-times $t$ and number of generations $gen$ needed; totally best observed objective values $W(AUG')$ (always from HGA).

<table>
<thead>
<tr>
<th>Problem</th>
<th>DROP $W(AUG)$</th>
<th>BGA $W(AUG)$</th>
<th>HGA $W(AUG)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$W(AUG)$</td>
<td>$t$ [s]</td>
<td>$gen$</td>
</tr>
<tr>
<td>A1</td>
<td>686</td>
<td>0.08</td>
<td>48</td>
</tr>
<tr>
<td>A2</td>
<td>496</td>
<td>0.06</td>
<td>33</td>
</tr>
<tr>
<td>A3</td>
<td>6804</td>
<td>0.13</td>
<td>50</td>
</tr>
<tr>
<td>A4</td>
<td>1642</td>
<td>0.16</td>
<td>55</td>
</tr>
<tr>
<td>A5</td>
<td>4281</td>
<td>0.48</td>
<td>180</td>
</tr>
<tr>
<td>B1</td>
<td>15512</td>
<td>0.45</td>
<td>192</td>
</tr>
<tr>
<td>B2</td>
<td>223</td>
<td>1.14</td>
<td>485</td>
</tr>
<tr>
<td>B3</td>
<td>7168</td>
<td>1.39</td>
<td>555</td>
</tr>
<tr>
<td>B4</td>
<td>5984</td>
<td>1.59</td>
<td>548</td>
</tr>
<tr>
<td>B5</td>
<td>13522</td>
<td>3.11</td>
<td>738</td>
</tr>
<tr>
<td>B6</td>
<td>19026</td>
<td>2.58</td>
<td>603</td>
</tr>
<tr>
<td>Avg(A)</td>
<td>2782</td>
<td>0.18</td>
<td>73</td>
</tr>
<tr>
<td>C1</td>
<td>22309</td>
<td>4.62</td>
<td>1261</td>
</tr>
<tr>
<td>C2</td>
<td>38191</td>
<td>7.27</td>
<td>1914</td>
</tr>
<tr>
<td>C3</td>
<td>59776</td>
<td>12.46</td>
<td>2973</td>
</tr>
<tr>
<td>C4</td>
<td>790</td>
<td>8.51</td>
<td>2510</td>
</tr>
<tr>
<td>Avg(C)</td>
<td>30266</td>
<td>8.21</td>
<td>2165</td>
</tr>
<tr>
<td>D1</td>
<td>4932</td>
<td>27.94</td>
<td>4915</td>
</tr>
<tr>
<td>D2</td>
<td>899</td>
<td>22.30</td>
<td>4196</td>
</tr>
<tr>
<td>D3</td>
<td>20411</td>
<td>53.33</td>
<td>8612</td>
</tr>
<tr>
<td>D4</td>
<td>8142</td>
<td>47.00</td>
<td>7602</td>
</tr>
<tr>
<td>D5</td>
<td>19602</td>
<td>63.14</td>
<td>8939</td>
</tr>
<tr>
<td>Avg(D)</td>
<td>10797</td>
<td>42.74</td>
<td>6853</td>
</tr>
<tr>
<td>M1</td>
<td>3010</td>
<td>41.84</td>
<td>7068</td>
</tr>
<tr>
<td>M2</td>
<td>4610</td>
<td>54.54</td>
<td>8510</td>
</tr>
<tr>
<td>M3</td>
<td>5040</td>
<td>48.21</td>
<td>6346</td>
</tr>
<tr>
<td>Avg(M)</td>
<td>4220</td>
<td>48.20</td>
<td>7908</td>
</tr>
<tr>
<td>N1</td>
<td>398</td>
<td>219.03</td>
<td>15631</td>
</tr>
<tr>
<td>N2</td>
<td>450</td>
<td>201.90</td>
<td>13600</td>
</tr>
<tr>
<td>Avg(N)</td>
<td>424</td>
<td>210.47</td>
<td>14616</td>
</tr>
</tbody>
</table>

$G_{\text{conv}} = 2000$ for HGA. The parameter $\gamma$, which controls the adaption of the mutation rate, was set to $l/2$ for BGA and to $l/4$ for HGA.

Table 5 shows results for the three algorithms. For BGA and HGA, the times $t$ (in seconds on a Pentium III/500MHz PC) and generations $gen$ indicate when the best-of-run solutions having objective value $W(AUG)$ had actually been encountered. All these values are average values determined from 10 independent
runs per problem instance. The best average objective values are always printed bold. In addition, also the totally best objective values from the 10 HGA-runs per instance ($W(AUG^*)$) are shown.

For the smaller problem categories A to C, the differences in the quality of final solutions between all three algorithms are only small. Especially for category A, HGA is able to identify the best solutions often already in the initial generation due to its heuristic, and the needed CPU-times are only fractions of a second. In some cases of category A, BGA and HGA are able to find better solutions than DROP. For the larger problem instances in categories C to N, BGA cannot compete with DROP and HGA, but HGA obtains in all cases solutions either equally good or better than those of DROP. Therefore, the repair heuristic included in HGA proves to be highly effective.

It is remarkable that HGA identifies good solutions with high confidence. For many problem instances, all or most of the 10 runs per problem instance found identical or equally good solutions. Note also that due to the heuristic, the number of needed generations to identify good solutions is dramatically reduced in comparison to BGA. Also regarding the CPU times, HGA is superior to BGA in most cases.

6 Conclusion and Future Work

A new GA-based approach for augmenting a given graph $G_0$ with an as cheap as possible set of additional edges so that the graph becomes edge-biconnected was proposed. By applying the preliminary shrinking of already edge-biconnected components into single new vertices, the set of edges that must be considered for augmentation can often greatly be reduced. Within the GA, candidate solutions are represented by binary vectors, and infeasible solutions are either discarded (in the basic GA) or repaired by a heuristic that adds cheap edges for detected bridges until the solution becomes edge-biconnected (HGA).

In the presented empirical study involving underlying graphs with up to 110 nodes and 1270 edges, the hybrid approach proved to give significantly better results than the basic GA and a recently proposed iterative heuristic [21]. Although quality differences between the final solution of the hybrid GA and the heuristic from [21] are only small, it is remarkable that HGA's solutions are nearly always better or equally good.

Future work should include the adaption of the algorithm to other graph-augmentation problems such as the more general $k$-edge-connectivity augmentation and the $k$-vertex-connectivity augmentation problems.

Acknowledgements

We thank Samir Khuller for giving us important informations related to the edge-connectivity augmentation problem and An Zhu for providing us the problem-instance generator as well as her results obtained by the DROP-algorithm ([21]).
This work is partly sponsored by the Austrian Science Fund (FWF) under the grant P13602-INF.

References

A Temporal Representation for GA and TSP

Ian Mitchell and Philip Pocknell

Middlesex University,
School of Computing Science
Hendon
London NW4 4BT
U.K.

Abstract. This paper proposes an algorithm that incorporates parallel message passing with an evolutionary process that is applied to the traveling salesman problem (TSP). The algorithm is a parallel system that relies on each city sending messages to all the other cities. A dilating circle represents the messages transmitted from each city. The emerging route is dependent on the collision criteria of the dilating circles. Each city is prohibited from transmitting messages until its temporal delay has expired. The delays associated with each city may be different and are represented in a genetic algorithm (GA) which is used to optimise the search space. This technique is not restricted to the euclidean domain, unlike the analogy used to explain it. This representation of the TSP requires no repair algorithm. The algorithm is a heuristic method for finding a near optimal route and tests on several TSPLIB are reported.

1 Introduction

Trying to find near optimal solutions to the TSP has yielded several interesting paradigms by abstracting from the natural world producing varied results e.g. ant colony optimisation, ACO, [4], elastic nets [5] and genetic algorithms [8].

Ant Colony Optimisation is based on message passing between ants upon completion of the TSP via the amount of pheromone deposits. Other methods based on emergent behaviour have been applied to networking problems [11] where the overall objective of the emergent behaviour is to provide load balancing whilst the system is undergoing some major/minor changes.

In the proposed model messages are transmitted from the cities and obey simple rules on collision with other messages from different cities. A solution emerges from these interactions to form a Hamiltonian cycle. All messages transmitted from one city are represented by a dilating circle. When two circles collide, a connection is made between the two cities that have made contact with each other and a new colony is formed. The introduction of colonies is necessary to prevent several closed paths being generated by a simple rule explained in a later section. The messages are transmitted in parallel and a temporal delay for each city is used to improve non-optimal paths. A genetic algorithm is used to find a 'fitter' or near-optimal path by heuristically modifying each of the delays belonging to the respective cities.
The paradigm proposed in this paper has advantages over previous methods that will be discussed briefly in section 1.1 and the model is investigated in more detail in section 2. Section 3 describes the details of the genetic algorithm used. Section 4 includes a summary of the results on various TSP instances. Section 5 investigates another graph problem that the same chromosome representation can solve. Finally, section 6 includes suggested variations on the proposed technique and conclusions.

1.1 Problem Definition: TSP and GA

In this paper the TSP is defined by [2] as the problem of finding a minimal length Hamiltonian cycle of \( n \) cities, where a Hamiltonian cycle is a closed path visiting each city exactly once.

The distance or cost of the Hamiltonian cycle, \( D \), is the sum of all of the distances of the connected cities on route. For symmetric TSPs the distance between two cities is fixed regardless of direction of travel. In the more general asymmetric TSP (ATSP) at least one pair of cities will have a differing distance that is dependent on the direction travelled.

The TSP is solved when the distance \( D \) is a minimum for visiting all cities. As the number of cities grows, the number of possible routes superexponentially escalates.

An effective way of searching such a search space has been to use a genetic algorithm. In Larrañaga et al [7], they show that current representations of the TSP cause many problems and it is evident that mutation, crossover and a combination of these may yield illegal representations e.g. paths that do not exist.

Larrañaga et al investigated representations of the TSP using genetic algorithms. Each representation discussed in [7] had associated repair algorithms to correct the illegal representation. This paper addresses and resolves the problem of repair algorithms by encoding genes as temporal sequences i.e. time delays for the initiation of dilating circles for each city. Therefore there are \( n \) genes and each gene represents an amount of time delay for each corresponding city. Such a method is not concerned with the order of the sequence encoded in the chromosome, but concerned with the time delay before the circles start dilating from the cities. In such a representation there can be no illegalities and hence a repair algorithm is unnecessary.

The removal of repair algorithms does not reduce the computational time taken to find the near-optimal solution since the computational burden is placed on the message passing between all the cities instead of the repair algorithm. Implementing a GA without a repair algorithm does allow the use of the many variations available to avoid any local minima and improve the search. Walters in [12] mentions that the superexponential properties of the TSP invalidates the results of simple crossover techniques; this technique relies on simple crossover, but, includes an heuristic in the preprocessing of the representation that overcomes such suggestions.
2 The Parallel Dilation Technique, PDT

Figure 1 shows a sequence of events where messages dilate synchronously and at a constant velocity. Each city sends $n - 1$ messages represented graphically by a dilating circle at time $t_s$. As the circles dilate from their respective source city synchronously, they will collide with other circles from different source cities to form new colonies of joined cities. The creation of a new colony is dependent on the colony ID of the cities to be joined and the degree of freedom of the two associated cities; both discussed subsections 2.1 and 2.2. From Figure 1 the final cost does not produce an optimal solution.

Figure 1 illustrates the sequence of events that occur when a temporal delay is introduced so that the circles start dilating at different time intervals, $t_{s+d}$, from each city at a constant velocity. To represent the temporal delay for asynchronous dilation a value is assigned randomly to each city. The city is regarded as 'dormant' until the time delay has expired, then the circle starts dilating. The non-optimal path produced on the left-hand side of Figure 1 is now replaced with an optimal path, on the right-hand side of Figure 1. The circles dilating asynchronously in parallel is the underlying principle of the proposed technique. However, such a method has some problems that will be discussed in the remainder of this section.

2.1 Freedom and Colonisation

The degree of freedom of each city is restricted, trivially, to 2. Colonies are introduced and are formed when two circles collide. The top left-hand corner of Figure 1 shows the moment city B and city C connect, the first colony is created consisting of BC and a colony ID, 1. Unconnected cities initially have a colony ID of -1. The next stage in Figure 1, city C and D connect and a new colony is created consisting of BCD with colony ID, 2, and so on for colony 3. If two cities exist within the same colony, regardless of their respective message collision, a connection is not be formed. Therefore the message collisions of BD and AC do not yield a connection. Eventually a single colony emerges that represents the path taken. In order to close the path it is necessary to have an exception. This exception occurs if there are only two cities left with a degree of freedom equal to 1 and therefore the message collision of AD yields the appropriate connection.

2.2 Overlap

Circles can overlap cities. Overlapping is when a dilated circle arrives at a dormant city, a city yet to transmit a message. The solution to overlapping is for the dilating circle to ignore the dormant city until initiated, that is the temporal delay has expired. When the dormant city becomes active, then that city is immediately joined, via a connection, to the overlapping circles’ city — and consequently a new colony is created.
Fig. 1. Simplified version of symmetric TSP. Illustrating PDT in synchronous and asynchronous mode arriving at non-optimal and optimal solutions respectively.
2.3 Collision Conditions

With respect to sections 2, 2.1 and 2.2 four conditions are provided below upon message collision:

1. Does the path exist between 2 cities?
2. Is the city active? i.e. not dormant - circle dilating.
3. Do both relevant cities have a degree of freedom of less than 2?
4. Do both cities belong to different colonies? The exception being if there are only two cities left with a degree of freedom of equal to 1.

If any of the four conditions are false then a connection is not made otherwise the two cities are connected. When all the cities have a degree of freedom equal to 2, a single colony emerges that represents the final path taken.

<table>
<thead>
<tr>
<th>node</th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
<th>(v_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t(_{value})</td>
<td>(t_{s+2})</td>
<td>(t_{s+10})</td>
<td>(t_{s+5})</td>
<td>(t_s)</td>
</tr>
<tr>
<td>bit value</td>
<td>0010</td>
<td>1010</td>
<td>0101</td>
<td>0000</td>
</tr>
</tbody>
</table>

Fig. 2. Illustration of the representation for a 4 node TSP using PDT and GA at time \(t_{s+10}\).

3 Genetic Algorithms

The representation for the time delay is illustrated in Figure 2. This representation increases linearly, since once the maximum time delay, \(t\), has been decided the chromosome length for an \(n\) city problem is \(n \times t\) bits. The genetic algorithm [6] is then used to search the solution space for a near optimal solution. The fitness function is the sum of the cost of all the connections in the path. As explained in [9] the selection is based on a 'roulette wheel' sampling method. A single crossover point is randomly selected and the crossover is haploid.
mutation point is randomly selected, and the probability of flipping the individual bit is set to 80%. This may seem unusually high, but was necessary to test PDT and lower values could yield a near optimal solution in fewer generations. The percentage of the population eradicated after each generation is set to 20% and replaced with the best 20% from the parent generation. Each generation was limited to population size of 40. The gene size was kept constant and low to 4, therefore enabling a possible delay of 15 units of time. Thus for a 76 city problem the chromosome size was 76*4=304 bits.

4 Results

The algorithm was tested on a number of problems from the TSPLIB [10] with a population size of 40 and run for 500 generations. The results can be viewed in Tables 1 and 2. The results in Table 1 show that PDT can find near optimal solutions, and compared with neural network methods it can compete. In [1] results show the instance of eil51 with the best result obtained had a relative deviance of $+2.86\%$. Table 1 shows that the PDT performed well and a hamiltonian cycle was selected and improved by 2.16%. Visual implementations of the PDT can be currently viewed at http://www.cwa.mdx.ac.uk/tsp/.

Compared with the best of other GA techniques, obtained from [7], the performance is matched, except for the gr48 library. However, PDT can find improved results over other GA variants, such as partially mapped crossovers, PMX. Although, this paper is primarily concerned with presenting a new chromosome representation and to investigate whether PDT combine with GA can yield an optimal solution. The result of TSPLIB gr24 is optimal and is encouraging, even though the set was small.

<table>
<thead>
<tr>
<th>TSPLIB</th>
<th>Number of cities</th>
<th>Optimal Value</th>
<th>Actual Distance</th>
<th>Relative Deviance of PDT</th>
<th>Relative Deviance of best ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>eil51</td>
<td>51</td>
<td>426</td>
<td>429</td>
<td>+0.704</td>
<td>+2.864</td>
</tr>
<tr>
<td>eil76</td>
<td>76</td>
<td>538</td>
<td>549</td>
<td>+4.646</td>
<td>+4.981</td>
</tr>
<tr>
<td>rd100</td>
<td>100</td>
<td>7910</td>
<td>8021</td>
<td>+1.403</td>
<td>+2.095</td>
</tr>
<tr>
<td>st70</td>
<td>70</td>
<td>675</td>
<td>724</td>
<td>+7.259</td>
<td>+1.511</td>
</tr>
</tbody>
</table>

Table 1. Results compared with best artificial neural network (see [1])

5 Fermat-Torricelli Point

Finding the Fermat-Torricelli point (aka first isogonic centre) [3] using a GA, may seem futile since the problem can be solved analytically or geometrically. However, the problem, no matter how trivial, successfully demonstrates how the
<table>
<thead>
<tr>
<th>TSPLIB</th>
<th>Number of cities</th>
<th>Optimal Value</th>
<th>Actual Distance</th>
<th>Relative Deviance of PDT</th>
<th>Relative Deviance of best GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr24</td>
<td>24</td>
<td>1272</td>
<td>1272</td>
<td>+0.000</td>
<td>+0.000</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>5046</td>
<td>5222</td>
<td>+3.488</td>
<td>+0.555</td>
</tr>
<tr>
<td>bayg29</td>
<td>29</td>
<td>9074</td>
<td>9143</td>
<td>+0.769</td>
<td>—</td>
</tr>
<tr>
<td>pr76</td>
<td>76</td>
<td>7544</td>
<td>7568</td>
<td>+0.317</td>
<td>—</td>
</tr>
<tr>
<td>berlin52</td>
<td>52</td>
<td>108159</td>
<td>113715</td>
<td>+5.137</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 2. Results compared with best GA results (see [7])

The same chromosome representation in PDT can solve a different problem. The Fermat-Torricelli point is a point O in a triangle such that the sum of the edges from each vertex to the point O is minimal. Finding the Fermat-Torricelli point using a GA requires a straightforward representation of choosing x-y coordinate as each gene and using a fitness function that calculates the sum of the distances of the three edges.

Fermat-Torricelli point can be interpreted as a graph problem and the genotype is the optimum solution. The representation has change from the nodes in the TSP to x-y coordinates in the current problem.

This problem can be solved keeping the proposed representation in PDT. However, although each gene represents a delay in the dilation of the message the rules governing the collision of the message do change.

This is inspired from nature, whereby, expose the same gene pool to a different environment and a different phenotype is generated, therefore it is the environment that influences the genotype. Hence given the same representation results in a different phenotype due to the changing environment, i.e. a new problem domain.

The psuedo-code governing the rules for message collision to generate a solution to the Fermat-Torricelli point is:-

1. if \((\text{number of messages that collide} \neq \text{number of nodes})\) then
2. \text{continue message dilation}
3. else
4. \text{prospective F-T point} \equiv \text{point of intersection of all dilating messages}

Figure 3 illustrates how PDT solves the Fermat-Torricelli point problem. Initially at time \(t_s\), nodes await for their temporal delays to expire before transmitting messages, this is demonstrated in the left-hand side of Figure 3 with one node still awaiting for it temporal delay to expire and therefore it remains dormant. Moving towards the right-hand side of Figure 3, at time \(t_{s+5}\), a collision occurs between two nodes dilating messages, however, unlike the rules that govern the message collision for the TSP, this is not the candidate solution since the number of messages involved in the collision, 2, is less than the number of nodes, 3. Continuing towards the right-hand side of Figure 3, at time \(t_{s+10}\), a candidate solution is reached whereby 3 messages collide at point \(O'\), thus equating the
expression in line 1, of the pseudo-code above, to false. Finally, by altering the temporal delays of each node new candidate solutions can be reached, to search the solution space of candidate solutions a GA is applied as in section 2.

6 Conclusions and Further Work

This paper suggests a new chromosome representation for the TSP. The outcome of this new representation is not clear, however, something distinctive about the genotype is the representation is difficult to visualise without the help of the program. Other methods have representations that are explicit, therefore the phenotype can be visualised from the genotype.

The advantage of having an implicit representation is that the chromosomes can concentrate on crossover, mutation and reproduction without knowing the consequences. The consequences are left to the set of rules. By introducing repair algorithms the technique receives feedback from the phenotype that alters the chromosome contents based on its physical representation. This ensures the visualisation between genotype and phenotype is explicit, whereas in nature the visualisation between the genotype and phenotype remains implicit. In the PDT the visualisation is implicit: from the genotype it is impossible to decipher the phenotype unless you have the correct set of rules governing the message collision.

The advantage this has is to widen the gap between the genotypes and phenotypes visualisation and therefore let the gene operations do what they do best: crossover, mutation and reproduction without interference from the phenotype. Repair algorithms receive additional feedback because they are unable to represent the chromosome representation in the phenotype, but by using the
representation in PDT the only feedback from the phenotype is in the form of selection.

The PDT algorithm does not facilitate exact solutions but is a heuristic method for finding a near optimal route using a combination of PDT and GA. The research so far has strictly applied the PDT algorithm to the symmetric TSP, however, investigations are exploring if changes to the rule set can make PDT more generic and be used to find near optimal solutions in similar domains e.g. asymmetric TSP, finding maximal cliques and Chinese postman problem.

Primarily this research provides a new representation for GA and it has yet to be investigated if the results in section 4 can be improved by varying the population size, crossover points, mutation and replacement percentages in the GA.

There is one disadvantage of PDT that is a result of the implicit representation in the genotype and will need further investigation if proof is needed that an optimal solution can be reached. In PDT there is no formal proof, that every path possible can be represented or reached by every city. If PDT cannot be proven to represent all paths then an optimal path may never be reached – Table 2 indicates optimal solutions can be reached, however, this is insufficient evidence and does not prove that all possible paths can be reached. Conversely, if PDT can be proven to represent all paths then the future is positive for PDT and will be combined with improved GA as mentioned above on the TSP and other graph related problems.

Acknowledgements

The authors would like to thank the many comments and discussions made with staff at Middlesex University, esp. Paul A. Cairns, Christian Huyck, Peter Passmore and many others. This paper was supported in part by Non-Formula Funded Research, an internal grant from Middlesex University.

References


A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem

Thomas Stützle\textsuperscript{1,2}, Andreas Grü\textsuperscript{1}, Sebastian Linke\textsuperscript{1} and Marco Rüttger\textsuperscript{1}

\textsuperscript{1} Darmstadt University of Technology, Computer Science Department, Alexanderstr. 10, 64283 Darmstadt
\textsuperscript{2} Université Libre de Bruxelles, IRIDIA, Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium

Abstract. The Traveling Salesman Problem is a standard test-bed for algorithmic ideas. Currently, there exists a large number of nature-inspired algorithms for the TSP and for some of these approaches very good performance is reported. In particular, the best performing approaches combine solution modification or construction with the subsequent application of a fast and effective local search algorithm. Yet, comparisons between these algorithms with respect to performance are often difficult due to different implementation choices of which the one of the local search algorithm is particularly critical. In this article we experimentally compare some of the best performing recently proposed nature-inspired algorithms which improve solutions by using a same local search algorithm and investigate their performance on a large set of benchmark instances.

1 Introduction

The task in the Traveling Salesman Problem (TSP) is to find a shortest closed tour through a given set of \( n \) cities with known inter-city distances such that each city is visited exactly once and the tour ends at the start city. The TSP has played a central role in the development of many nature inspired heuristics like Evolutionary Computation \cite{6, 7, 19, 22, 28}, Ant Colony Optimization \cite{4, 25}, Neural Networks \cite{1, 10}, or Simulated Annealing \cite{14, 15} to name just the most important. Currently, for several such approaches very good performance is reported. Examples are the genetic algorithms (GAs) by Merz and Freisleben \cite{7, 19}, Walters \cite{28}, Nagata and Kobayashi \cite{22}, and Gorges-Schleuter \cite{8}, the iterative partial transcription algorithm of Moebius et al. \cite{21}, the renormalization algorithm by Martin and Houdayer \cite{11}, Ant Colony Optimization (ACO) algorithms due to Dorigo and Gambardella \cite{5} or Stützle and Hoos \cite{27}, and Iterated Local Search (ILS) algorithms \cite{12, 13, 18, 17, 24}. Good computational results are also reported for some Simulated Annealing and Neural Network approaches, but, as described in the extensive review by Johnson and McGeoch \cite{12}, they seem not to be competitive with the previously mentioned algorithms.

The current state-of-the-art in the TSP suggests that the best performing algorithms combine solution construction or modification with the subsequent application of fast and effective local search algorithms \cite{12, 24, 28}. Which of the existing possibilities is the most promising one for the TSP? One possible answer would be to look at published results. Yet, often it is difficult to compare these results because of differences in the local search implementation or the use of different local search algorithms, different computing environments like computer configuration, compiler or operating system, different stopping criteria etc. Certainly, if the reported results differ substantially, it is reasonable to
conjecture that these differences are intrinsic to the algorithms compared. However, for the above cited approaches differences are typically small and concern tenth of percents of average deviation from optimal solutions or, if the same solution quality is reached, small factors with respect to computation time.

Here, we present a comparison of some of the best performing algorithms for the TSP. When comparing algorithms for the TSP, particularly important is that all algorithms use a same underlying local search algorithm, because its choice has a decisive influence on the final performance. Additionally, apparently small differences in the local search implementation may cause significant differences in the performance of these hybrid algorithms. For the TSP it is well known that the sophisticated Lin-Kernighan local search algorithm [16] often yields best performance with respect to solution quality. Yet, it is known that its run-time is very sensitive to the particular instance and the algorithm is rather complex and requires a lot of fine-tuning to run fast and produce high-quality solutions. On the other side, 2-opt is rather straightforward to implement but the solution quality it returns is not very satisfactory. Therefore, here we settled on using an efficient implementation of 3-opt by Tim Walters and that has been used in [28]. As our computational results will show, with the application of the 3-opt algorithm very good performance is obtained, justifying the choice of this local search algorithm.

The article is structured as follows. In Section 2 we introduce details on the algorithms which are compared here. Then we give details on the experimental setup and the benchmark instances used in Section 3 and report the experimental results in Section 4. We end with some concluding remarks in Section 5.

2 Nature-inspired algorithms for the TSP

The goal in the TSP is to find a shortest Hamiltonian circuit through a given set of \( n \) points. For our experimental comparison we have chosen four algorithms which showed very good performance and reimplemented them around a given 3-opt algorithm. In particular, we implemented two GAs, analogous to the GA developed by Merz and Freisleben [7, 19] and one close to that of Walters [28], an ACO algorithm [4] by Stützle and Hoos [27] and an ILS algorithm [18, 17, 24]. Certainly, other previously mentioned approaches which showed very good performance would also be worthwhile investigating. Yet, due to the considerable effort of re-implementing the algorithms we had to limit the number of algorithms compared. It is important to note that the algorithms we applied are not exact reimplementations of the original algorithms. Nevertheless, the reimplementations capture the main characteristics of these algorithms and some of them use additional diversification mechanisms, which improved performance for larger run-times but were not present in the original algorithm descriptions. In some preliminary runs we tuned each of the algorithms to achieve very good performance without removing essential parts of the original algorithm ideas.

2.1 Genetic algorithm using DPX-crossover

The genetic algorithm (GA) by Merz and Freisleben (in the following called GA-DPX)[7, 19] uses a specific crossover operator called DPX which attempts to generate an offspring that has equal distance to both of its parents. DPX works as follows: the content of the first parent is copied to the offspring and all edges which are not in common in the two
parents are deleted. The resulting parts of the broken tour are reconnected using a greedy heuristic but taking care not to use non-shared edges of the parents. For the mutation in GA-DPX we use double-bridge (DB) moves [18, 17, 12]. The DB mutation cuts the current tour into four sub-tours $s_1 - s_2 - s_3 - s_4$ (contained in the solution in that order) and reconnects them in the order $s_4 - s_3 - s_2 - s_1$. While Merz and Freisleben use a fine-tuned Lin-Kernighan local search to improve solutions, here we use "only" a 3-opt local search. This choice may lead to some loss of peak performance; nevertheless, our implementation appears to incur only a slight loss of solution quality.

An additional feature of GA-DPX is that it uses a rather small population. To avoid stagnation of the search we added a diversification mechanism: If in $n/5$ consecutive iterations since the last diversification no improved solution is found, we apply to each solution of the current population 10 random double-bridge moves and continue the search. After each diversification the best solution found since the start of the trial is reinserted into the current population if $0.9 \cdot n/5$ iterations no improved solution is found to refocus the search around this best found solution.

Our algorithm follows roughly the algorithm scheme presented in [7]: a child is generated by applying the DPX-crossover to two randomly chosen parents (in each generation 10 offspring are generated), the child is locally optimized, mutated with a probability of 0.2 (if the child is mutated, again local search is applied to it) and then it replaces an individual in the current population. The population size is chosen as 20. Note that in [19] GA-DPX follows a slightly different outline, but we found the one used here to achieve, in general, better performance.

2.2 Repair-based GA by Walters

The GA by Walters differs substantially from GA-DPX by the fact that in Walters GA the crossover and the mutation may generate infeasible tours which are then repaired by an ingenious repair mechanism; we call our GA using this repair idea GA-Repair. The main idea of the repair algorithm is to replace infeasible edges by edges with a length as close as possible to the original length of the infeasible edge. We refer to [28] for details. An additional difference to GA-DPX is that GA-Repair uses a relatively large population. Additionally, Walters uses a brood selection mechanism [28] which generates several children for a pair of parents and then chooses only the best one or two children as offspring. We did not implement exactly this mechanism, but in a similar spirit we generated a large number of offspring (three times the population size $m$ which is set to $m = 100$ as in [28]) in each generation, choosing parents using a fitness based ranking which gives the fittest solution a three times higher probability to be chosen for the crossover than the worst one. Each offspring is mutated with a probability of $p = 0.5$ by applying two random 2-opt moves and is then locally optimized. The new population is then composed of the $popsize$ best solutions avoiding duplicate solutions. Additionally, we also introduced a diversification mechanism similar to that applied in GA-DPX, yet, without reintroducing the best solution since the start of the trial.

2.3 Ant colony optimization

In ACO algorithms a colony of artificial ants iteratively constructs solutions to the problem to be solved and communicates indirectly by (artificial) pheromone trails which, in
the TSP application, are associated with the edges. The ants construct, starting at some initial city, tours by iteratively choosing from the current city \( i \) a next city \( j \) with a probability proportional to \( \tau_{ij} \cdot \eta_{ij}^\beta \). For many problems, best performance is achieved if the ants’ solutions, once they are completed, are improved by a local search algorithm. Here, we applied \( MAX-MIN \) Ant System (\( MMAS \)) which has been shown to be one of the best performing ACO algorithm for the TSP [24, 27, 25]. In \( MMAS \), features of search diversification like pheromone trail re-initialization etc. are used, similar in spirit to the ones we used in GA-DPX; we refer to [27] for more details.

In \( MMAS \) we use a colony of 25 ants, \( \beta = 2 \), and the next city is chosen among a candidate set of the 20 nearest cities. The pheromone trails are reinitialized if in \( \max\{100, n/10\} \) consecutive iterations no improved solution is found (for more details see [27]).

### 2.4 Iterated local search

Iterative improvement algorithms are easily trapped in local minima. The basic idea of ILS is to modify the current solution \( s \) applying a \textit{kick-move}, moving it to a point \( s' \) beyond the neighborhood searched by the local search algorithm and to continue the local search from \( s' \). Typically, the solution modified in each iteration is the best solution found so far [17, 12]. ILS algorithms and, in particular the iterated Lin-Kernighan version of ILS, are known to be among the best performing algorithms for the TSP [17, 12].

As in most other ILS algorithms for the TSP we apply the double-bridge (DB) kick-move where the cutpoints are chosen randomly within a window of length \( \max\{n/2, 500\} \) of the current tour (the same mutation was used in GA-DPX). One problem associated with applying the DB move only to the best solution is that ILS may easily get trapped in specific regions of the search space [24, 26]. One simple possibility to avoid this behavior is to restart ILS from scratch if no improved solution is found for a given number of iterations (ILS-Restart), here chosen as \( n \) iterations. Yet, here we apply a more sophisticated mechanism, which is called Fitness-Distance-Based Diversification (ILS-FDD) in [24, 26], to diversify the search. ILS-FDD generates a new starting solution for ILS in an iterative way; in particular, it obtains such a solution by first generating a set of candidate solutions by the standard ILS mechanism and then choosing one candidate solution which is of high quality as well as rather distant from the current solution. This process is repeated until a solution at a given minimal distance to the current solution is obtained (see [24, 26] for more details).

### 2.5 The 3-opt implementation

The 3-opt implementation we use is based on a code provided by Tim Walters [28]. The 3-opt algorithm uses the following standard speed-up techniques: It (i) restricts the set of moves which are examined to those contained in a candidate list of the 40 nearest neighbors [2, 16, 23]; (ii) it performs a fixed radius nearest neighbor search [2] and uses \textit{don’t look bits} associated with each node [2]. Before applying 3-opt we set all don’t look bits off, that is all cities are considered as a starting city to find an improving move. The 3-opt algorithm is a first-improvement local search which gives preference to applying 3-opt moves before the 2-opt moves which are also checked in a 3-opt implementation. The 3-opt implementation takes care that the exchange of partial tours in a move is restricted to the smallest possible part to avoid the cost associated with copying arrays; the distances between all cities are kept in a distance matrix.
3 Experimental setup

We compare the algorithms on a large set of symmetric, Euclidean TSP instances with known optimal solutions from TSPLIB, accessible at http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95, ranging in size from 198 cities up to 2392 cities. The number in the instance identifier is the problem size, for example, instance rat783 has 783 cities. As a stopping criterion we used a CPU-time limit which was chosen large enough such that each of the algorithms could reach limiting behavior and an additional significant increase in solution quality would require a considerably increased CPU-time. In the experiments we noted that all the algorithms could find optimal solutions for instances with up to 1000 cities in almost all trials. For these instances we compare the algorithms using run-time distributions (RTDs) [9] which give the cumulative empirical probability of finding an optimal solution as a function of the run-time. The RTDs are based on 25 independent trials of each algorithm. For instances with more than 1000 cities, 10 trials were done if not indicated otherwise. On these larger instances we present standard summary data due to the small number of trials.

To eliminate some factors which might affect final performance, all algorithms were coded in "C" and, where possible, they used similar data structures (for example, tour representation by arrays, same local search algorithm etc.) and were compiled and run on the same machine, a double processor Pentium III 450MHz machine running Redhat Linux 6.1 with 256 MB RAM. Due to the sequential implementation of the algorithms only one single processor is used.

4 Experimental results

4.1 Experimental comparison with run-time distributions

Among the instances in TSPLIB with $n \leq 1000$ we have chosen those proposed in the First and the Second Contest on Evolutionary Optimization [3]. Of these instances we do not present results for eil51 and kroA100 because they were often solved in the initialization phase of the algorithms. Additionally, we run for each algorithm 25 trials on the instances pr1002, pcb1173, and dl291.

Figure 1 gives the resulting RTDs. For most instances, ILS-FDD shows best performance which can be noted at the fact that for the same computation time it reaches a higher probability of finding optimal solutions than the competitors and on some instances (att532 and dl291) it is the only algorithm which always finds optimal solutions in the given run-time. Yet, in general the relative ranking of the algorithms is strongly instance dependent. For example, GA-Repair is the worst performing algorithm on instance pcb442, but among the best performing ones on instances rat783 and pr1002. Additionally, it can be observed that MMAS has particular problems of finding optimal solutions on instance pr1002 which, in contrast, is solved easily by the other algorithms.

An interesting observation from the RTDs is that the algorithms need an algorithm and instance specific initialization time $t_{init}$ before they find the first optimal solution. $t_{init}$ appears to be particularly large for GA-Repair and MMAS. In GA-Repair this effect is mainly due to the large population size compared to GA-DPX or even ILS-FDD, which uses only one single solution; MMAS has been specifically designed to be an ACO algorithm with a strong initial exploration of the search space, explaining this behavior.
Fig. 1. Comparison of the algorithms using run-time distributions which give the cumulative empirical probability of finding optimal solutions on the TSP instances d198 (upper left), lin318 (upper right), pcb442 (second row, left), att532 (second row, right), rat783 (third row left), pr1002 (third row, right), pcb1173 (fourth row, left), and d1291 (fourth row, right). On each instance 25 independent trails have been run.
Yet, once this initialization time is passed, within some additional computation time the algorithms find a large number of additional optimal solutions. This effect is more notable on the smaller instances with a few hundreds of cities, while on most larger instances it is significantly harder to find optimal solutions and the variance of the solution times appear to increase. For example, on instance lin318 GA-Repair finds the first optimal solution after 5.9 seconds and in the longest of the 25 trials it takes 12.6 seconds to find the optimal solution. An interesting question regarding $t_{\text{init}}$ is its scaling behavior. This behavior has direct consequences on the usefulness of algorithm restarts (which would be the case for ILS-Restart) when compared to other diversification mechanisms which avoid such algorithm restarts (this is the case in ILS-FDD).

4.2 Experimental comparison on large instances

To obtain a more complete picture of the algorithms' performance we extend our experiments to a larger set of TSPLIB instances comprising all Euclidean TSPLIB instances with $1000 \leq n \leq 2392$ cities. On most instances 10 trials are run; the experimental results are given in Table 1. In general, for all algorithms very good performance is obtained and all algorithms found, with only few exceptions, on average solutions within only few tenths percent of the optimal solution. The computational results on these larger instances also confirm the findings in the previous section. On most instances ILS-FDD gives the best average performance and finds the largest number of optimal solutions. Yet, in general, which of the algorithms shows best or second best performance is again strongly instance dependent. Noteworthy is the very good performance of GA-Repair on two of the largest instances, d2103 and pr2392, where it was the only algorithm which was able to find in all runs optimal solutions. Yet, on the two instances u2152 and u2319 both GA show rather poor results, which may be rather due to some particularities of these instances than because of a loss in the performance of the two GAs. Regarding the influence of the instance characteristics on the performance we noticed that ILS-FDD compared very favorable to the other algorithms on clustered instances. Extreme examples of such clusters of cities are fl11400 and fl11577, but to some less extent such clusters are also available on some other instances. Yet, many other factors like the number of optimal solutions etc. may be responsible for the performance differences.

The surprisingly good performance of ILS-FDD is certainly due to the diversification features of ILS-FDD (note that the diversification mechanism applied in the two GAs and the ACO algorithms also improve their performance on most instances). If instead we apply ILS without any such features (called ILS-standard), that is if the DB kick-move is always applied to the best solution found since the start of the trial, in the same computation time ILS-standard could find only 9 optimal solutions on pr1002, 3 on pcb1173, 6 on d1291, 1 on f11577 and none on pr2392 and the average percentage deviation increased to 0.15%, 0.11%, 0.082%, 0.47%, and 0.21%, respectively. Yet, already by the very simple ILS-Restart extension, a much improved performance over ILS-standard can be obtained finding 25, 2, 22, 4, and 1 optimal solutions, respectively and reducing the average percentage deviation to 0.0%, 0.015%, 0.0035%, 0.0054% and 0.11%, respectively. In general we noticed that the advantage of ILS-FDD over ILS-Restart strongly increases with instance size, certainly also due to the large initialization time for larger instances. Additionally, also the population-based algorithms compare much more favorably to ILS-Restart: on most instances at least one of the three population-based algorithms performs
Table 1. Comparison of the algorithms on a large set of TSPLIB instances with 1000 \( \leq n \leq 2392 \).

For each instance we report how often the optimal solution is found in a given number of trials (\( n_{\text{opt}} \) / No. of trials), the average percentage deviation from the optimum (\( A_{\text{avg}} \)), and the average CPU-time \( t_{\text{avg}} \) to find the best solution in a trial. The maximally allowed CPU-time for each trial has been limited to 1,200 sec. for \( 1,000 \leq n \leq 1,500 \), 2,400 sec. for \( 1,500 < n \leq 2,000 \), and 3,600 sec. for \( n > 2,000 \). Best results with respect to \( n_{\text{opt}} \) or \( A_{\text{avg}} \) are indicated in boldface, second best in italics.

<table>
<thead>
<tr>
<th>Instance</th>
<th>DPX 3-opt</th>
<th>Repair</th>
<th>MMAS</th>
<th>ILS-FDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( n_{\text{opt}} )</td>
<td>( A_{\text{avg}} )</td>
<td>( t_{\text{avg}} )</td>
<td>( n_{\text{opt}} )</td>
</tr>
<tr>
<td>dasj1000</td>
<td>0/10</td>
<td>0.0076</td>
<td>607.7</td>
<td>0/10</td>
</tr>
<tr>
<td>pr1002</td>
<td>25/25</td>
<td>0.0</td>
<td>132.8</td>
<td>25/25</td>
</tr>
<tr>
<td>u1060</td>
<td>0/10</td>
<td>0.031</td>
<td>662.5</td>
<td>0/10</td>
</tr>
<tr>
<td>vm1084</td>
<td>0/10</td>
<td>0.020</td>
<td>225.9</td>
<td>0/10</td>
</tr>
<tr>
<td>pcb1173</td>
<td>16/25</td>
<td>0.015</td>
<td>571.4</td>
<td>15/25</td>
</tr>
<tr>
<td>d1291</td>
<td>16/25</td>
<td>0.034</td>
<td>396.1</td>
<td>16/25</td>
</tr>
<tr>
<td>r11304</td>
<td>10/10</td>
<td>0.0</td>
<td>45.5</td>
<td>10/10</td>
</tr>
<tr>
<td>r11323</td>
<td>1/10</td>
<td>0.009</td>
<td>104.5</td>
<td>0/10</td>
</tr>
<tr>
<td>nrw1379</td>
<td>0/10</td>
<td>0.067</td>
<td>744.3</td>
<td>0/10</td>
</tr>
<tr>
<td>f11400</td>
<td>0/10</td>
<td>0.12</td>
<td>787.9</td>
<td>1/10</td>
</tr>
<tr>
<td>u1432</td>
<td>0/10</td>
<td>0.45</td>
<td>787.1</td>
<td>0/10</td>
</tr>
<tr>
<td>f1577</td>
<td>0/10</td>
<td>0.16</td>
<td>1283.4</td>
<td>0/10</td>
</tr>
<tr>
<td>d1655</td>
<td>9/10</td>
<td>0.0003</td>
<td>809.6</td>
<td>9/10</td>
</tr>
<tr>
<td>vm1748</td>
<td>2/10</td>
<td>0.045</td>
<td>713.8</td>
<td>0/10</td>
</tr>
<tr>
<td>u1817</td>
<td>0/10</td>
<td>0.14</td>
<td>1470.6</td>
<td>0/10</td>
</tr>
<tr>
<td>r11889</td>
<td>7/10</td>
<td>0.041</td>
<td>1102.8</td>
<td>1/10</td>
</tr>
<tr>
<td>d2103</td>
<td>6/10</td>
<td>0.016</td>
<td>1476.8</td>
<td>10/10</td>
</tr>
<tr>
<td>u2152</td>
<td>0/10</td>
<td>0.18</td>
<td>2149.7</td>
<td>0/10</td>
</tr>
<tr>
<td>u2319</td>
<td>0/10</td>
<td>1.24</td>
<td>2322.3</td>
<td>0/10</td>
</tr>
<tr>
<td>pr2392</td>
<td>4/10</td>
<td>0.075</td>
<td>842.9</td>
<td>10/10</td>
</tr>
</tbody>
</table>

Better than ILS-Restart.

One last concern is how our computational results compare to the previously published results of the original algorithms. The performance of MMAS is, except on instance f11577 significantly better than those reported with 3-opt local search in [27] which is mainly due to the better performance of the 3-opt applied here. For ILS-FDD the results in [26] appear, when adjusting for the different computer speed, to be slightly better than those reported on the larger instances, while they are worse on the smallest five instances. Again, this is probably due to slight differences of the 3-opt implementation and other implementation details. Also our GA-DPX implementation has shown very good performance. It appears to perform roughly comparable on the instances with \( n \leq 1000 \) to the results published in [19]; better results are reported by the same authors for a somewhat modified MA in [20]. Yet, with increasing instance size our GA-DPX appears to perform slightly worse which may be due to the more powerful Lin-Kernighan local search algorithm applied in [19, 20] (yet, a detailed comparison is not possible here because the authors report only results for two instances with \( n > 1000 \)). Regarding GA-Repair, our implementation performs comparable to the implementation of [28] with the exception of instance f11577, where the computational results reported in [28] are sig-
nificantly better. Our implementation achieves somewhat better results on other instances like, for example, rat783 (we found optimal solutions in every run, while in [28] only a solution probability of 0.8 was reached).

5 Conclusions

In this paper we have experimentally compared some of the best performing algorithms for the TSP. Because the particular local search algorithm chosen has a very significant influence on the final performance, we re-implemented two GAs, one ACO algorithm, and one ILS algorithm, around a same 3-opt algorithm. For all algorithms very good computational results could be obtained, sometimes even better than in the original publications. We found ILS-FDD to be the best performing algorithm on most instances. This is quite remarkable, because typically iterated local search algorithms are relatively easy to implement and have less parameters to be tuned than other, more complex algorithms. This fact also suggests that a population of solutions is not necessary to obtain a very high performing algorithm for the TSP. Nevertheless, it should be noted that the ILS-FDD variant uses some extra features like a specific diversification mechanism. More straightforward ILS extensions like ILS-Restart still show very good performance, but they compare less favorable to the three population-based approaches. Hence, one interesting conclusion is that if algorithms which manipulate only a single solution include appropriate diversification mechanisms, they may be very competitive with high performing population-based algorithms. This observation raises interesting research issues. One such issue is, how the relative performance of the algorithms applied here differs if weaker (for example, 2-opt) or stronger (for example, the Lin-Kernighan heuristic) local search algorithms are applied. Another issue is on which types of practically relevant problems population-based algorithms achieve more advantage over algorithms modifying single solutions.

Acknowledgements

We would like to thank Tim Walters for making available a version of his 3-opt and remarks on this work and Peter Merz for comments on the paper.

References

A Genetic Algorithm for VLSI Floorplanning

Christine L. Valenzuela\(^1\) and Pearl Y. Wang\(^2\)

\(^1\) Department of Computer Science, Cardiff University
PO Box 916, Cardiff CF24 3XF, United Kingdom
C.L.Valenzuela@cs.cf.ac.uk

\(^2\) Department of Computer Science MS4A5, George Mason University
Fairfax, VA 22030-4444, USA
pwang@cs.gmu.edu

Abstract. We present a genetic algorithm (GA) which uses a normalized postfix encoding scheme to solve the VLSI floorplanning problem. We claim to have overcome the representational problems previously associated with encoding postfix expressions into GAs, and have developed a novel encoding system which preserves the integrity of solutions under all the genetic operators. Optimal floorplans are obtained for module sets taken from some MCNC benchmarks. The slicing tree construction process, used by our GA to generate the floorplans, has a run time scaling which compares very favourably with other recent approaches.

1 Introduction

One of the most important stages in the physical design of VLSI circuits is floorplan design: the placement of a set of rectangular circuit modules on a chip so as to minimize the total area and the total interconnecting wire length. When placing circuit modules (or *macro cells*) many of the modules are themselves not yet fully designed and frequently have some flexibility in their shape. For example a circuit module made up from 12 identical components may have them placed in one row of 12 components, 2 rows of 6 components, 3 rows of 4 components etc., offering the floorplan designer a range of possible shapes for that module. Using a technique based on a *slicing floorplan*, which can be obtained by recursively dividing a rectangle into two parts with either a vertical or a horizontal line, it is possible to fully exploit the available flexibility in the circuit modules and efficiently combine module placement and area optimization into a single algorithm. The purpose of our paper is to present a genetic algorithm (GA) which appears to be very effective at breeding good (often optimal) slicing floorplans. We believe that with our GA we have overcome most of the representational problems usually associated with encoding floorplans, and claim that our solution is elegant as well as effective.

We now summarize some important definitions based on [13]. Let a given rectangle, \( R \), have height \( h(R) \), width \( w(R) \) and area \( A(R) \). The *aspect ratio* of \( R \) is...
R is the ratio $h(R)/w(R)$. A soft rectangle is one that can have different shapes as long as the area remains the same. The shape flexibility of a soft rectangle specifies the range of its aspect ratio. A soft rectangle of area $A(R)$ is said to have a shape flexibility $r$ if its aspect ratio can take on any value between $1/r$ and $r$. The shape flexibility of modules provides a continuous range of candidate aspect ratios for our soft modules.

An alternative to the slicing floorplan, which is favoured by many researchers, is the non-slicing floorplan. In the non-slicing floorplan there is no requirement for recursive construction, and tighter packings are often possible using this approach. Our main motivation for using a slicing floorplan approach is that by considering only slicing floorplans, we are able to significantly reduce the size of the search space. Stockmeyer [9], examining cases where each subcircuit may have different layout alternatives in a floorplan, showed that there is an efficient polynomial time area optimization algorithm for slicing floorplans whereas the area optimization problem for non-slicing floorplans is NP-Hard.

A recent non-slicing placement technique, called the sequence-pair method [5], has been extended to handle soft modules and area optimization [4]. However the sequence-pair method has to solve expensive convex programming problems in order to determine the exact shape of each module, and this results in a very long runtime. Another relatively new technique, called the Bounded Sliceline Grid (BSG) packing algorithm [6], has proved successful in the placement of hard rectangles (i.e. rectangles with no flexibility in their shape). On the downside, though, a single application of the BSG packing algorithm scales at $T(n) = O(n^2)$ for hard modules, given the $n \times n$ grid size suggested by the authors. This compares with $T(n) = O(n)$ for hard modules using a slicing floorplan approach. It is our belief that the average case run time scaling of the combined placement and area optimization algorithm for slicing floorplans with soft modules at $T(n) = O(n \lg n)$ will be hard to beat.

To the best of our knowledge the slicing structure representation used in our GA is novel. Although there are several other examples of genetic algorithms applied to slicing tree structures in the literature, it would appear that none of them uses a GA to manipulate encoded normalized postfix strings. Schnecke and Vornberger, for example, used a GA to manipulate the slicing tree directly [8]. However, to facilitate their crossover involves complex repair mechanisms simply to ensure that the final product (or offspring) represents a legal slicing floorplan, with no duplications or deletions of modules. Another approach [2], in what is probably the best known study of its type, used a collection of four different crossovers and applied them to postfix expressions that were not normalized (using non-normalized expressions greatly increases the search space).

We view our main contribution to the field of slicing tree optimization as the extension of the ideas of Wong, Leong and Liu [12]: they used a normalized postfix representation for simulated annealing, and, through the addition of a novel encoding, we have adapted the approach to produce a simple but effective genetic algorithm. We test our GA on soft modules from the benchmark MCNC data sets, ami33 and ami49. The objective of our present study is a 'proof of
concept' and we limit our objective function to the construction a floorplan of minimum area. Other elements, such as the minimization of the total wire length, will be included in the cost function at a later date.

In section 2 we begin with a review of slicing floorplans and their postfix representations, and then we briefly describe our approach to the addition of shape curves for the combination of soft modules. Section 3 describes the representation we use to encode our slicing floorplans, and also the decoder which interprets these structures as normalized postfix expressions. In section 4 we describe our genetic algorithm and section 5 presents our results. We conclude in section 6 with a summary of our achievements and an outline of our plans for future work.

2 Slicing Structures and Postfix Representations

A slicing floorplan is a rectangular floorplan with \( n \) basic rectangles that can be obtained by recursively cutting a rectangle into smaller rectangles using a series of vertical and horizontal guillotine cuts. A slicing floorplan can be represented in the form of a binary tree, called a slicing tree, in which each internal node of the tree is labelled either * or +, corresponding to a vertical or a horizontal cut respectively. Each leaf represents a basic rectangle and is labelled between 1 and \( n \), where \( n \) is the total number of basic rectangles. Each slicing tree can be represented, alternatively, using a postfix expression. The postfix expression is derived by carrying out a post-order traversal.

There is a one-to-many relationship between slicing floorplans and slicing tree representations for slicing floorplans. If we restrict our representations to skewed slicing trees, however, we obtain unique depictions for slicing floorplans [12]. A skewed slicing tree is a slicing tree in which no node and its right child have the same label in \{*, +\} and it is obtained by making consecutive vertical cuts from right to left, and making consecutive horizontal cuts from top to bottom. The postfix expression derived from a skewed slicing tree is called a normalized postfix expression, and provides a linear form of the representation. Figure 1

![Slicing floorplan, skewed slicing tree and corresponding normalized postfix expression](image-url)
illustrates a typical slicing floorplan, and the corresponding skewed slicing tree and normalized postfix expression. A normalized postfix expression is obtained by traversing a skewed slicing tree in post-order and is characterized by chains of \{*, +\} operators in which the operators alternate. For example the postfix expression 1 2 3 * 4 * is normalized, but the expression 1 2 3 + + 4 * is not (because of the two adjacent + symbols). A slicing floorplan with \((n - 1)\) cuts will produce \(n\) basic rectangles. Thus a postfix expression consists of exactly \((2n - 1)\) entries.

A normalized postfix expression which characterizes a slicing floorplan can be written: \(\pi_1 c_1 \pi_2 c_2 \pi_3 c_3 \pi_4 c_4, \ldots, \pi_n c_n\) where \(\pi_1 \pi_2 \pi_3 \pi_4 \ldots \pi_n\) represent a permutation of the \(1, 2, \ldots, n\) basic rectangles, and the \(c_i\)'s are chains of operators, either \(+ * + * + * \ldots, or * + * + * \ldots\). If we let \(l(c_i)\) represent the length of the chain, \(c_i\), then \(\sum l(c_i) = n - 1\), and \(l(c_1) = 0\). Also for any position, \(i : 1 \leq i \leq n\), \(\sum l(c_i) \leq i - 1\) (this is often referred to as the balloting property).

### 2.1 Circuit Module Placement

In the discussion so far, we have viewed a slicing tree as a top down description of a slicing floorplan, in which the slicing tree specifies how a given rectangle is cut into smaller rectangles by a series of vertical and horizontal cuts. An alternative is to view a slicing tree as a description of a bottom up procedure. From a bottom up point of view the slicing tree describes how pairs of smaller rectangles can be combined recursively to yield larger rectangles. Figure 2 shows the actions of the binary operations + and * on the two rectangles \(A\) and \(B\): ‘+’ puts \(B\) on top of \(A\), and ‘*’ puts \(B\) on the right of \(A\). In the example depicted in Figure 2, after the two rectangles \(A\) and \(B\) have combined under + or *, the combined module is replaced by the smallest possible enclosing rectangle, resulting in the creation of dead space (or waste) in the floorplan.

\[
\begin{align*}
\text{A} + \text{B} &= \begin{array}{c}
\text{B} \\
\text{A}
\end{array} \\
\text{A} * \text{B} &= \begin{array}{c}
\text{A} \\
\text{B}
\end{array}
\end{align*}
\]

**Fig. 2.** Binary operations for combining rectangles

### 2.2 Area Optimization Using Soft Modules

Various vertical (\(y\) coordinate) and horizontal (\(x\) coordinate) dimensions are possible for a soft module with aspect ratio, \(\rho\), such that \(1/r \leq \rho \leq r\), and these can be modelled by a shape curve, \(\Gamma\). \(\Gamma\) is a smooth, continuous curve, lying entirely within the first quadrant, such that the \(x\) and \(y\) coordinates of points lying on or above the curve define the feasible region.
Fig. 3. Shape curve addition for vertical combination. Shape flexibility for $A$ is 2 and $B$ is 3.

Pairs of soft modules, $A$ and $B$, can be combined by adding their shape curves: $A \ B \ +$, by adding along the $y$ direction and $A \ B \ \ast$, by adding along the $x$ direction. Figure 3 illustrates a vertical (+) combination of a pair of modules. The shape curves depicted in the diagram indicate possible height and width dimensions for $A$, $B$ and for the enclosing rectangle $A \ B \ +$. The two points at either end of each of the curves mark the limits of flexibility for the rectangles, which means that the rectangles can only be made taller or wider than this by the addition of dead space. When a pair of soft modules is combined, the new shape curve can be computed simply by adding together the so-called ‘corners’ of the curves for the component modules. The diagram shows clearly that the shape curves for basic modules of fixed orientation (i.e. no rotation is allowed) are each completely characterized by two ‘corners’. (Note: a hard module of fixed orientation is completely characterized by a single point or ‘corner’).

So much for combining basic rectangles in pairs. In order to produce a slicing floorplan from a postfix expression it is necessary to create a repetitive process which combines together super-modules, as well as basic modules, adding together their shape curves in a bottom-up fashion. Fortunately the process of adding shape curves for super-modules is essentially the same as the procedure for combining two basic modules, only with more ‘corners’ to add. Although full details of our routines for combining shape curves are omitted from the present paper, to save space, we plan to publish them elsewhere [11]. Essentially our approach makes use of our observation that the space occupied by a rectangular module/super-module varies uniformly between any two adjacent corners, and we perform a full evaluation of all the points. Since a basic module of fixed orientation has a maximum of two corners, a combined module, produced from two basic modules, will have at most four corners on its shape curve. Combining $n$ modules following an arbitrary binary tree, where the number of corners may
double at each level as the algorithm combines more modules and moves up the tree, gives an average case run time of \( T(n) = O(n \log n) \) [9]. Despite scaling well the run time for our shape curve combination routine is currently rather long, because we have not, as yet, incorporated approximations, as suggested by Wong [12], to reduce the number of corners accumulated by our shape curves.

3 The Representation and Decoder

Our representation is order based and consists of an array of records, with one record for each of the basic rectangles of the data set. Each record contains three fields:

- a rectangle ID field: this identifies one of the basic rectangles from the set \( \{1, 2, 3, \ldots, n\} \)
- an op-type flag: this boolean flag distinguishes two types of normalized postfix chains, \( T = +*+**++\ldots \) and \( F = *++++++*\ldots \)
- a chain length field: this field specifies the maximum length of the operator chain consecutive with the rectangle identified in the first field.

Outline decoder algorithm

1. Examine next (first) record; print the rectangle ID.
2. Generate a chain of alternating operators of op-type specified in op-type flag. This chain should have length defined in the length field.
3. Print operators, in sequence, from the chain generated in 2) until either you get to the end of the chain or the addition of more operators would violate the balloting property.
4. If there are more records left to process then go to 1) else complete the normalized postfix expression by printing further operators at the end of the postfix string until the number of operators is one less than the total number of rectangles in the expression

Algorithm 1 Outline Decode Algorithm

Our decoder converts a given instantiation of the array of records into a legal normalized postfix expression by writing down the rectangle IDs in the order given, and inserting the type of normalized chain of operators indicated by the op-type flag immediately following each rectangle number. The maximum length of each chain of operators given in the chain length field is allocated provided that the balloting property is not violated by doing so (i.e. if we are currently processing the \( i^{th} \) rectangle in the list the total number of operators in the postfix expression constructed so far must be less than or equal to \((i-1)\)). If the decoder reaches the end of the sequence of records and the resulting postfix string has
insufficient operators (less than \( n - 1 \)), extra operators are added on to the end of the string maintaining the normalized pattern of \( ...+ * + * ... \) etc. The decoder algorithm is presented in Algorithm 1. Below is an example showing an encoded string and its normalized postfix interpretation:

rectangle 5 rectangle 2 rectangle 4 rectangle 1 rectangle 3
op-type * op-type + op-type * op-type * op-type +
length 2 length 1 length 0 length 2 length 0

Postfix expression generated: 5 2 + 4 1 * + 3 +

4 The Genetic Algorithm

The simple genetic algorithm (GA) used here is derived from the model of [3] and is an example of a 'steady state' GA (based on the classification of [10]). It uses the 'weaker parent replacement strategy' first described in [1]. The GA applies the genetic operators to permutations of rectangle records. The fitness values are based on the amount of dead space produced in each floorplan defined by the individual normalized postfix expressions encoded in the population. The first parent is selected deterministically in sequence, but the second parent is selected in a roulette wheel fashion, the selection probabilities for each genotype being calculated using the following formula:

\[
\text{Selection Probability} = \frac{\text{Rank}}{\sum \text{Ranks}}
\]

where the genotypes are ranked according to the values of the dead space that they have produced, with the worse ranked 1, the second worse 2 etc. and the best ranked highest. The GA breeds permutations of records from which our decoder produces normalized postfix strings. These strings are, in turn, processed using a stack to generate a floorplan. For each horizontal or vertical combination, the shape curves are added as described in section 2. As each floorplan design is generated, the dead space is calculated and recorded. The initial population consists of random permutations of records with each basic rectangle represented exactly once in each list. The op-type flag for each record is set to '+' or '*' with equal probability, and the value in the length field is generated in two stages:

- Stage 1: length = 0, with a probability of 0.5
- Stage 2: if the length is not set to zero, then it is generated from a Poisson distribution with mean 3.

4.1 Genetic operators for permutations

We use three different mutation operators, one for each of the fields in our encoding structure (rectangle ID, op-type, and op length):

- M1 Swap positions of two rectangle IDs.
- M2 Switch op-type flag, + to * or vice versa.
- M3 Mutate length by incrementing or decrementing (i.e. length = length + 1, length = length - 1) with equal probability. (If length is zero we increment).
M1 and M2 produce an identical effect to the M1 and M2 operators defined in chapter 3 of [12]. Our M3 operator, on the other hand, is different from the M3 described in Wong et al, although its effect is similar. Our M3 will always produce a legal postfix expression. In the very early stages of our study we chose some non-problem specific permutation crossovers for testing and carried out some extensive comparisons to test the performance of four crossovers on our data sets. Overall Cycle Crossover (CX) [7] came out best and was selected for our study. Our implementation of CX is efficient and runs in linear time.

For our GA we choose a population size of $20n$, where $n$ is the number of modules in the problem. We chose this rather large size for our population because it matched the number of evaluations undertaken at each temperature by the simulated annealing algorithm of Young and Wong in their recent papers [14, 15]. The GA is halted when 40 generations have passed since the last improvement was recorded in the best-so-far. We set the limits on the aspect ratio for the final enclosing rectangle (i.e. the chip aspect ratio) to those used in 1988 by [12]: \(1/2 \leq \text{chip aspect ratio} \leq 2\). Unfortunately there does not appear to be a way to predict a chip aspect ratio in advance of a full evaluation of a postfix expression. To ensure that we obtain acceptable solutions, we simply reject all offspring in which the best point on the final shape curve does not correspond to a legal chip aspect ratio, and try generating them again.

5 Results

Table 1. Means of 5 replicate runs for % dead space of the genetic algorithm

| Problem | Shape flexibility | Genetic algorithm | | | | |
|---|---|---|---|---|---|
| | | mean % dead space | # evaluations | Mean run time mins:secs | Best % dead space |
| ami33 | 2 | 0 | 66734 | 2:33 | 0 |
| ami33 | 3 | 0 | 39183 | 1:33 | 0 |
| ami33 | 4 | 0 | 28919 | 1:11 | 0 |
| ami49 | 2 | 0.07 | 197934 | 11:42 | 0 |
| ami49 | 3 | 0 | 101405 | 6:08 | 0 |
| ami49 | 4 | 0 | 72913 | 4:31 | 0 |

The results for our GA are summarized in Table 1. We use the modules from the benchmark data sets *ami33* and *ami49* with 33 and 49 modules respectively. Column 2 gives the shape flexibility of the basic rectangles for the experiments in the rows, and columns 3 to 6 summarize the dead space obtained by the GA. The percentage of dead space for the best floorplan is recorded over five replicate runs of the GA in column 3, and the overall best for the five runs is shown in column 6. The total number of postfix expression evaluations averaged over the
five replicate runs is noted in column 4, and the average run time appears in
column 5. (The individuals rejected because of illegal chip aspect ratios are
counted amongst the evaluations.) As we mentioned earlier, we have not as yet
incorporated any approximations to reduce the number of 'corners' accumulated
during the floorplan construction process. Thus we could surely improve our run
times. From Table 1 it is clear that the GA frequently produces optimal results.
Figure 4 shows a typical floorplan found by our GA for *ami49*.

![Fig. 4. Ami49 floorplan with shape flexibility 2 and 0 % dead space](image)

6 Conclusions and Further Work

This paper describes a genetic algorithm (GA) which uses a novel encoding to
breed normalized postfix expressions for macro cell placement and area optimiza­
tion in VLSI floorplan design. Our experiments confirm that simple order based
genetic operators are effective in guiding the genetic search when our encoding
is used.

The only recent related work on VLSI, of which we are aware, is that reported
by Young and Wong [13–15]. Young and Wong use soft modules and normalized
postfix expressions with their simulated annealing algorithm. In the introduction
to [15] the authors report results of less than 1 % dead space for several MCNC
benchmarks including *ami33* and *ami49*. Although we achieve 0 % dead space
for the same problems, we must be cautious about making direct comparisons.
We do not as yet include wire length in our optimization, and this is usually
accounted for by Young and Wong.

We believe that our new approach has opened up some interesting possibil­
ities for genetic algorithms applied to slicing floorplans. Not only does our al­
gorithm produce excellent results but the slicing tree construction process used
to generate the floorplans has a run time scaling of $O(n)$ for hard modules and
$O(n \log n)$ for soft modules. This compares very favourably with the Bounded­
Sliceline Grid (BSG) that has been used in GAs and Simulated Annealing for
VLSI placement by several authors of recent papers. Work in progress is currently focused on incorporating some simple heuristics into our approach and extending the GA to allow rotation of the modules and super-modules. We also plan to incorporate a cost for wire length into the objective function in the near future, and extend to much larger problems.

References

Applications
Scalability and Efficiency of Genetic Algorithms for Geometrical Applications

Steven van Dijk, Dirk Thierens, and Mark de Berg

Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands, {steven, dirk, markdb}@cs.uu.nl

Abstract. We study the scalability and efficiency of a GA that we developed earlier to solve the practical cartographic problem of labeling a map with point features. We argue that the special characteristics of our GA make that it fits in well with theoretical models predicting the optimal population size (the Gambler's Ruin model) and the number of generations until convergence. We then verify these predictions experimentally. It turns out that our algorithm indeed performs according to the theory, leading to a scale-up for the total amount of computational effort that is linear in the problem size.

1 Introduction

Genetic Algorithms are generally known for their powerful problem solving capabilities and their flexibility in design, which makes them suitable for a wide range of problems. Besides solving a problem well, an algorithm should also be efficient: it should provide the solution in a reasonable amount of time. The balance between solution quality and response time should be good to make the algorithm practically useful. Furthermore, the algorithm should scale up well and predictably to larger problem instances.

The efficiency of a GA is basically the product of two factors: the population size and the number of generations it takes to converge. Of course these two factors also influence the quality of the solution. For example, the selection pressure determines the speed of convergence, but pushing too hard may be detrimental to the solution quality. Population sizing is also a trade-off between solution quality and computational resources, where one does not want the population to be too small and get bad solutions but also does not want it to be too high and make the algorithm inefficient. The mixing of building blocks determines the rate of success in finding the right combination of building blocks, but it is intimately bound with the issue of disruption which destroys them. For these reasons it is in general quite hard to predict the efficiency of a GA.

For abstract problems, however, interesting results have been obtained. For instance, it is known [1, 5–7] that for GA's with tournament selection and an uniformly scaled, additively decomposable fitness function, the number of generations until convergence is $O(\sqrt{l})$, where $l$ is the size of the problem instance. Furthermore, the Gambler's Ruin model [3] predicts that the optimal population
size for GA's under similar assumptions is $O(\sqrt{l})$ as well. Unfortunately these models have only been investigated using artificial problems. The goal of our paper is to investigate whether these models are also applicable to GA's solving more practical problems.

![Image of a labeled map](image_url)

**Fig. 1.** An example of a labeled map.

To this end we study the efficiency of a GA that solves the so-called map labeling problem: given a set of cities (points) on a map, try to label the cities (with their names, for instance) such that the number of free labels (not intersecting another label) is maximized - see Figure 1 for an example. Even in its simplest form (where one allows only four possible positions for a label, namely with one of its corners at the point to be labeled), the label-placement problem is NP-complete [4]. The GA solving this problem is taken from a previous paper [10], where we presented a framework for designing GA's to solve geographical problems. Our GA employed a novel operator called the geometrically local optimizer which gave the GA flexibility and the capability to be extended with additional constraints without suffering in performance.

In the present paper we study scalability and efficiency issues of our map labeling GA. In particular, we want to see whether the theoretical results on population sizing and convergence are applicable to our practical GA. To this end we first study the conditions under which these models are applicable. It turns out that, partly because of our use of the geometrically local optimizers, many of these conditions are indeed satisfied. This leads us to expect that the models are applicable, which we then verified experimentally. Indeed, our experiments show that both the optimal population size and the number of generations until convergence scale roughly as $O(\sqrt{l})$, leading to a linear scale-up for the total amount of computational effort. To our knowledge, this is the first work relating the theoretical models to a GA for a practical problem.

The rest of the paper is organized as follows. Some preliminary discussion on the theoretical models used is given in Section 2. The GA which was designed for the problem of map labeling is then described in Section 3. We then explain
in Section 4 how the geometrically local optimizers of our GA made it possible to apply the models. Section 5 is devoted to experimental verifications of the predictions, as well as a brief investigation in the role of selection pressure. We conclude the article in Section 6.

2 Preliminaries

A GA is efficient if the amount of work spent to solve a problem is close to the minimal amount of work needed to obtain that result. For the amount of work we can take the following: \( W = G^* \cdot n^* \), where \( G^* \) is the number of generations when the selection pressure is optimal and \( n^* \) is the optimal population size. Both factors \((G^* \text{ and } n^*)\) therefore determine the efficiency of the GA.

For \( G^* \) results known from literature \([1,5-7]\) indicate that for tournament selection and an uniformly scaled, additively decomposable fitness function the number of generations until convergence is \( G = O(\sqrt{l}) \), with \( l \) the length of the bitstring. A fitness function is additively decomposable if it can be formulated as the sum of the fitnesses of low-order partitions. We expect that the map labeling problem will behave similarly when we restrict ourselves to finding low-order relations (in other words, finding the optimum of a NP-complete problem requires finding high-order relations and thus is an infeasible approach).

The issue of determining \( n^* \) was investigated by Goldberg et al. \([2]\), who provided a model of the GA based on statistical decision making. Assuming that the GA would find the best solution if the search progressed in the right direction after the first generation, they acquired a population sizing equation. The drawbacks of this approach were that it did not model the way a GA can recover from decision errors and did not include a building block supply model. In the work of Harik et al. \([3]\) the model was extended by integrating the Gambler's Ruin model with the previous model and a building block supply model.

Statistical decision making views the search process as a propagation of building blocks through the population, assuming mixing is adequate. A building block is the element of a partition of genes with the highest marginal fitness contribution. Since decisions are made on the level of strings, a competition between a building block and another (sub-optimal) element from the same partition can result in the loss of the building block. Such an event is called a decision error. Since building blocks have a higher marginal fitness contribution, on average they will win the competition.

The Gamblers Ruin model is a one-dimensional random walk between absorbing barriers, corresponding with the loss of the building block (no building blocks left; we call this the depletion barrier) and the existence of the building block in all individuals (\( n \) building blocks in the population; we call this the saturation barrier). The walk starts at \( x_0 \), the number of building blocks in the initial population. Each competition advances the walk to either the saturation barrier (the string with the building block wins the competition) which increases the number of building blocks, or the depletion barrier (a decision error) which decreases the number of building blocks.
The initial number of building blocks is given by $x_0$. A simple building-block supply model assumes each element of the partition is equally likely to be (randomly) created. Therefore, if only one element of the partition is a building block, the number of genes in the partition is $k$ and the alphabet size is denoted by $A$, the expected number of building blocks in the initial population of size $n$ is $n/A^k$.

The probability $P_n$ of the gambler hitting the saturation barrier using a population of $n$ is:

$$P_n = \frac{1 - \left(\frac{1-p}{p}\right)^{x_0}}{1 - \left(\frac{1-p}{p}\right)^n}$$  \hspace{1cm} (1)

with

$$p = N\left(\frac{d}{\sigma_{BB} \sqrt{2 \cdot (m-1)}}\right).$$ \hspace{1cm} (2)

Here $p$ is the probability of making the right decision (from Goldberg et al. [2]), $N$ is the cumulative distribution function of a unit normal distribution, $d$ is the difference in marginal fitness contribution between the building block and its closest competitor, $\sigma_{BB}$ is the standard deviation of the fitnesses of all strings containing the building block and $m$ is the total number of building blocks in the string. Extracting $n$ from Equation 1, assuming a binary alphabet and approximating $p$ gives the following approximation of the population sizing equation (Harik et al. [3]):

$$n = -2^{k-1} \ln(1 - \alpha) \frac{\sigma_{bb} \sqrt{\pi (m-1)}}{d}$$ \hspace{1cm} (3)

where $\alpha$ is the expected fraction of gamblers that hit the saturation barrier.

The prediction for the scale-up behavior of the GA (which adheres to the assumptions of the Gambler’s Ruin model) is $n^* = O(\sqrt{l})$, where $n^*$ denotes the optimal population size and $l$ is the length of the problem.

### 3 Description of the GA

The GA for labeling point features using geometrically local optimizers was described in Van Dijk et al. (1999) [10]. Extensive experiments and implementation details can be found in Van Dijk et al. (1998) [9]. The map labeling problem consists of placing a label in one of four positions for a set of points such that the number of labels that do not intersect another label is maximized. A useful property of this problem is that the linkage of the building blocks seems fairly clear, since it is geometrically determined. We assumed the building blocks consist of good labelings of a city and several close-by cities. To make this more concrete, we define two points as rivals if their labels can intersect. A rival group consists of a certain point and its rivals. We assume the best labelings for rival groups are building blocks.

The GA consists of the following ingredients:
**Representation:** The GA represents the labeling by a string of numbers between one and four, indicating positions. Each city has an index which indicates its position in the string.

**Initialization:** For every point in every individual a random position is chosen.

**Fitness function:** The fitness function counts the number of non-intersecting labels.

**Selection scheme:** Tournament selection with a tournament size of two individuals was chosen, since it follows the assumptions which were made for the analysis (see Section 4).

**Recombination:** The crossover operator works by generating a set $S$ of points. The labelings of points in $S$ are transferred from the first parent to the first child. Labelings of points not in $S$ are taken from the other parent. The other child is constructed in a complementary fashion. The set $S$ is generated by randomly picking a point on the map and placing its rival group in the set. When the size of $S$ exceeds half the total number of points, the crossover operator stops picking points. Since disruption is minimized using the geometrically local optimizers (explained later), we have the probability of applying crossover set to one.

**Mutation:** No mutation in the traditional sense (assigning a random allele to a randomly chosen gene) is used.

**Geometrically local optimizer:** After crossover, points which came from $S$ but had a rival not in $S$ (and vice-versa) can have new conflicts. On these points the geometrically local optimizer is applied, which determines which position is free for the label and moves it there if it was intersecting another label.

---

![Comparison of different crossover operators. Average of three runs on the same map (created as explained in Section 5).](image)

---

1 In the previous paper describing this GA, we used the Elitist Recombination Scheme (see Thierens et al. [8]). We use tournament selection here since we do not want to deviate from the Gambler's Ruin model more than is strictly necessary, but we will show the same results apply for Elitist Recombination.
The crossover operator is non-standard to exploit the (geometrical) structure of the problem which determines the linkage of the building blocks. As such, good mixing can be obtained with relatively little disruption. This effect can be seen in Figure 2 where we compare this crossover (called “rival crossover” in the picture) against uniform crossover. The comparison was done using the number of label intersection tests, as this is a direct measure of computational effort. As can be seen, crossovers with rival groups suffer less disruption and thus less computational effort is needed to achieve good results. However, mixing is slightly less. Disruption is still quite heavy, which is as expected since rival groups by definition overlap. The effect of “repairing” the map after crossover using the geometrically local optimizer can also be seen. Disruption is minimized and both crossovers perform alike, although the outcome favors the crossover which mixes hardest: uniform crossover. Keeping in mind that a full-fledged map labeling problem also contains other constraints for which a linkage respecting crossover is necessary, we think using rival crossover in conjunction with geometrically local optimizers is best. For example, on a map where a city lies next to a river, preferably one does not want the river to lie between the city and its label. This depends on rival cities (since there needs to be room for the label in a good position) and a favorable configuration should be preserved by the crossover operator.

Another characteristic of the GA for map labeling is the fact that the fitness function can be kept simple since non-combinatorial constraints can be put in the geometrically local optimizer. This keeps the fitness function additively decomposable (see Section 2), which is a requisite for the models.

4 Applying Theory in Practice

In order to apply the theoretical models (for \( G^* \) and \( n^* \)), we need to check their assumptions and see if they hold for the case of the GA for map labeling described in Section 3. Both the models assume additively decomposable functions and tournament selection. Also we consider the case of uniformly scaled fitness functions (prediction: \( G = O(\sqrt{I}) \)), since each city can contribute equally to the overall fitness.

The assumptions of the Gambler’s Ruin model are as follows:

- **The fitness function is additively decomposable:** This holds since we can avoid placing penalty functions in the fitness function. Penalty functions can be avoided by placing additional constraints in the geometrical local optimizers instead of the fitness function. Of course, the structure of the problem allows an additively decomposable fitness function.

- **The GA progresses through a series of competitions:** This requirement is met by using tournament selection (each tournament is a competition).

- **The problem is \( k \)-defineable:** In the case of map labeling there are certain higher order relations which have to be considered in order to find the optimal solution. Since this is an NP-complete problem, that would not be practical
and we restrict ourselves to low order relations that are still able to find close to optimal solutions.

**Each partition is independent and therefore so are the gamblers:** We assume that the geometrically local optimizers make sure disruption of building blocks gets resolved when it appears, so partitions are quasi-independent.

**The order of partitions is fixed (k):** The partitions in the map labeling problem (rival groups) are not of fixed order, but the largest rival group can be taken as a conservative estimate. Moreover, the size of rival groups does not vary too much (on the dense maps used in the experiments, cities have on average about six or seven rivals).

**No disruption or formation of building blocks takes place:** In practice, some disruption takes place but is minimized due to the use of geometrically local optimizers with the effect that it has a marginal influence on the behavior of the algorithm. Application of the geometrical local optimizer can result in building block formation, but this will only relax the need for all building blocks to be present in the initial population.

**The barriers of the walk are absorbing:** A gambler that reached the saturation barrier will with very high probability stay in the proximity of it.

**All building blocks are present in the initial population:** Since building block formation is possible and indeed very likely to happen, this requirement can be dropped.

### 5 Experimental Verification

To verify the predictions from the Gambler’s Ruin model and to explore other efficiency issues, we gathered experimental data on randomly generated maps. The maps were embedded on a torus (to remove edge effects) and were made by repeated randomly selecting a location for a point and its label, making sure they did not overlap other labels. Then the labels were removed and the GA was used to find a placing for the labels again. This way we were certain that it was possible to place all labels without intersecting other labels and the optimum was always the number of cities on the map.

In the remainder of this section, functions will be fitted to data by using the Levenberg-Marquardt algorithm for non-linear least-squares fitting, with the data points weighed by their standard deviation. The notation “Data: \( x \times y \)” will be used to indicate that \( x \) different maps were generated and for each map \( y \) runs of the GA were done using different seeds for the random number generator. The data point will be the average of those \( x \times y \) runs.

**Gambler’s Ruin.** Since we can satisfy the assumptions (or at least not deviate from them significantly) we can apply the Gambler’s Ruin model to predict the behavior of the GA for map labeling. Equation 1 gives us the probability a certain gambler hits the saturation barrier. We have \( m \) gamblers running in parallel in the GA (where \( m \) is the number of labels) and an optimal placement of a label increments the fitness by one. Recall that \( P_n \) is the probability that the gambler will hit the saturation barrier. Hence,
\[ \text{fit}_n = m \cdot P_n. \] (4)

![Graph showing fit of gamblers ruin prediction to data. Data: 3 * 3.](image)

**Fig. 3.** Fit of gamblers ruin prediction to data. Data: 3 * 3.

Here \( \text{fit}_n \) is the expected fitness when a population of \( n \) is used. For a map of 1000 cities we plotted experimental data for different population sizes in Figure 3 (the whole figure was scaled to make the optimum 1, so comparing is easier). To this data we fitted Equation 4. The closeness of the fit shows that the Gambler’s Ruin model is indeed a good model for the GA for map labeling with geometrically local optimizers.

**Selection Pressure.** First we investigated the optimal selection pressure by using a large population size of 200 which, as will become apparent from the other experiments, is large enough. The termination criterion used for all runs was convergence of fitness (i.e. the average fitness becomes equal to the fitness of the best individual). The results are tabulated in Table 1. As can be seen in the table, the optimal tournament size seems to be two or three, since with larger tournaments the quality begins to drop too much.

<table>
<thead>
<tr>
<th>Tour. size</th>
<th>Ave. #gen.</th>
<th>SD of gen.</th>
<th>Ave. fit.</th>
<th>SD of fit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>80.64</td>
<td>10.35</td>
<td>.98548</td>
<td>.00483</td>
</tr>
<tr>
<td>3</td>
<td>63.44</td>
<td>11.09</td>
<td>.98304</td>
<td>.00535</td>
</tr>
<tr>
<td>4</td>
<td>63.2</td>
<td>16.25</td>
<td>.97812</td>
<td>.00666</td>
</tr>
<tr>
<td>5</td>
<td>53.64</td>
<td>11.80</td>
<td>.97176</td>
<td>.00737</td>
</tr>
</tbody>
</table>

**Table 1.** Influence of different selection pressures. Table shows tournament size, average number of generations and its standard deviation, average fitness found (fraction of optimum) and its standard deviation. Data: 5 * 5.
Population Sizing. We performed experiments to determine the critical population size. After running the GA for different population sizes (data: \(3 \times 3\)), we fitted Equation 4 to deduce the critical population size for obtaining 97% of the optimum for a map of a certain size. Thus, for each map size a figure like Figure 3 is produced from which the critical population size can be read. The results are plotted in Figure 4 (left), where a square root function is fitted to verify the prediction of the Gambler’s Ruin model. This prediction, which states that the relation between critical population size \(n^*\) and problem length \(l\) should be \(n^* = O(\sqrt{l})\), is confirmed. Also it is clear that very small population sizes are sufficient. The number of generations needed to obtain the critical population size (shown in Figure 4 (right) with a fit to a square root) was obtained in a similar fashion, using a fit of the function \(f(x) = -1/x\) (instead of Equation 4) to give a decent interpolation. Again the prediction of \(G^* = O(\sqrt{l})\) is confirmed.

Since the amount of work \(W = G^* \cdot n^*\) it follows that \(W = O(l)\) (the amount of work done scales up linearly with the problem size).

![Graph showing critical population size and runtime for GA](image)

Fig. 4. Experimental data with fit to predicted function for tournament selection and the elitist recombination selection scheme. Data: \(3 \times 3\). Left: critical population size for a quality of 97% of the global optimum. Right: runtime in number of generations of GA when using critical population size.

We also tried the same experiments with elitist recombination (used in our previous paper [10]) instead of tournament selection as the selection scheme. The results are also shown in Figure 4 and show the same scale-up behavior. It should be noted that elitist recombination succeeds in finding better solutions than tournament selection.

6 Conclusion

This paper investigated scalability and efficiency issues for a practical GA. We used the GA for geographical applications from a previous paper applied to the
NP-complete problem of placing labels on a map. Predictions for the number of generations needed to converge and the optimal population size were taken from theoretical studies.

We were able to satisfy the assumptions of the models by using the geometrically local optimizers which kept the fitness function additively decomposable, minimized disruption and provided a rich source of building blocks. We verified the predictions with experiments and the agreement was good. As far as we know, this is the first time these models (most notably the Gambler’s Ruin model for the optimal population size) have been applied for the analysis of a real-world GA.

The GA for map labeling gives good solutions, is flexible and extendible to other problem instances and has been shown to be efficient with linear scale up behavior. We hope these results also apply for other GA’s which are designed in a similar fashion: exploiting the geometric structure by using a linkage respecting crossover and geometrically local optimizers.

References

Genetic Optimization of the EPR Spectral Parameters: Algorithm Implementation and Preliminary Results

Bogdan Filipič and Janez Štrancar

1 Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
bogdan.filipic@ijs.si

2 Department of Solid State Physics, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
janez.strancar@ijs.si

Abstract. Electron paramagnetic resonance (EPR) spectroscopy is a nondestructive technique suitable for inspection of biological systems. Characterization of such a system is much more reliable when relevant spectral characteristics are extracted from a biophysical model of the system. To tune the model parameters, stochastic optimization techniques are used more and more frequently. Since many single-point algorithms require time-consuming preparation of promising starting points to produce reasonable results, we have addressed the problem with population-based, evolutionary computation approach. We have applied a genetic algorithm to the EPR spectral parameter optimization and evaluated it on synthetic spectra. Preliminary numerical experiments show the new approach is beneficial in that it produces satisfactory results and reduces the time a spectroscopist spends for navigating the optimization process.

1 Introduction

To obtain relevant experimental information about the structure and dynamics of complex biological systems in a nondestructive way, spectroscopic techniques are widely used. Among them is electron paramagnetic resonance (EPR) spectroscopy which exploits the physical phenomenon of absorption of microwave radiation by paramagnetic molecules or ions subjected to an external magnetic field. In combination with labeling based on nitroxide spin probes, EPR spectroscopy is especially suitable for studying cell membranes [3,9]. It can detect alterations caused by biologically active substances, such as changes of membrane fluidity and permeability, and the involvement of pathological conditions, such as acute phase, cancer, etc. However, the recorded EPR spectra provide much more reliable and biologically meaningful information when characterized by means of computer-aided spectrum simulation rather than by simple measurements of spectral peak characteristics.

It was demonstrated that under certain preconditions the so-called motional-restricted fast-motion approximation satisfactorily describes the EPR spectra
recorded in the complex geometry of tissues or cell suspensions [10]. Exploration of such systems reveals a number of domains which reflect the heterogeneity of biological membranes [4]. The parameters of the biophysical model used for spectrum simulation in the restricted fast-motion approximation provide data about ordering, dynamics and the polarity at various locations in different membrane domains. Furthermore, the approach seems to be a good compromise between the exact theoretical consideration, empirical knowledge and computational requirements in order to perform characterization of an experimental EPR spectrum in a reasonable period of time.

To characterize an experimental spectrum via computer simulation, parameters of the involved biophysical model have to be tuned in order for the simulated spectrum to match with the experimental one. The membrane heterogeneity results in a number of parameters, which are, in addition, partially correlated. For these reasons, the power of stochastic algorithms is widely used in EPR spectral parameter optimization. Unfortunately, traditional single-point procedures require a spectroscopist to initially provide good starting parameter values which may be very time-consuming. An automated optimization procedure would allow the spectroscopist to focus on experiments and derive as many different insights into complex biological systems as possible. With this motivation, we employed an evolutionary population-based technique to EPR spectral parameter optimization. We integrated a generational real-coded genetic algorithm with the spectrum simulation model and carried out a preliminary experimental verification on synthetic EPR spectra consisting of various number of domains. Initial results are promising both in view of accuracy and reduction of the time spent by a spectroscopist to navigate the optimization procedure.

Subsequent sections of the paper present the EPR spectrum simulation model, the genetic algorithm applied in spectral parameter optimization, and the results obtained in numerical experiments. The paper concludes with a summary of results and issues for further work.

2 The EPR Spectrum Simulation Model

With spin-labeled cell suspension or tissue we encounter superimposed EPR spectra consisting of several spectral components, also called domains. They arise from various compartments (solution, membrane domains, outer and inner layers of the membrane, aggregates, etc.) with different physical characteristics. In the restricted fast-motion approximation the calculation of every spectral component involves three steps. The procedure is discussed in detail in [10]. Here, the procedure is briefly reviewed with an emphasis on the role of spectral parameters.

In the first calculation step, the magnetic interaction tensors are averaged over fast stochastic restricted rotational motions of the spin-probe molecules. This is done according to the order parameter, $S$, for assumed averaged orientation of the nitroxide group relative to the membrane normal vector. In the same calculation step, the polarity correction factors $p_A$ and $p_g$ are used to describe
the effect of the neighboring electric fields which influence the electron density distribution of the spin-probe molecules [2].

The second step includes the derivation of the Lorentzian lineshape using the motional narrowing approximation [8]. Two parameters are applied: the rotational correlation time, $\tau_c$, and the broadening constant, $W$. While $\tau_c$ describes the rate of motion in the approximation, $W$ arises primarily due to unresolved hydrogen superhyperfine structure together with minor paramagnetic impurities (for example, due to usually present oxygen), external field nonhomogeneities, modulation effects, etc.

In the third step, the convolution of the resonant field distribution with the first derivative of the lineshape is calculated for all spectral lines which gives us the signal intensity. The proportion of a spectral domain in the spectrum is denoted by the weighting factor $d$.

In summary, the spectrum simulation requires the following parameters for each spectral domain:

- order parameter $S$,
- rotation correlation time $\tau_c$,
- broadening constant $W$,
- polarity correction factors $p_A$ and $p_g$,
- weighting factor $d$.

Given the values of spectral parameters, the model produces a simulated EPR spectrum. For the sake of optimization, an objective function is introduced which measures the goodness of fit of the simulated spectrum with the experimental one. The measure is the reduced $\chi^2$, i.e. the sum of the squared residuals between the experimental and simulated spectra divided by the squared standard deviation of the experimental points, $\sigma$, and by the number of points in the experimental spectrum, $N$:

$$\chi^2 = \frac{1}{N} \sum_{i=1}^{N} \frac{(y_i^{\text{exp}} - y_i^{\text{sim}})^2}{\sigma^2}. \quad (1)$$

The standard deviation $\sigma$ is assessed numerically from the experimental points of those parts where the simulated spectrum derivatives are close to zero. This is usually at both ends of the spectrum.

3 A Genetic Algorithm for EPR Spectral Parameter Optimization

Genetic algorithms can be viewed as computer simulations of evolutionary phenomena known from biological systems, such as genetic recombination and survival of the fittest population members. Genetic algorithms apply these concepts as search heuristics to efficiently explore large solution spaces. Population-based search and robustness make genetic algorithms powerful problem-solving techniques applicable to a broad spectrum of tasks, including numerical optimization [6].
In this study, a genetic algorithm is used to enhance the characterization of experimental EPR spectra. More precisely, it tunes the parameters of the spectrum simulation model. To evaluate a parameter setting, the algorithm activates the model which produces a simulated spectrum, and then evaluates the goodness of fit of the simulated spectrum with the experimental spectrum according to Equation 1. A generational genetic algorithm is applied which includes the fitness-proportional selection [1]. It is an elitist algorithm preserving a certain number of the best solutions from the previous generation in the current population.

EPR parameter settings processed by the genetic algorithm are represented as real-valued vectors. The number of vector components depends on the number of the encountered spectral domains. The number of domains is not a subject of optimization and needs to be provided by the user before the optimization process. For a single domain spectrum, five parameter values need to be determined and the solution vector is of the form

\[ \mathbf{q} = (S, \tau_c, W, p_A, p_g) \]  

where the vector components denote the model parameters listed in Section 2. The weighting factor \( d \) equals to 1 since only one spectral domain is considered. In case of two domains, the solution vector is

\[ \mathbf{r} = (S_1, \tau_{c1}, W_1, p_{A1}, p_{g1}, d_1, S_2, \tau_{c2}, W_2, p_{A2}, p_{g2}) \]  

and \( d_2 = 1 - d_1 \). In general, for an EPR spectrum with \( k \) domains, \( 6k - 1 \) spectral parameters are subject to optimization, and the weighting factor for \( k \)-th domain is

\[ d_k = 1 - \sum_{i=1}^{k-1} d_i. \]  

For each spectral parameter the interval of possible values and the search step (resolution) are set in advance by the user according to the preferences and physical limitations. The parameters can only take values satisfying these constraints. The solution vectors are varied by multi-point crossover and uniform mutation, and the mutation operator is restricted to alter a vector component by increasing or decreasing it for a random number of resolution steps within the prescribed search interval. This ensures that during the optimization process the parameter values remain limited and discretized as specified by the user.

### 4 Numerical Experiments and Results

The aim of preliminary numerical experiments was to compare the performance of genetic optimization with that of parameter optimization techniques previously applied in EPR spectroscopy of biological membranes [10]. These include downhill simplex [7] and the Monte Carlo algorithm using the Metropolis criterion for accepting new parameter settings [5].
The study was performed on synthetic spectra artificially contaminated with noise to represent "experimental" spectra. The noise level was 5% of the amplitude of the original synthetic spectra. Three test samples were provided consisting of one, two and three spectral domains, and therefore requiring 5, 11 and 17 parameters to be optimized, respectively. Each spectrum consisted of 1024 points. An example of a test experimental spectrum generated for the purpose of numerical experiments is shown in Fig. 1.

![Synthetic EPR spectrum](image)

**Fig. 1.** A synthetic EPR spectrum contaminated with noise, considered as an experimental spectrum in numerical experiments

Parameter space to be searched during the optimization process was prescribed in advance by a spectroscopist. The applied search intervals and resolutions for all spectral parameters are summarized in Table 1. For spectra composed of two or three domains, identical constraints were applied to all domains.

<table>
<thead>
<tr>
<th>Spectral parameter</th>
<th>Unit</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>order parameter $S$</td>
<td>-</td>
<td>0.02</td>
<td>1</td>
<td>0.005</td>
</tr>
<tr>
<td>rotation correlation time $\tau_r$</td>
<td>ns</td>
<td>0.1</td>
<td>3</td>
<td>0.05</td>
</tr>
<tr>
<td>broadening constant $W$</td>
<td>mT</td>
<td>0.02</td>
<td>0.3</td>
<td>0.005</td>
</tr>
<tr>
<td>polarity correction factor $p_A$</td>
<td>-</td>
<td>0.8</td>
<td>1.2</td>
<td>0.001</td>
</tr>
<tr>
<td>polarity correction factor $p_g$</td>
<td>-</td>
<td>0.9998</td>
<td>1.0002</td>
<td>0.0000002</td>
</tr>
<tr>
<td>weighting factor $d$</td>
<td>-</td>
<td>0.01</td>
<td>0.99</td>
<td>0.005</td>
</tr>
</tbody>
</table>
hybridization were at the level of the best results known so far for the 1-domain and 2-domain test problems, and significantly better for the 3-domain problem. The figures are given in Table 3.

Table 3. Results of spectral parameter optimization

<table>
<thead>
<tr>
<th>Optimization method</th>
<th>1 domain</th>
<th>2 domains</th>
<th>3 domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downhill simplex</td>
<td>1.19±0.01</td>
<td>1.16±0.12</td>
<td>1.26±0.21</td>
</tr>
<tr>
<td>Monte Carlo + downhill simplex</td>
<td>1.23±0.02</td>
<td>1.24±0.16</td>
<td>1.31±0.22</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>1.47±0.08</td>
<td>1.56±0.09</td>
<td>1.82±0.29</td>
</tr>
<tr>
<td>Genetic algorithm + downhill simplex</td>
<td>1.18±0.01</td>
<td>1.17±0.03</td>
<td>1.04±0.14</td>
</tr>
</tbody>
</table>

In addition, the fact that hybrid genetic algorithm successfully performs optimization of the EPR spectral parameters from random initial settings is beneficial from the practical point of view. It prevents a spectroscopist from time-consuming preparation of good starting points and allows him/her to focus on the spectrum characterization itself.

5 Conclusions

A genetic algorithm was implemented to perform the EPR spectral parameter optimization which is an important step of characterization of complex biological systems with EPR spectroscopy. The algorithm was verified on synthetic spectra and its results were compared to that of human-navigated single-point search techniques. The hybrid approach consisting of genetic search and local optimization produced superior results and its effectiveness was clearly demonstrated on the most complex test problem. An additional advantage of the approach is that it releases the spectroscopists from manual preprocessing of spectral parameters which is unavoidable when using less powerful search techniques.

The study will continue with a systematic numerical experimentation covering a larger set of both synthetic and real experimental spectra, as well as alternative, physically meaningful measures for the goodness of fit. More intensive hybridization of the genetic search will be considered and the algorithm parameters refined to balance between performance, computational cost and applicability for the users who are not specialists in genetic algorithms.

Acknowledgement

The work presented in this paper was supported by the Ministry of Science and Technology of the Republic of Slovenia. The authors are grateful to anonymous referees for their comments and suggestions.
References

Fitting Fluorescence Spectra with Genetic Algorithms

J.A. Hageman¹, R. Wehrens¹, R. de Gelder², W. Leo Meerts³, and L.M.C. Buydens¹*

¹ Laboratory of Analytical Chemistry
² Department of Inorganic Chemistry
³ Department of Molecular and Laser Physics
*Corresponding author

University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Abstract. Up until now it was not possible to automate the interpretation of spectra in which both frequencies and intensities contain (chemical) information. If the possibility of shifting peaks exist, point-wise comparisons of intensities at specific wavelengths is no longer adequate because wrong peaks could easily be compared. We show that with a suitable fitness function, which is generally applicable, spectra with shifted peaks can be solved using a standard GA. The method is very robust and illustrated using laser induced fluorescence spectra taken from Indole, Benzimidazole and 4-Aminobenzonitrile.

1 Introduction

Information on the identity, conformation or other physico chemical properties of a chemical sample is often obtained by one of several forms of spectroscopy. These techniques lead to a spectrum, which indicates the amount of energy absorbed or emitted (y-axis) and the frequencies at which this occurs (x-axis). Interpretation of these spectra then leads to the desired information. Several attempts to automate spectrum interpretation exist, but up to now they have only been successful in cases where the characteristic frequencies were known beforehand. Examples are originating from ¹H- and ¹³C-NMR experiments [1, 2], IR-spectroscopy, UV-spectroscopy [3, 4], powder diffraction data [5–8] and fluorescence spectroscopy [9]. An often used procedure for the interpretation of spectra is the minimization of the differences between a theoretical and an experimental spectrum. The assumption is that parameter values, leading to a theoretical spectrum that is identical to the measured spectrum, are the correct values one is interested in. This procedure works correctly if only intensities of peaks need to be compared. As soon as peaks can change their position, these automatic approaches fail. This is the case in those flavors of spectroscopy where
the location of the peaks is related to the physical properties of the sample, e.g., nuclear magnetic resonance (NMR) and laser induced fluorescence (LIF). A pointwise comparison of peaks is in these cases no longer sensible, because the wrong peaks could easily be compared.

In the present paper, a method for the automated interpretation of such spectra is given. This method applies a Genetic Algorithm (GA) [10] with a fitness function able to deal with peak shifts. This is illustrated for the laser induced fluorescence (LIF) spectra of three organic molecules.

2 Laser Induced Fluorescence Spectroscopy

With LIF spectroscopy, it is possible to obtain information on the geometry of molecules in the form of rotational constants. These constants give information on intra- and intermolecular bond lengths and their changes upon excitation. Using a rigid asymmetric rotor Hamiltonian a theoretical spectrum can be calculated. The Hamiltonian describes the free rotation and accompanying energies and wave functions of a molecule in the gas phase. A full description of this Hamiltonian is beyond the scope of this paper but details can be found elsewhere [11]. The model is controlled by 12 parameters.

1. Six rotational constants. Three parameters \( A'', B'', C'' \) describing the ground state and three parameters \( \Delta A, \Delta B, \Delta C \) describing the between the ground and excited state values \( \Delta A = (A' - A'') \) etc. Here the double and single primes label the ground and excited state constants. These parameters are responsible for the location of (groups of) peaks and cause many peaks to shift left or right.

2. Three parameters \( T_1, T_2 \) and \( W \) that describe the relative intensities of the transitions between energy levels in a molecule described by the Hamiltonian.

3. Three further parameters: the line width \( \Delta \nu \), a frequency shift parameter \( \nu \) and one parameter \( \theta \) describing changes upon excitation of the molecule under investigation.

Minimization of the difference between a theoretical spectrum obtained with this model and an experimental spectrum should yield optimal values for the 12 parameters and in particular for the 6 rotational constants. In the next section, an adequate difference function is discussed. The power of this method is demonstrated for the LIF spectra of Indole, Benzimidazole and 4-Aminobenzonitril (4-ABN), which were discussed in Refs. [11,12] and are shown in Figure 1.

3 Evaluation function

The similarity between the calculated spectrum and experimental spectrum has to be expressed in a single number if it is going to be used in combination with GAs. Several methods can be used for this purpose. In cases where only intensities of peaks can vary and frequencies (peak positions) remain constant, an
Fig. 1. High resolution LIF spectra of Indole, Benzimidazole and 4-ABN. In all cases the frequencies are relative to 0.0, according to Ref. [11] and Ref. [12], which corresponds to the rotational free electronic transition. The intensity on the vertical scale is in arbitrary units.

evaluation function based on a root-mean-square error (RMS) or a correlation will probably suffice for most applications [3,6–8]. If not only intensities but also frequencies contain information, comparison methods should include a comparison of the neighborhood to deal with peak shifts [5]. Another possibility would be to identify important and/or characteristic peaks frequencies and define a similarity measure with these peaks only.

Our initial attempts clearly demonstrated the inability of an RMS-type of evaluation function to recognize the similarity between spectra originating from nearly identical sets of parameters. Other approaches, based on peak picking and minimizing the distance to neighboring peaks in both spectra, failed as well. Since the relative position of peaks, in this application in particular, can change dramatically, one is never sure if the correct peak pairs are compared. With these types of evaluation functions, similar spectra with shifts in peak positions will not properly be recognized as similar.

To correctly compare this kind of spectra one should, in some way, compare the neighborhood of a given frequency. On way to achieve this is by using a cross correlation function as given in Eq. (1):

\[ C_{fg}(r) = \frac{\sum_{x=k}^{x=0} f(x) \cdot g(x + r)}{\| f \| \cdot \| g \|} \]
Here \( f(x) \) and \( g(x) \) are spectra \( f \) and \( g \) with equal length \( k \), and the term in the denominator is a normalization constant. Eq. (1) compares two spectra with a shift \( r \) added to one of them. When calculating several \( C_{fg} \) with different \( r \) values, it can be seen that similar spectra do not have their maximal value for \( C_{fg}(0) \), as depicted in Figure 2. Here, the spectrum of Indole is compared with itself, two slightly modified spectra of Indole and a spectrum of another compound. The difference between the two different compounds is clearly visible, while the similarity between the three spectra is quite high.

\[
-100 \quad -50 \quad 0 \quad 50 \quad 100
\]

Fig. 2. Correlogram of the calculated spectrum of Indole. Autocorrelogram: solid line, Crosscorrelogram: dashed line (\( \Delta \) changed by 1.0 MHz), dash dotted line (\( \Delta A \) changed by 1.0 MHz) and dotted line (calculated spectrum of Benzimidazole).

Larger shifts (\( r \)-values) can be penalized by a weight function \( w(r) \). Several weight functions have been tested but the simple triangular function (Eq.(2)) worked best. Another advantage is that it is only controlled by one parameter, the base width of the triangle:

\[
w(r) = 1 - \frac{|r|}{l} \tag{2}
\]

Combining Eqs. (1) and (2) yields Eq. (3), the area under the weighted cross-correlation function:

\[
C_{fg}^{ws} = \frac{\sum_{r=-l}^{r=l} C_{fg}(r)w(r)}{\sqrt{\sum_{r=-l}^{r=l} C_{ff}(r) * w(r) \cdot \sum_{r=-l}^{r=l} C_{gg}(r) * w(r)}} \tag{3}
\]
For two identical spectra $C^*_{fg}$ is 1 and for distinctly different spectra $C^*_{fg}$ is close to zero.

A comparison of different evaluation function values is given in Table 1. It shows the inability of the RMS and, to a lesser extent, the correlation coefficient to recognize the similarity between the Indole spectra. All Indole spectra used in Table 1 are calculated with nearly identical sets of parameters and should therefore be recognized as similar which only is accomplished for our evaluation function. The correlation coefficient seems to work well, but is not good at discriminating between "very similar" and "similar" (data not shown).

<table>
<thead>
<tr>
<th>Evaluation function</th>
<th>$A'$</th>
<th>$\Delta A$</th>
<th>Benzimidazole</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS</td>
<td>1013.509</td>
<td>3389.386</td>
<td>95812.967</td>
</tr>
<tr>
<td>CORR</td>
<td>0.994</td>
<td>0.933</td>
<td>0.150</td>
</tr>
<tr>
<td>F</td>
<td>1.000</td>
<td>0.996</td>
<td>0.330</td>
</tr>
</tbody>
</table>

Evaluation values calculated with different evaluation function (CORR = correlation coefficient, $F = $ proposed function (left column). The calculated spectrum of Indole is compared with two nearly identical spectra of Indole ($A'$ changed by 1.0 MHz, second column and $\Delta A$ changed by 1.0 MHz, third column), and the spectrum of Benzimidazole (right column).

The final evaluation function used in the GA calculations is defined as:

$$F = 100 \times (1 - C^*_{fg})$$  \hspace{1cm} (4)

and its value is minimized.

A more detailed discussion and comparisons with other methods for the assessment of similarity between 1-dimensional spectra, can be found in the work of De Gelder et al. [13].

4 Experimental

The spectra of Indole, Benzimidazole and 4-ABN are shown in Figure 1. The spectra of Indole and Benzimidazole contain 65536 equidistant data points and the spectrum of 4-ABN contains 40972 data points. All 12 parameters were coded as 10-bit gray binary numbers. The rotational constants in the excited state are expressed on the string as the difference with the ground state. $T_2$ is coded on the string as $\alpha$, with $T_2 = \alpha^* T_1$ and $\alpha >1$. The calculated spectra always contain the same number of data points as the corresponding experimental ones. The optimal settings of the GA were determined in preliminary experiments, based on previous experience, and are shown in Table 2.

The optimal size of the neighborhood in Eq. (2) has been established from several experiments. The optimal value for $l$ was 100 data points. A larger range
Table 2. GA settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum number of generations</td>
<td>500</td>
</tr>
<tr>
<td>population size</td>
<td>300</td>
</tr>
<tr>
<td>elitism</td>
<td>150</td>
</tr>
<tr>
<td>crossover type</td>
<td>two-point-crossover</td>
</tr>
<tr>
<td>crossover probability</td>
<td>0.85</td>
</tr>
<tr>
<td>mutation type</td>
<td>new random value within boundaries</td>
</tr>
<tr>
<td>mutation probability</td>
<td>0.05</td>
</tr>
<tr>
<td>selection type</td>
<td>probabilistic</td>
</tr>
<tr>
<td>fitness type</td>
<td>raw&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<sup>a</sup> Fitness value increases inversely proportional with evaluation value of a string.

also results in a correct solution but leads to longer run times. For a significantly smaller range no correct solution is obtained, indicating that the inclusion of neighborhood information is crucial. After establishing the optimal settings, the experimental spectra of Indole, Benzimidazole and 4-ABN were fitted using boundary constraints as given in Table 3. The duration of a run has been set to 500 generations, long enough to converge to a minimum. All runs were repeated 5 times with different random generator seeds.

The robustness of the GA method was investigated in a number of runs. Synthetic spectra of Indole and Benzimidazole were modified with different levels of normally distributed (white) noise, increased line widths and a combination of these two factors.

All GA calculations were performed with the GA library PGAPack version 1.0 [14], which can run on parallel processors. PGAPack and the evaluation function are written in ANSI-C, the rigid asymmetric rotor Hamiltonian function was written in Fortran. All calculations were performed on a Sun-Ultra-Enterprise-10000 with 24 processors each running at 333 MHz. With 16 processors, the average runtime was about half an hour for 500 generations and 65536 data points. In practice, this runtime can be reduced drastically, because often runs converged to their final solution long before the maximum number of generations was reached.

5 Results and Discussion

Table 4 shows the 12 parameters for all four experimental spectra resulting from the GA, together with the results reported by Ref. [11] (Indole and Benzimidazole) and Ref. [12] (4-ABN), using the manual methods. The values obtained with our present GA approach are in close agreement with these previous results. Results from a GA using an evaluation function based on the RMS did not lead to valid results at all. The correlation function leads to improved results, but still was not able to fit all 12 parameters. A comparison of the error landscapes
Table 3. Boundary constraints for all 12 parameters used for Indole, Benzimidazole and 4-ABN.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Indole and Benzimidazole</th>
<th>4-ABN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A''$</td>
<td>3800 - 4200</td>
<td>5000 - 6000</td>
</tr>
<tr>
<td>$B''$</td>
<td>1400 - 1800</td>
<td>800 - 1200</td>
</tr>
<tr>
<td>$C''$</td>
<td>800 - 1400</td>
<td>600 - 1000</td>
</tr>
<tr>
<td>$T_1$</td>
<td>1 - 6$^b$</td>
<td>1 - 6</td>
</tr>
<tr>
<td>$T_2$</td>
<td>1.5 - 5</td>
<td>1.5 - 5</td>
</tr>
<tr>
<td>$W$</td>
<td>0 - 1</td>
<td>0 - 1</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$0° - 90°$</td>
<td>$90°$, fixed$^c$</td>
</tr>
<tr>
<td>$\nu$</td>
<td>$-300 - 300°$</td>
<td>$-5000 - 5000$</td>
</tr>
<tr>
<td>$\Delta A$</td>
<td>$-200 - 0$</td>
<td>$-400 - 400$</td>
</tr>
<tr>
<td>$\Delta B$</td>
<td>$-50 - 0$</td>
<td>$-100 - 100$</td>
</tr>
<tr>
<td>$\Delta C$</td>
<td>$-50 - 0$</td>
<td>$-100 - 100$</td>
</tr>
<tr>
<td>$\Delta \nu$</td>
<td>$10 - 40$</td>
<td>$10 - 90$</td>
</tr>
</tbody>
</table>

$^a$Rotational constants in the ground state are indicated by $A''$, $B''$ and $C''$. Rotational constants in the excited state are given by their deviations from the ground state ($\Delta A$, $\Delta B$ and $\Delta C$). $\nu$ is line width of the Lorentzian peaks. Rotational constants, $\nu$ and $\Delta \nu$ are in MHz, $T_1$ and $T_2$ in K.

$^b$Range is 2 - 8 for the spectrum taken from Benzimidazole.

$^c$The frequency of the origin ($\nu$) is set to zero. The area of deviation is taken to be ± 10% of the reported value from Refs. [11] [12]

$^d$The frequency of the origin ($\nu$) is set to zero. The area of deviation is taken to be ± 10% of the reported value from Refs. [11] [12]

$^e$The frequency of the origin ($\nu$) is set to zero. The area of deviation is taken to be ± 10% of the reported value from Refs. [11] [12]

$^f$The frequency of the origin ($\nu$) is set to zero. The area of deviation is taken to be ± 10% of the reported value from Refs. [11] [12]

The GA using the proposed evaluation function was able to find the correct solution for the Indole and Benzimidazole spectra in all 5 replicated runs. The correct solution for the 4-ABN data was found in only 2 of the 5 cases, as shown in Figure 4. The cause of the reduced reproducibility of the 4-ABN run is probably the larger boundary constraints, which makes it more difficult for the GA to locate the correct solution.

The absolute evaluation function values did not reach the same level for the 3 compounds. This is due to the noise level, line width and total number of data points in a particular spectrum. High noise levels intrinsically give rise to large
Fig. 3. Difference in error landscape between a RMS-based (left), correlation coefficient (middle) and our new evaluation function (right).

values of the evaluation function. However, the minimum obtained in each case is the global minimum, irrespective of the absolute evaluation value.

The addition of noise and increased line widths to the spectra led to an increase in evaluation value. Although the quality of the solutions appeared to deteriorate, the rotational constants were hardly influenced by the elevated noise levels. The deviations were mostly found in $T_1$, $T_2$ and in $\theta$. Because one is mostly interested in the rotational constants the method can be considered quite robust for the determination of these parameters.

A decrease of the number of data points (where the overall frequency range is kept constant) only shows an effect on the Benzimidazole spectrum. For a smaller number of data points, the solutions become worse. This is due to the

Fig. 4. Progression of the best solution during a run for Indole, Benzimidazole and 4-ABN.
Table 4. Results from GA runs for Indole, Benzimidazole and 4-ABN.\(^a\)

<table>
<thead>
<tr>
<th></th>
<th>Indole</th>
<th>Benzimidazole</th>
<th>4-ABN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(^a)</td>
<td>3879.8</td>
<td>3880.7</td>
<td>3929.0</td>
</tr>
<tr>
<td></td>
<td>3930.5</td>
<td>5579.7</td>
<td>5579.3</td>
</tr>
<tr>
<td>B(^a)</td>
<td>1637.0</td>
<td>1637.5</td>
<td>1679.2</td>
</tr>
<tr>
<td></td>
<td>1679.5</td>
<td>990.23</td>
<td>990.26</td>
</tr>
<tr>
<td>C(^a)</td>
<td>1151.3</td>
<td>1152.1</td>
<td>1177.1</td>
</tr>
<tr>
<td></td>
<td>1176.7</td>
<td>841.45</td>
<td>841.39</td>
</tr>
<tr>
<td>T(_1)</td>
<td>2.2</td>
<td>1.50</td>
<td>5.63</td>
</tr>
<tr>
<td></td>
<td>4.88</td>
<td>2.63</td>
<td>3</td>
</tr>
<tr>
<td>T(_2)</td>
<td>7.93</td>
<td>5.03</td>
<td>21.52</td>
</tr>
<tr>
<td></td>
<td>20.0</td>
<td>4.56</td>
<td>-</td>
</tr>
<tr>
<td>W(^b)</td>
<td>0.1</td>
<td>0.22</td>
<td>0.423</td>
</tr>
<tr>
<td></td>
<td>0.42</td>
<td>0.84</td>
<td>-</td>
</tr>
<tr>
<td>t(^o)</td>
<td>37.4(^o)</td>
<td>±38.3</td>
<td>22.0(^o)</td>
</tr>
<tr>
<td></td>
<td>±22.0(^o)</td>
<td>0(^o)</td>
<td>0(^o)</td>
</tr>
<tr>
<td>(\nu)</td>
<td>0.78</td>
<td>0.0</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>-1.61</td>
<td>0.0</td>
</tr>
<tr>
<td>(\Delta A)</td>
<td>-134.70</td>
<td>134.66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-155.62</td>
<td>-155.70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-315.54</td>
<td>-316.61</td>
<td></td>
</tr>
<tr>
<td>(\Delta B)</td>
<td>-18.08</td>
<td>-17.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-15.30</td>
<td>-15.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.66</td>
<td>10.849</td>
<td></td>
</tr>
<tr>
<td>(\Delta C)</td>
<td>-20.72</td>
<td>-20.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-21.41</td>
<td>-21.31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.29</td>
<td>0.095</td>
<td></td>
</tr>
<tr>
<td>(\Delta \nu)</td>
<td>16.2</td>
<td>20.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19.33</td>
<td>19.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16.16</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Evaluation Values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>4.1815</td>
<td>9.956</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6460</td>
<td>0.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.200</td>
<td>2.800</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\)All runs were repeated 5 times with different seeds for the random number generator; the solutions with the lowest evaluation values are shown. Values from Ref. [11] and Ref. [12] are listed in the respective columns. Molecular constants in Ref. [11] are averages from multiple spectra and were determined using very accurate ground rotational constants from microwave experiments. Values given here are based on a spectral analysis of the same spectrum, where the ground rotational constants were also determined. The parameters that describe the relative intensity of a transition (T\(_1\), T\(_2\), W) have different values from those reported in Ref. [11]. (Ref. [12] used a one-temperature model, so their findings cannot be compared with our results). The difference is due to the fact that different sets of parameters result in equal spectral intensities. Rotational constants, \(\nu\) and \(\Delta \nu\) are in MHz, T\(_1\) and T\(_2\) in K.

\(^b\)The absolute frequency of the origin is given as the deviation from the reported value from Ref. [11] and Ref. [12].

fact that spectral information gets lost if the distance between two successive data points becomes too large.

6 Conclusions

The automated interpretation of high resolution spectra becomes of great importance if the interpretation by other methods is not feasible, is too time-consuming or just tedious. In cases where intensities as well as frequencies are dependent on the parameters that are optimized, it is crucial that both are taken into account when devising an appropriate difference function. In our example, the success of the GA method crucially depends on the newly developed evaluation function. Other, more standard, evaluation functions lead to no results.

The GA method is quite robust. It is insensitive to large line widths in the spectrum, and only at very high noise levels do the results deteriorate. Even
then, the most important parameters were unaffected. It is shown that the GA is able to use all information present in the spectrum and therefore its performance increases with the number of data points. The method of matching experimental data with simulated model data taking into account peak opens up vast possibilities in other fields, such as NMR-spectroscopy, where peak shifts determine spectral characteristics.

References

Real-Coded Adaptive Range Genetic Algorithm Applied to Transonic Wing Optimization

Akira Oyama¹, Shigeru Obayashi² and Takashi Nakamura³

¹Tohoku University, Currently, NASA Glenn Research Center, Cleveland, OH, USA.
²Tohoku University, Department of Aeronautics and Space Engineering, Sendai, 980-8579, Japan.
obayashi@ieee.org
³National Aerospace Laboratory, Chofu, Tokyo, 182-8522, Japan.
nakamura@nal.go.jp

Abstract. Real-coded Adaptive Range Genetic Algorithms (ARGAs) have been applied to a practical three-dimensional shape optimization for aerodynamic design of an aircraft wing. The real-coded ARGAs possess both advantages of the binary-coded ARGAs and the floating-point representation to overcome the problems of having a large search space that requires continuous sampling. The results confirm that the real-coded ARGAs consistently find better solutions than the conventional real-coded Genetic Algorithms do.

1 Introduction

Most of commercial aircrafts today, such as B747, B777, and A340 cruise at transonic speeds, that is, just below the speed of sound. During the long duration of cruise, engine thrust is applied to maintain aircraft speed against aerodynamic drag. Since a large part of their maximum takeoff weights is occupied by the fuel weight, the objective of an aerodynamic design optimization of a transonic wing is, in principle, minimization of drag.

Unfortunately, drag minimization has many tradeoffs. There is a tradeoff between drag and lift because one of the drag component called induced drag increases in proportion to the square of the lift. A wing that achieves no induced drag would have no lift. Another tradeoff lies between aerodynamic drag and wing structure weight. An increase in the wing thickness allows the same bending moment to be carried with reduced skin thickness with an accompanying reduction in weight. On the other hand, it will lead to an increase in another component of the drag called wave drag. Therefore, the aerodynamic design of a transonic wing is a challenging problem.

Furthermore, optimization of a transonic wing design is difficult due to the followings. First, aerodynamic performance of a wing is very sensitive to its shape. Very precise definition of the shape is needed and thus its definition usually requires more than 100 design variables. Second, function evaluations are very expensive. An aerodynamic evaluation using a high fidelity model such as the Navier-Stokes equations usually requires 60-90 minutes of CPU time on a vector computer.

Among optimization algorithms, Gradient-based Methods (GMS) are well-known algorithms, which probe the optimum by calculating local gradient information.
Although GMs are generally superior to other optimization algorithms in efficiency, the optimum obtained from these methods may not be a global one, especially in the aerodynamic optimization problem.

On the other hand, Genetic Algorithms (GAs) are known to be robust methods modeled on the mechanism of the natural evolution. GAs have capability of finding a global optimum because they don't use any derivative information and they search from multiple design points. Therefore, GAs are a promising approach to aerodynamic optimizations.

Finding a global optimum in the continuous domain for the aerodynamic design is challenging even for GAs. In traditional GAs, binary representation has been used for chromosomes, which evenly discretizes a real design space. Since binary substrings representing each parameter with a desired precision are concatenated to form a chromosome for GAs, the resulting chromosome encoding a large number of design variables for real-world problems would result in a string length too long. In addition, there is discrepancy between the binary representation space and the actual problem space. For example, two points close to each other in the real space might be far away in the binary-represented space. It is still an open question to construct an efficient crossover operator that suits to such a modified problem space.

A simple solution to these problems is the use of floating-point representation of parameters as a chromosome [1]. In these real-coded GAs, a chromosome is coded as a finite-length string of the real numbers corresponding to the design variables. The floating-point representation is robust, accurate, and efficient because it is conceptually closest to the real design space, and moreover, the string length reduces to the number of design variables. It has been reported that the real-coded GAs outperformed binary-coded GAs in many design problems [2]. However, even the real-coded GAs would lead to premature convergence when applied to aerodynamic shape designs with a large number of design variables.

The objective of the present work is to develop robust and efficient GAs applicable to aerodynamic shape designs. To achieve this goal, the idea of dynamic coding, in particular Adaptive Range GAs [3,4], is incorporated with the use of the floating-point representation. The resulting approach is then applied to a practical wing design problem as well as a simple test case to examine its performance.

To perform the practical wing design, the computation was processed in parallel using Numerical Wind Tunnel (NWT) at National Aerospace Laboratory, Japan. NWT has 166 vector processing elements at peak performance of 280 GFLOPS. The actual computation took 108 hours with 64 PE’s.

2 Adaptive Range Genetic Algorithms

To treat a large search space with GAs more efficiently, sophisticated approaches have been proposed, referred to as dynamic coding, which dynamically alters the coarseness of the search space. In [5], Krishnakumar et al. presented Stochastic Genetic Algorithms (Stochastic GAs) to solve problems with a large number of real design parameters efficiently. Stochastic GAs have been successfully applied to Flight Propulsion Controller designs [5] and air combat tactics optimization [6].

Adaptive Range Genetic Algorithms (ARGAs) proposed by Arakawa and Hagiwara [3] are a quite new approach, also using dynamic coding for binary-coded
GAs to treat continuous design space. The essence of their idea is to adapt the population toward promising regions during the optimization process, which enables efficient and robust search in good precision while keeping the string length small. Moreover, ARGAs eliminate a need of prior definition of search boundaries since ARGAs distribute solution candidates according to the normal distributions of the design variables in the present population. In [4], ARGAs have been applied to pressure vessel designs and outperformed other optimization algorithms.

Since the ideas of the Stochastic GAs and the use of the floating point representation are incompatible, ARGAs for floating point representation are developed. The real-coded ARGAs are expected to possess both advantages of the binary-coded ARGAs and the floating point representation to overcome the problems of having a large search space that requires continuous sampling.

### 2.1 ARGAs for Binary Representation

When conventional binary-coded GAs are applied to real-number optimization problems, discrete values of real design variables \( p_i \) are given by evenly discretizing prior-defined search regions for each design variable \( [p_{i,\text{min}}, p_{i,\text{max}}] \) according to the length of the binary substring \( b_{ij} \) as

\[
p_i = (p_{i,\text{max}} - p_{i,\text{min}}) \frac{c_i}{2^{d_i - 1}} + p_{i,\text{min}}
\]

where \( d_i \) represents string length and

\[
c_i = \sum_{j=1}^{d_i} (b_{ij} \cdot 2^{j-1}).
\]

In binary-coded ARGAs, decoding rules for the offspring are given by the following normal distributions,

\[
N'\left(\mu_i, \sigma_i^2\right)(p_i) = \sqrt{2\pi} \sigma_i \cdot N(\mu_i, \sigma_i^2)(p_i) = \exp\left(-\frac{(p_i - \mu_i)^2}{2\sigma_i^2}\right)
\]

where the average \( \mu_i \) and the standard deviation \( \sigma_i \) of each design variable are determined by the population statistics. Those values are recomputed in every generation. Then, mapping from a binary string into a real number is given so that the region between \( N'_{\text{ub}} \) and \( N'_{\text{lb}} \) in Fig. 1 is divided into equal size regions according to the binary bit size as

\[
p_i = \begin{cases} 
\mu_i - \sqrt{2\sigma_i^2 \cdot \ln(N'_{\text{lb}} - (N'_{\text{ub}} - N'_{\text{lb}}) \frac{c_i}{2^{d_i - 1}} - 1)} & \text{for } c_i \leq 2^{d_i - 1} - 1 \\
\mu_i + \sqrt{2\sigma_i^2 \cdot \ln(N'_{\text{ub}} - (N'_{\text{ub}} - N'_{\text{lb}}) \frac{c_i - 2^{d_i - 1}}{2^{d_i - 1} - 1}} & \text{for } c_i \geq 2^{d_i - 1}
\end{cases}
\]

where \( N'_{\text{ub}} \) and \( N'_{\text{lb}} \) are additional system parameters defined in \( [0,1] \). In the ARGAs, genes of design candidates represent relative locations in the updated range of the design space. Therefore, the offspring are supposed to represent likely a range of an optimal value of design variables.
Although the original ARGAs have been successfully applied to real parameter optimizations, there is still room for improvements. The first one is how to select the system parameters $A^\prime$ and $A^\prime$ on which robustness and efficiency of ARGAs largely depend. The second one is the use of constant intervals even near the center of the normal distributions. The last one is that since genes represent relative locations, the offsprings become constantly away from the centers of the normal distributions when the distributions are updated. Therefore, the actual population statistics does not coincide with the updated population statistics.

2.2 ARGAs for Floating-Point Representation

In real-coded GAs, real values of design variable are directly encoded as a real string $r_i$, where $p_{i,\min} \leq r_i \leq p_{i,max}$. Otherwise, sometimes normalized values of the design variables are used as

$$p_i = (p_{i,\max} - p_{i,\min}) \cdot r_i + p_{i,\min} \tag{4}$$

where $0 \leq r_i \leq 1$.

To employ floating-point representation for ARGAs, the real values of design variables $p_i$ are rewritten here by the real numbers $r_i$ defined in $(0,1)$ so that integral of the probability distribution of the normal distribution from $-\infty$ to $p_{i,n}$ is equal to $r_i$ as

$$p_i = \sigma_i \cdot p_{i,n} + \mu_i \tag{5}$$

$$r_i = \int_{-\infty}^{p_{i,n}} N(0,1)(z)dz \tag{6}$$

where the average $\mu_i$ and the standard deviation $\sigma_i$ of each design variable are calculated by sampling the top half of the previous population so that the present population distributes in the hopeful search regions. Schematic view of this coding is illustrated in Fig. 2. It should be noted that the real-coded ARGAs resolve drawbacks of the original ARGAs; no need for selecting $N'_{ub}$ and $N'_{lb}$ as well as arbitrary resolution near the average.

Updating $\mu$ and $\sigma$ every generation, however, results in inconsistency between the actual and updated population statistics in the next generation because the selection operator picks up the genes that correspond to the promising region according to the old population statistics. To prevent this inconsistency, the present ARGAs update $\mu$ and $\sigma$ every $M$ ($M>1$) generations and then the population is reinitialized. Flowchart of the present ARGA is shown in Fig. 3. To improve robustness of the present
ARGAs further, relaxation factors, \( \omega_\mu \) and \( \omega_\sigma \) are introduced to update the average and standard deviation as

\[
\mu_{\text{new}} = \mu_{\text{present}} + \omega_\mu (\mu_{\text{sampling}} - \mu_{\text{present}}) \tag{7}
\]

\[
\sigma_{\text{new}} = \sigma_{\text{present}} + \omega_\sigma (\sigma_{\text{sampling}} - \sigma_{\text{present}}) \tag{8}
\]

where \( \mu_{\text{sampling}} \) and \( \sigma_{\text{sampling}} \) are determined by sampling the top half of the population. Here, \( \omega_\mu \), \( \omega_\sigma \), and \( M \) are set to 1, 0.5 and 4, respectively. They are determined by parametric studies using some simple test functions.

In this study, design variables are encoded in a finite-length string of real numbers. Fitness of a design candidate is determined by its rank among the population based on its objective function value and then selection is performed by the stochastic universal sampling [7] coupled with the elitist strategy. Ranking selection is adopted since it maintains sufficient selection pressure throughout the optimization. One-point crossover is always applied to real-number strings of the selected design candidates. Structured coding [8] is incorporated for the wing design. Mutation takes place at a probability of 0.1, and then a uniform random disturbance is added to the corresponding gene in the amount up to 0.1.

### 2.3 Test Problem Using a Multi-Modal Function

To demonstrate how the real-coded ARGA works, it was applied to minimization of a high dimensional multi-modal function:

\[
F1 = \sum_{i=1}^{20} (x_i^2 + 5(1 - \cos(x_i \cdot \pi))) \tag{9}
\]

where \( x_i \in [-3, 3] \). This function has a global minimum at \( x_i = 0 \) and two local optima near \( x_i = \pm 2 \). In the real-coded ARGA, \( x_i \) correspond to \( p_i \) in eq.(5). 150 generations were allowed with a population size of 300. Five trials were run for each GA changing seeds for random numbers to give different initial populations. Figure 4
compares the performances of the conventional GA and the ARGA. Figure 5 plots all $x_i$'s from the temporary solutions, which helps to understand why the ARGA works better than the conventional GA. This figure shows that the ARGA maintains gene diversity longer than the conventional GA in the initial phase and then adapts to their search space to the local region near the optimal. While the initial gene diversity contributes to the ARGA's robustness, the adaptive feature of the ARGA improves their local search capability. The ARGA also showed its advantages over a real-coded GA on dynamic control problem and aerodynamic airfoil shape optimization [9].

3 Aerodynamic Design of a Transonic Wing

A wide range of approximations can represent the flow physics. Among them, the Navier-Stokes equations provide the state-of-the-art of aerodynamic performance evaluation for engineering purposes. Although the three-dimensional Navier-Stokes calculation requires large computer resources to estimate wing performances within a reasonable time, it is necessary because a flow around a wing involves significant viscous effects, such as potential boundary-layer separations and shock wave/boundary layer interactions in the transonic regime. Here, a three-dimensional Reynolds-averaged Navier-Stokes solver [10] is used to guarantee an accurate model of the flow field and to demonstrate the feasibility of the present algorithm.

The objective of the present wing design problem is maximization of lift-to-drag ratio $L/D$ at the transonic cruise design point, maintaining the minimum wing thickness required for structural integrity against the bending moment due to the lift distribution. The cruising Mach number is set to 0.8. The Reynolds number based on the chord length at the wing root is assumed to $10^6$.

In the present optimization, a planform shape of generic transport was selected as the test configuration (Fig. 6). Wing profiles of design candidates are generated by the PARSEC airfoils as briefly described in the next section. The PARSEC parameters and the sectional angle of attack (in other words, root incident angle and twist angle) are given at seven spanwise sections, of which spanwise locations are also treated as design variables except for the wing root and tip locations. The PARSEC parameters are rearranged from root to tip according to the airfoil thickness so that the resulting wings always have maximum thickness at the wing root. The twist angle parameter is also rearranged into numerical order from tip to root. The wing surface is then interpolated in spanwise direction by using the second-order Spline interpolation.

In total, 87 parameters determine a wing geometry. Parameter ranges of the design space are shown in Table 1. It should be noted that in ARGAs, user-defined design space is used just to seed the initial population. ARGA can promote the search space outside of the initially defined design space.

To estimate the required thickness distribution to stand the bending moment due to the lift distribution, the wing is modeled by a thin walled box-beam as shown in Fig. 6. The constraint for wing thickness $t$, is specified by using the minimum thickness $t_{\text{min}}$ calculated from the wing box sustaining the aerodynamic bending moment $M$ as,

$$t_1 > \frac{M}{\sigma_{\text{ultimate}} \cdot c \cdot t_2} = t_{\text{min}}$$

(10)
where following assumptions are made: the thickness of the skin panels are 2.5[cm] and its ultimate normal stress $\sigma_{\text{ultimate}}$ is 39[ksi]. The length of the chord at wing root $c$ and maximum wingspan $b/2$ are 10[m] and 18.8[m], respectively (for the derivation of Eq. (10), see [11] for example).

![Planform](image)

**Planform**

\[ c_{\text{root}} \]

\[ b/2 \]

\[ t_1 \]

\[ t_2 \]

**Wing box**

![PARSEC parameters](image)

**PARSEC parameters**

Fig. 6 Wing geometry definition. Planform shape is frozen during the optimization. Wing box is used to estimate its structural strength. PARSEC parameters are the design variables for airfoil shapes defined at seven spanwise sections.

Table 1 Parameter ranges of the design space. PARSEC is determined by leading-edge radius ($r_{LE}$), upper and lower crest locations including curvatures ($X_{UP}$, $Z_{UP}$, $Z_{XXUP}$, $X_{LO}$, $Z_{LO}$, $Z_{XXLO}$), trailing-edge ordinate ($Z_{TE}$) and thickness ($\Delta Z_{TE}$) and direction and wedge angles ($\alpha_{TE}$, $\beta_{TE}$).

<table>
<thead>
<tr>
<th>parameters</th>
<th>$r_{LE}$</th>
<th>$Z_{TR}$</th>
<th>$\alpha_{TR}$</th>
<th>$\beta_{TR}$</th>
<th>$X_{UP}$</th>
<th>$Z_{UP}$</th>
<th>$Z_{XXUP}$</th>
<th>$X_{LO}$</th>
<th>$Z_{LO}$</th>
<th>$Z_{XXLO}$</th>
<th>$\Delta Z_{TE}$</th>
<th>twist angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper bound</td>
<td>0.030</td>
<td>0.01</td>
<td>-3.0</td>
<td>8.0</td>
<td>0.7</td>
<td>0.18</td>
<td>0.0</td>
<td>0.0</td>
<td>0.02</td>
<td>0.9</td>
<td>0.18</td>
<td>7 deg</td>
</tr>
<tr>
<td>Lower bound</td>
<td>0.002</td>
<td>-0.01</td>
<td>-13.0</td>
<td>4.0</td>
<td>0.3</td>
<td>0.08</td>
<td>-0.3</td>
<td>0.2</td>
<td>-0.04</td>
<td>0.3</td>
<td>0.3</td>
<td>-1 deg</td>
</tr>
</tbody>
</table>

3.1 **PARSEC Airfoils**

An airfoil family “PARSEC” has been recently proposed to parameterize an airfoil shape [12]. A remarkable point is that this technique has been developed aiming to control important aerodynamic features effectively by selecting the design parameters based on the knowledge of transonic flows around an airfoil.

Similar to 4-digit NACA series airfoils, the PARSEC parameterizes upper and lower airfoil surfaces using polynomials in coordinates $X$, $Z$ as,

\[
Z = \sum_{n=1}^{6} a_n \cdot X^{n-1/2}
\]  

(11)

where $a_n$ are real coefficients. Instead of taking these coefficients as design parameters, the PARSEC airfoils are defined by basic geometric parameters:
leading-edge radius \(r_{LE}\), upper and lower crest locations including curvatures \(X_{up}, Z_{up}, Z_{XXP}, X_{LOP}, Z_{LOP}, Z_{XXLO}\), trailing-edge ordinate \(Z_{TE}\), thickness \(\Delta Z_{TE}\) and direction and wedge angles \(\alpha_{TE}, \beta_{TE}\) as shown in Fig. 6. These parameters can be expressed by the original coefficients \(a_i\) by solving simple simultaneous equations. Eleven design parameters are required for the PARSEC airfoils to define an airfoil shape in total. In the present case, the trailing-edge thickness is frozen to 0. Therefore, ten design variables are used to give each spanwise section of the wing.

### 3.2 Optimization Using Real-Coded ARGA

Because the objective function distribution of the present optimization is likely to be more complex than the above test function minimization, the relaxation factor \(\omega_0\) is now set to 0.3. The structured coding coupled with one-point crossover proposed in [13] is also incorporated. The present ARGA adopts the elitist strategy where the best and the second best individuals in each generation are transferred into the next generation without any crossover or mutation. The parental selection consists of the stochastic universal sampling and the ranking method using Michalewicz's nonlinear function. Mutation takes place at a probability of 10% and then adds a random disturbance to the corresponding gene in the amount up to \(\pm 10\%\) of each parameter range in Table 1. The population size is kept at 64 and the maximum number of generations is set to 65 (based on the CPU time allowed). The initial population is generated randomly over the entire design space.

The main concern related to the use of GAs coupled with a three-dimensional Navier-Stokes solver for aerodynamic designs is the computational cost required. In the present case, each CFD evaluation takes about 100 min. of CPU time even on a vector computer. Because the present optimization evaluates \(64 \times 65 = 4160\) design candidates, sequential evolutions would take almost 7000 h (more than nine months!). Fortunately, parallel vector computers are now available at several institutions and universities. In addition, GAs are intrinsically parallel algorithms and can be easily parallelized. One of such computers is *Numerical Wind Tunnel (NWT)* located at National Aerospace Laboratory in Japan. NWT is a MIMD parallel computer with 166 vector-processing elements (PEs) and its total peak performance and the total main memory capacity are about 280 GFLOPS and 45GB, respectively. For more detail, see [14]. In the present optimization, evaluation process at each generation was parallelized using the master-slave concept. This made the corresponding turnaround time almost 1/64 because the CPU time used for GA operators are negligible.

To handle the structural constraint with the single-objective GA, the constrained optimization problem was transformed into an unconstrained problem as

\[
\text{fitness function} = \begin{cases} 
100 + \frac{L}{D} & \text{if } t \geq t_{min} \\
(100 + \frac{L}{D}) \cdot \exp(t - t_{min}) & \text{otherwise}
\end{cases} \tag{12}
\]

where \(t\) and \(t_{min}\) are thickness and minimum thickness at the span station of the maximum local stress. The exponential term penalizes the infeasible solutions by reducing the fitness function value. Because some design candidates can have negative \(L/D\), the summation of 100 and \(L/D\) is used.
3.3 Results

The optimization history is shown in Fig. 7 in terms of $L/D$. During the initial phase of the optimization, some members had a strong shock wave or failed to satisfy the structural constraint. However, they were weeded out from the population because of the resultant penalties to the fitness. The final design has $L/D$ of 18.91 ($C_l = 0.26213$ and $C_d = 0.01386$) satisfying the given structural constraint. Turnaround time of this optimization was about 108 h on NWT.

To examine whether the present optimal design is close to a global optimum, we have checked it against analytically and empirically established design guidelines. In aerodynamics, spanwise lift distribution should be elliptic to minimize the induced drag. However, the structural constraint leads to a tradeoff between induced drag and wave drag. This enforces the spanwise lift distribution to be linear rather than elliptic. The present solution has a linear distribution. To produce this distribution, a wing is usually twisted in about five degrees. The present wing is twisted in six degrees.

Figure 8 shows the designed airfoil sections and the corresponding pressure distributions at the 0, 33, and 66% spanwise locations. In the pressure distributions, neither any strong shock wave nor any flow separation is found. This ensures that the present wing has very little wave drag and pressure drag. At 33 and 66% spanwise locations, the rooftop, front-loading and rear loading patterns are observed, which are typical for the supercritical airfoils [15] used for advanced transport today. The corresponding airfoil shapes are indeed similar to supercritical airfoils. Overall, these detailed observations of the design confirm that the present design is very close to a global optimum expected by the present knowledge in aerodynamics.

4 Summary

To develop GAs applicable to practical aerodynamic shape designs, the real-coded ARGAs have been developed by incorporating the idea of the binary-coded ARGAs with the use of the floating-point representation. The real-coded ARGA has been applied to a practical aerodynamic design optimization of a transonic wing shape for generic transport as well as a simple test case. The test case result confirms the
present GA outperforms the conventional GA.

Aerodynamic optimization was performed with 87 real-number design variables by using the Navier-Stokes code. The realistic structural constraint was imposed. The resulting wing appears very similar to advanced wing designs based on supercritical airfoils. The straight span load distribution of the resulting design represents a compromise between minimalizations of induced drag and wave drag. The designed wing also has a fully attached flow and the allowable minimum thickness so that pressure drag and wave drag are minimized under the present structural constraint. These results confirm the feasibility of the present approach for future applications.

5 References

Stream Cyphers  
with One- and Two-Dimensional Cellular Automata  

Marco Tomassini and Mathieu Perrenoud  

Computer Science Institute, University of Lausanne,  
1015 Lausanne, Switzerland  

Abstract. In the age of universal electronic connectivity and the ensuing possibility of message eavesdropping and electronic fraud, cryptography becomes a basic requirement to secure data storage and transmission. In this work we describe a single key cryptographic system based on one- and two-dimensional non-uniform cellular automata that produce high-quality pseudo-random bit sequences. The robustness of the scheme against cryptanalytic attacks is discussed and it is shown that direct cryptanalysis requires an exponentially growing amount of computational resources. The advantage of implementing the proposed scheme in hardware for high-speed operation is also discussed.

1 Introduction  

Public and private organizations have become increasingly dependent on cryptographic techniques in the present times characterized by the growth of digital information storage and transmission through global networks. Thus, there is a need for privacy and authentication in many areas, especially in electronic commerce transactions and for classified material. Cryptography has a long history and many different cryptographic techniques have been suggested: an excellent review is given in [8]. Here we will describe the application of some types of cellular automata in this domain. Cellular automata (CA) have previously been used as encrypting devices by Wolfram [13] and by Nandi et al. [7]. Gutowitz [3] and Guam [2] used CAs for public-key cryptography but their work will not be discussed here since we apply CAs to symmetric systems, where the encryption key and the decryption key are the same or can be calculated from each other. Our encryption scheme is based on the generation of pseudorandom bit sequences by cellular automata as in [13, 7]. In the next section, we summarise work done on CAs for random number generation by our group and by others. The following section presents the implementation of the proposed cryptographic system and compares it with previous approaches. Finally, we discuss the vulnerability of the scheme to possible cryptanalytic attacks, showing that it is robust and of practical use for many common types of cryptographic applications.
2 Cellular Automata for Pseudorandom Number Generation

Pseudorandom number sequences are needed in many important applications, such as Monte Carlo techniques, Brownian dynamics, and stochastic optimization methods. Cellular automata offer a number of advantages over other methods as random number generators (RNG). Among the beneficial features of CAs for VLSI implementation one can cite simplicity, regularity, and locality of inter­
connections, which make them suitable for on-board applications such as built-in self-test circuits, associative memories, and special purpose massively parallel fine-grained computers. CAs are dynamical systems in which space and time are discrete. A cellular automaton consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. Here we will only consider Boolean automata for which the cellular state s ∈ {0, 1}. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. The cellular array (grid) is d-dimensional, where d = 1, 2, 3 is used in practice; in this paper we shall concentrate on d = 1 and d = 2, i.e., one and two-dimensional grids. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table (also known as the transition function), with an entry for every possible neighborhood configuration of states. The cellular neighborhood of a cell consists of itself and the surrounding (adjacent) cells. For one-dimensional CAs, a cell is connected to r local neighbors (cells) on either side where r is referred to as the radius (thus, each cell has 2r + 1 neighbors). For two-dimensional CAs, two types of cellular neighborhoods are usually considered: 5 cells, consisting of the cell along with its four immediate nondiagonal neighbors (also known as the von Neumann neighborhood), and 9 cells, consisting of the cell along with its eight surrounding neighbors (also known as the Moore neighborhood). When considering a finite-size grid, spatially periodic boundary conditions are frequently applied, resulting in a circular grid for the one-dimensional case, and a toroidal one for the two-dimensional case. Non-uniform (also known as inhomogenous) cellular automata are the same as uniform ones with the exception that cellular rules need not be identical for all cells. Asynchronous updating of the cell values is also possible but it will not be used here.

S. Wolfram [14] first proposed CAs as pseudo-random number generators (PRNG). In particular, he extensively studied the bit sequences generated by the one-dimensional, r = 1, rule 30 where the rule number represents in decimal format the binary number encoding the rule table. In Boolean form rule 30 can be written as:

\[ s_i(t+1) = s_{i-1}(t) \text{ XOR } (s_i(t) \text{ OR } s_{i+1}(t)), \]

where \( s_i(t) \) is the state of cell \( i \) at time \( t \). The formula gives the state of cell \( i \) at time step \( t + 1 \) as a Boolean function of the states of the neighboring cells at time \( t \). Pseudorandom bit sequences are obtained by sampling the values that a
Fig. 1. A good quality non-uniform one-dimensional random number generator consisting of a random combination of rules 90, 105, 150 and 165. The horizontal axis depicts the configuration (black stands for 0 and white stands for 1) at a certain time \( t \) and the vertical axis depicts successive time steps.

particular cell (usually the central one) attains as a function of time. To further decorrelate bit sequences time spacing and site spacing are used. Time spacing means that not all the bits generated are considered as part of the random sequence while in site spacing, one considers only a fraction of the sites in a row. For applications that need very good quality random numbers, this sacrifice is acceptable, if one takes into account that, in fact, many parallel streams of random bits are being generated simultaneously by a CA.

Hortensius et al. proposed a non-uniform CA randomizer consisting of two rules, 90 and 150, arranged in a specific order in the grid [4, 5]. In Boolean form rule 90 can be written as:

\[ s_i(t + 1) = s_{i-1}(t) \ XOR \ s_{i+1}(t), \]

and rule 150 can be written as:

\[ s_i(t + 1) = s_{i-1}(t) \ XOR \ s_i(t) \ XOR \ s_{i+1}(t). \]

The performance of this non-uniform CA in terms of random number generation was found to be superior to that of rule 30 and to the usual linear feedback shift register approach.
Sipper and Tomassini [10] showed that good non-uniform CA randomizers can be evolved by a genetic algorithm, rather than being designed. This work has been recently pursued and excellent generators were obtained by randomly mixing the four rules that appeared most frequently within successful evolved CAs. The rule numbers are 90, 105, 150 and 165 and any random mixture of these rules produces a good RNG: Figure 1 depicts one of the best one-dimensional non-uniform CA that was obtained. For details on the artificial evolution of these RNGs and their testing and evaluation see [12].

More recently, we have obtained by artificial evolution and design 2-D, non-uniform CAs for random number generation with even better statistical quality [11], as demonstrated by the application of a complete battery of stringent tests for pseudo-randomness. Figure 2 demonstrates the operation of one such $15 \times 15$ CA.

Fig. 2. The operation of a two-dimensional CA randomizer. In the above image, the 15 lines of the $15 \times 15$ grid have been juxtaposed, thus forming one 225-cell line. Each line in the image depicts the configuration of (225) states at a given time, with time increasing down the page. The random nature of the CA bit sequences can be visually appreciated (a fact confirmed by the statistical randomness tests).

The following section presents the application of these pseudorandom bit sequences to the generation of stream cyphers for cryptography.
3 A CA-Based Key Stream Generator

Let $P$ be a plaintext message and $E$ an enciphering algorithm. The fundamental transformation to obtain the ciphertext $C$ is thus:

$$C = E_k(P),$$

where $k$ is the key of the transformation which distinguishes a particular encryption in a family of transformations using the same enciphering algorithm. To recover the original message, a deciphering function $D_k$, using the same key, is defined as the inverse of $E$:

$$P = D_k(C) = D_k(E_k(P)).$$

Enciphering algorithms that operate on the plaintext a single bit at a time are called stream algorithms or stream ciphers. A stream cipher breaks the message $P$ into a stream of successive bits or bytes $p_1, p_2, \ldots, p_q$ and enciphers each $p_i$ with a stream of bits (or bytes) $k_1, k_2, \ldots, k_q$ generated by a key stream generator such that:

$$E_k(P) = E_{k_1}(p_1)E_{k_2}(p_2)\ldots$$

A common enciphering operation, and the one used here, is the exclusive-or operation $XOR$:

$$c_i = k_i \ XOR \ p_i,$$

where $c_i$ is the $i$-th bit of the cyphertext. Applying the same operation on the cyphertext allows the recovery of the original text:

$$p_i = c_i \ XOR \ k_i = (k_i \ XOR \ p_i) \ XOR \ k_i.$$ 

If the key stream is truly unpredictable, then we have the so-called “one-time pad”, a non-repeating key variation of the Vernan cipher [8] system which is perfectly safe (assuming that the keys are not stolen or eavesdropped). However, the one-time pad is impractical because the sender and the receiver must be in possession, and protect, the random key. Moreover, the total amount of data that can be encrypted is limited by the amount of key available.

Thus, the security of a practical stream cypher scheme rests on the predictability of the bits in the sequence. Good statistical pseudorandomness of the key stream is not sufficient in cryptographic applications: a perfectly good RNG may be completely unsuitable if the next random bit can be predicted from the previous sequence. From that point of view, cellular automata are more suitable than classical RNGs such as linear congruential generators, which are very easy to crack, given the algorithm and a small portion of the sequence (see [8] for example).
3.1 Earlier Proposals

Given a CA of size N, a configuration of the grid at time t is defined as:

\[ C(t) = (s_0(t), s_1(t), \ldots, s_{N-1}(t)) \],

where \( s_i(t) \in \{0, 1\} \) is the state of cell i at time t. For a CA-based key stream generator, the initial configuration of states \( C(0) \) is the key or a part of the key. Thus, cryptanalysis of the system amounts to the reconstruction of \( C(0) \). In Wolfram’s proposal [13], the bits \( k_1, k_2, \ldots \) to be used for enciphering the message stream \( p_1, p_2, \ldots \) are extracted from the sequence \( (s_i(t), s_i(t + 1), s_i(t + 2), \ldots) \) attained through time at a site i (say, the central cell) of a rule 30 CA. It would appear that reconstructing the key from a fragment of the sequence is a difficult problem. However, Meier and Staffelbach [6] showed that there is a known plaintext attack against this generator which is successful in reasonable time and with limited computing power for key sizes N up to 500. The attack is based on reconstructing a right (or left) sequence adjacent to the sequence at site i by guessing adjacent site states with high a probability of success, due to the particular nature of rule 30. Once the adjacent sequence is known, it is an easy matter to successively compute the missing site values backwards to the complete \( C(0) \) key.

In another study, Nandi et al. [7] proposed using a non-uniform CA generator based upon rules 90 and 150 for producing a key stream. It has been shown that such a generator produces good-quality pseudorandom streams (see section 2 and [4]), better than either rule 30 or linear feedback shift registers. Nandi et al. presented both block and stream cipher algorithms, here we will only briefly discuss the stream cipher scheme. They proposed two different systems: a programmable CA with ROM (PCA) and a two-stage programmable CA. Both schemes are easy to implement in hardware. In the PCA, an N-cell CA register is loaded with one of a number of (90, 150) rule configurations stored in a ROM memory and run for a clock cycle. A group of four consecutive output bits is chosen out of the N output bits and xored with a portion of plaintext of the same length to give the corresponding ciphertext. The CA is then loaded with another rule configuration from the ROM and the cycle continues until all the text has been encrypted. The two-stage system works according to the same principles but instead of configuring the CA from a fixed set of configurations stored in ROM, a second CA register, which is itself constituted by a fixed (90, 150) rule vector, randomly produces the desired (90, 150) current rule configuration for the first CA register at each clock cycle.

In view of the static and periodically repeating rule configurations stored in ROM, the PCA appears to be vulnerable to attacks. In fact, successive rule configurations are always used in the same sequential order. Moreover, the suggested values for the CA register length and for the number of rule configurations stored in ROM are small, thus facilitating a cryptoanalytic attack. The two-stage system should be more secure since, ideally, there is no correlation between successive rule configurations. Both systems use groups of four consecutive bits, a fact that makes rule induction by divide-and-conquer and completion easier.
3.2 Our Scheme

In our scheme, we use one- and two-dimensional non-uniform CAs as key stream generators. In the 1-D case each cell rule is randomly sampled with uniform probability from rules 90, 105, 150 and 165. We have shown in ref. [11] (see also section 2) that these automata have excellent pseudorandom number generation properties. The key $K$ is formed from the current vector of CA rules $R_i$ and the initial random state of the CA $C(0)$: $K = (R_i, C(0))$. For an N-cell CA each of the $N$ rules can be chosen independently among the set of four and each bit in the initial configuration is also independent; thus, the apparent length of the key is: $4^N \times 2^N = 2^{3N}$. To give an order of magnitude, $N$ can be well above 100 if the key stream generator is implemented in software and in the order of 100 for VLSI implementation, although this is only a cost/benefit tradeoff and suitable values of $N$ can be used as a function of the degree of security one wants to reach. Obviously the hardware implementation will be several orders of magnitude faster. The rule vector and the initial state can be established once per session (session key) or they can be changed at any time. As in Wolfram's proposal [13], bits are sampled from the central column of a cyclically connected CA. But in our case divide-and-conquer attacks of the kind proposed in [6] are much more difficult as shown by the following argument.

Suppose that we have a plaintext and the corresponding cyphertext of a given arbitrary but finite length. The goal of the cryptanalysis is to find the key $K = (R_i, C(0))$ starting from the bit stream generated with $K$ during a number $n$ of time steps. The bit stream used for encoding can obviously be deduced unambiguously from the plaintext/cyphertext pair. The search space is huge as there exist $2^{3N}$ possible key configurations. However, among those only a small fraction generate the given bit stream as the following counting argument will show. We will start by trying to reconstruct the automaton backwards in a triangle shape starting from a given bit in the central sequence until we have reconstructed a full line. The triangle shape is due to the fact that each cell state depends on the state of its two neighbours at the previous time step. For each time step we must make hypothesis about the rules and the states of the cells and we will count the number of guesses we must do in order to complete the line. Consider Figure 3 (a) which represents the states of the relevant cells and the rules at time steps $t$ and $t-1$ respectively. Cell states $x$ and $y$ are known since they belong to the central sequence. We must guess the central cell’s rule, let us call it $A$, and the value (0 or 1) of one of the neighbors of the cell whose value is $x$. Assume that we guess $m$. This is enough to infer the $t-1$ slice of the triangle since we only work with additive rules ($XOR$ and $XNOR$ logic only). At this stage, there are 8 different configurations of the triplet $(A, m, n)$ producing the sequence $(x, y)$. If we now go one time step further backwards to time $t-2$, we have the situation depicted in part (b) of the Figure. The central cell value $z$ is known and we need to infer the state values $o, p, q, r$. With $A, m, x, y, n$ now fixed we must guess rules $B$ and $C$ and one of $o, p, q, r$ as the other values can then be inferred. There are 32 possible configurations of $(B, C, o, p, q, r)$ that produce the bits $(z, m, x, y, n)$. Now it is easy to see that for each further time
step backwards there are 32 possible configurations of rules/values to consider in order to obtain the situation at the previous time step and the process can be iterated \((n-1)/2\) times until we get a full line. Taking the product gives us the number of combinations of rules/values that produce the given sequence of \(n/2\) bits. This number is \(u = 2^{(5n-9)/2}\). But only one of those combinations will fit a complete sequence. In fact, if we now try to extend the bit sequence further down by 1 bit, only half of the solutions will produce the right sequence. The next bit will reduce that to a quarter and so on. Thus, to find the right combination of rules and initial state values, one has to test all the \(u\) cases over \(\log(u)\) subsequent bits belonging to the central sequence. The system can be made even more secure by time spacing the bit stream which will have the effect of increasing the number of rules/values that have to be guessed to reconstruct the time/space diagram backwards.

![Schema of the first two stages in the backward completion of the automaton by guessing rule and cell values.](image)

Two-dimensional non-uniform CAs are even better PRNGs than 1-D ones [11] and they can be used in a manner analogous to the one-dimensional case for generating key streams. Seven additive rules tend to emerge consistently via artificial evolution giving rise to excellent PRNGs. Randomly mixing these rules again produces very good generators. The key length is thus \(7^N \times 2^N\), larger than the length in the one-dimensional case. Moreover, cryptanalysis of the stream by backwards completion is even more difficult, as can be easily seen by extending the previous argument to a neighborhood of five cells instead of two and seven possible rules instead of four. Thus, two-dimensional non-uniform CAs of the type described here are very good candidates as encrypting devices.

Finally, it has to be acknowledged that although direct cryptanalysis requires an exponentially growing amount of computational resources in the key length, one cannot rule out the possibility that, by exploiting some kind of information-theoretic or statistical regularities, the system be transformed into an equivalent one that is easier to analyze. With the information given here and in refs. [11, 12] our generators can be easily programmed and any length of plaintext/cyphertext
pair can be generated. As such, the proposed encryption technique is open to scrutiny.

4 Conclusions

The system we have described is based on non-uniform one-dimensional or two-dimensional cellular automata that are able to generate pseudo-random bit sequences of very good quality as measured by stringent statistical tests. The system appears to be secure, especially when 2-D CAs are used as we have shown that reconstructing the key requires an amount of computational resources which is an exponential function of the key length.

In this respect, it is also worth noting that all known practical cryptographic techniques are insecure in absolute terms and there is always a tradeoff between the time and the resources available to the cryptanalyst and the lifetime of the encryption key. With our system, the key can be changed at will and on the fly, although this in turn raises the question of the key exchange between parties. But this problem can be solved by using standard public-key cryptography for exchanging one-time session keys as frequently as needed. Of course, theoretically, even RSA public-key cryptography is not absolutely safe since the difficulty of factoring, barring new and unlikely efficient algorithms, depends on the length of the key and on the amount of computing power that a cryptanalyst is able to deploy. This power is on the raise, which implies that longer keys have to be used. In fact, public-key schemes could easily be cracked today if efficient quantum computing algorithms devised for the task could be implemented in practice [9]. Cryptographic techniques that are both provably secure and physically realizable do exist. They are based on the protocol invented in 1984 by C. Bennet and G. Brassard [1] which makes use of quantum information concepts. This approach is extremely promising but still difficult to use in practice due to the technological constraints of today’s communication and storage devices.

The practical advantage of the CAs described here is that they are very easy and cheap to implement both in hardware and software and could meet the needs of many applications in which a high degree of security is needed but provably absolute inviolability is not called for. Using VLSI implementations, very high speed encrypting and decrypting circuits can be built and incorporated in communication devices such as hand-held computers and mobile phones. In this respect, an exciting prospect would be to build them using modern reconfigurable circuits such as Field-Programmable Gate Arrays (FPGA). These circuits are becoming competitive with custom VLSI but, instead of being wired once and for all, their function can be easily and quickly reprogrammed on the fly. The advantage is that in this way programmability of the generator would be retained together with the implied security aspects, as well as speed of operation.

References


Machine Learning and Classifier Systems
Investigating Generalization in the Anticipatory Classifier System

Martin V. Butz¹, David E. Goldberg¹, and Wolfgang Stolzmann²

¹ Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign, IL, USA,
{butz, deg}@illgal.ge.uiuc.edu
² Institut für Psychologie III
Universität Würzburg, Germany
stolzmann@psychologie.uni-wuerzburg.de

Abstract. Recently, a genetic algorithm (GA) was introduced to the Anticipatory Classifier System (ACS) which surmounted the occasional problem of over-specialization of rules. This paper investigates the resulting generalization capabilities further by monitoring the performance of the ACS in the highly challenging multiplexer task in detail. Moreover, by comparing the ACS to the XCS classifier system in this task it is shown that the ACS generates accurate, maximally general rules and its population converges to those rules. Besides the observed ability of latent learning and the formation of an internal environmental representation, this ability of generalization adds a new advantage to the ACS in comparison with similar approaches.

1 Introduction

Generalization is one of the most important issues to enable problem solving in nature. Higher animals filter out which information is important to satisfy each of their needs. Instincts can be viewed as evolved generalized rules dependent on only a few stimuli. In artificial systems generalization is as important as in nature. For example an autonomous robot equipped with many different sensors must be able to generalize to solve similar tasks. Generalization is a powerful property of learning classifier systems (LCSs).

The Anticipatory Classifier System (ACS) (Stolzmann, 1998) enhances the rule structure of traditional LCSs with an effect part anticipating the consequences of an action in a certain situation. This additional effect part opens the portal to an internal representation of the perceived environment and thus to many cognitive processes such as lookahead planning (Stolzmann & Butz, 2000), mental acting and lookahead action selection (Stolzmann, Butz, Hoffmann, & Goldberg, 2000). While the great benefits of the cognitive capabilities were shown in the cited papers, the aim herein is to analyze the genetic generalization capabilities, introduced in Butz, Goldberg, and Stolzmann (2000a), further using the multiplexer test bed. Although this problem is a one step problem and thus not typical for the ACS, it serves well to analyze the generalization capabilities of the system. The test bed was already successfully applied in the ACS (Butz,
Goldberg, & Stolzmann, 2000b) but a complete convergence could not be shown. This paper will show that the ACS is able to converge to the accurate, maximally general rules. Furthermore, the performance will be compared to the XCS classifier system (Wilson, 1995), which is able to solve the multiplexer problem with an optimal generalization (Wilson, 1998).

The next section will give an overview of the ACS with the present specialization and generalization pressure and the interaction of the two pressures. After that, Section 3 introduces the multiplexer environment and analyzes the performance of the ACS. Finally, a summary is given.

2 Overview of the Anticipatory Classifier System

The ACS is based on LCSs and on the psychological learning theory of anticipatory behavioral control (Hoffmann, 1993) that is realized by the anticipatory learning process (ALP). The theory states that learning is directed by the need to correctly anticipate behavioral consequences; thus requiring the construction of an internal representation of the environment. The enhanced LCS structure combined with the ALP realizes this need in the ACS.

However, the ACS is not the first mechanism that is able to form an internal environmental representation. Sutton (1991) introduced the Dyna-Q algorithm which uses an environmental representation to speed up the reinforcement learning capabilities. Holland (1990) and Riolo (1991) developed another classifier system which was able to form an environmental representation. Drescher (1991) published a schemata approach that is based on the Piagetian development theory. The ACS differs from these approaches in its directed rule discovery due to the realization of the anticipatory behavioral control theory as well as the type of genetic generalization.

2.1 The Basics of the ACS

The ACS perceives states \( \sigma \in \{s_1, s_2, ..., s_m\}^L \) from the environment, where \( L \) is the length of the strings, \( m \) is the number of possible different values, and \( s_1, ..., s_m \) are the different possible values. It acts upon the environment with possible actions \( \alpha \) that are also coded with discrete values \( \alpha \in \{r_1, r_2, ..., r_n\} \), where \( n \) is the number of different actions and \( r_1, ..., r_n \) are the different actions. After the execution of an action it receives a payoff \( \rho \), a simple real value.

The knowledge of the ACS is represented by its population of classifiers. Each classifier has a condition part \( C \), an action part \( A \), an effect part \( E \), a mark \( M \), a time stamp \( t_s \), a numerosity \( num \), a quality measure \( q \), and a reward prediction measure \( r \). \( C \) and \( E \) have the structure \( \{s_1, ..., s_m, \#\}^L \). A '\#' symbol in an attribute of \( C \), which is called 'don't-care' symbol, matches any perceived value in this attribute. The more 'don't-care' symbols are in \( C \), the more general a classifier is. A '\#' symbol in an attribute of \( E \), which is called 'pass-through' symbol, predicts that this attribute will remain the same (i.e. the corresponding value in the perception does not change after the execution of an action). The action part \( A \) consists out of any action which is possible in the environment (i.e. \( A \in \alpha \)). The mark has the structure \( M = (m_1, ..., m_L) \) with \( m_i \subseteq \{s_1, ..., s_m\} \). \( M \) serves as a remainder that the classifier did not work correctly sometimes as well
as specifies the situations where it did not work correctly. Due to its structure it is assured that its size does not exceed \( L \times m \). num gives the classifier the property of a macro-classifier identical to XCS (Wilson, 1995). Each classifier can represent several identical classifiers in the population. This representation is used for an increase in the speed of the system and affects the performance only slightly. \( q \) measures the quality of the anticipations of the classifier and \( r \) predicts the reward.

At time \( t \), a behavioral act starts with the perception of the current situation \( \sigma(t) \). First, the ACS forms a match set out of all classifiers whose conditions match the current situation. Next, the ACS chooses an action (here randomly), executes the action, and forms an action set out of the match set. Using the perceptions of the next state \( \sigma(t+1) \) it forms a learning set out of the action set by applying the ALP. The ALP realizes the latent learning and the formation of the environmental representation. It resembles the specialization pressure in the ACS. Finally, the actual payoff \( \rho(t) \) combined with the predicted payoff in the resulting situation is distributed among the classifiers in the resulting learning set applying an update similar to Q-learning with discount factor \( \gamma \) and learning rate \( b_r \):

\[
r = (1 - b_r) \times r + b_r \times (\gamma \times \max_{cl \in M_{set}(t+1)}(q_{cl} \times r_{cl}) + \rho(t))
\]

\[\text{(1)}\]

### 2.2 The Specialization Pressure

The ACS assumes in the beginning that no action has any effect. (i.e. each possible action is represented by one classifier that only has ’#’-symbols in \( C \) and \( E \). The ALP then carefully generates more specific rules out of those most general ones. It is applied in the action set and distinguishes the following cases:

- A classifier anticipates \( \sigma(t+1) \) incorrectly. In this case the incorrect classifier is marked with \( \sigma(t) \) and its quality is decreased. Furthermore:
  - If \( E \) can be further specialized (i.e. ’#’-symbols can be changed to specific values) to correct the anticipations, then a new classifier is built which is specified in these attributes in \( C \) and \( E \).
  - If \( E \) cannot be further specialized or nothing changed in the environment at all, then no new classifier is created.

- A classifier anticipates \( \sigma(t+1) \) correctly. Now the quality is increased. Furthermore, if the classifier is marked (which means that the classifier anticipated incorrectly in earlier experienced situations) and it can be further specialized, a new classifier is formed which is more specialized in \( C \). It can be further specialized, if it is not already completely specialized and if the number of specified unchanging attributes (i.e. attributes which have non-’#’-symbols in \( C \) but ’#’-symbols in \( E \)) does not exceed the threshold \( u_{max} \).

An increase or decrease of \( q \) is realized by the Widrow-Hoff delta rule with learning rate \( b_q \) (i.e. \( q = (1 - b_q) \times q + b_q \) or \( q = (1 - b_q) \times q \)). If \( q \) decreases under \( \theta_i = 0.1 \) the classifier is deleted. If \( q \) increases over \( \theta_r = 0.9 \) the classifier is considered as reliable and thus part of the internal environmental representation. If
a generated classifier already exists, the newly generated classifier is not inserted into the population but the quality of the old one is increased.

Although the ALP generates more specialized classifiers carefully there are some causes for over-specialization. (1) The ALP specializes the changing attributes in a classifier always in \( C \) and \( E \) although the specialization in \( C \) is not always necessary. (2) The information in the mark can be misleading due to its limited structure and due to the possible only partial observation of the environment. (3) Non deterministic environments can further mislead the ALP.

2.3 The Genetic Generalization Pressure
The causes of over-specialization are overcome now with a genetic algorithm that is biased towards generalization. The GA works as follows:

- The GA is applied in the action set realizing a niche GA.
- In order to control the frequency of the GA application, the GA is applied in an action set only if the average time since the last GA application in the set is greater than the threshold \( \theta_{GA} \).
- Two classifiers are selected by roulette wheel selection using \( q^3 \) as the bid for each classifier.
- The parents remain in the population.
- Each specific attribute in the conditions is changed to a '−'-symbol with a probability \( \mu \).
- Crossover is applied with a probability \( \chi \).
- The resulting classifiers are only inserted into the population if they are not completely general.
- If the action set size exceeds the action set size threshold \( \theta_{as} \) as many classifiers are deleted in the action set as necessary.
- The deletion method uses tournament selection with sizes of 1/3 of the action set size. The classifier with the lowest quality is deleted. If two classifiers have approximately the same quality \( q \) (i.e. \( |q_1 - q_2| < 0.1 \)), then the one with more specific attributes in \( C \) is deleted. Thus, the deletion method is a niche deletion which favors formal generality.

If a newly generated classifier already exists, it is not directly inserted but the numerosity \( num \) of the old classifier is increased consequently representing the new one as well.

The generalization pressure is realized by the (1) biased selection which chooses high quality classifiers that consequently must be at least specific enough, by the (2) generalizing mutation which only generates more general offspring, and by the (3) biased deletion which prefers inaccurate but also over-specialized classifiers.

2.4 The Interaction of the ALP and the GA
While the ALP specializes and sometimes over-specializes, the GA fixes the over-specializations but also sometimes over-generalizes. In order to keep the two pressures in equilibrium we use subsumption deletion similar to XCS (Wilson, 1998). When a new classifier is generated, the ACS checks if a classifier
exists in the population that subsumes the new classifier. A classifier subsumes
a new classifier if it is (1) reliable, (2) experienced (i.e. the classifier was applied
more than 20 times), (3) not marked, (4) more general, and (5) has the same
action. If such a classifier exists, then the new classifier is not added to the pop¬
ulation. Furthermore, \( q \) or \( \text{num} \) of the old classifier is increased when the ALP
or the GA created the new one, respectively. Thus, the ALP and the GA does
not create over-specialized classifiers once the maximally general classifiers are
found. However, the GA still creates over-general classifiers out of the maximally
general ones. To prevent the over-generalization due to the GA, the specialization
pressure needs to be strong enough. As we will see in 3.1, the specialization
pressure can be influenced by the specificity threshold \( u_{\text{max}} \) (see Section 2.2).

3 The Multiplexer Problem

We chose the multiplexer problem, a one-step problem, to investigate the gener¬
alization capabilities in the ACS. In order to make the problem solvable we need
to code a perceptual causality. This is done by coding the reward into the per¬
ceived consequences. The task is especially hard as it mainly asks for a further
differentiation of the conditions but learning in the ACS is primarily based on
the correct anticipation of behavioral consequences (see Section 2).

The multiplexer function is defined for binary strings of length \( l = k + 2^k \). Its
value is determined by the address specified by the first \( k \) bits in the remaining \( 2^k \)
bits. The value of the string '100010' in the 6-Multiplexer function is 1, since the
first two bits address the third bit in the remaining four bits. The function can
also be written in disjunctive normal form: \( F_6 = \neg x_1 \neg x_2 x_3 \vee \neg x_1 x_2 x_4 \vee x_1 \neg x_2 x_5 \vee x_1 x_2 x_6 \). Thus, in the multiplexer problem there are \( 2^{k+1} \)
constellations to learn which can be classified as either correct or wrong.

As mentioned above, we need to code the reward into the perceptions in order
to supply a perceptual causality. Wilson (1995) published a reward map for the
multiplexer problem which divides the payoff distinguishing between the \( 2^k \)
possible constellations: Payoff = 300 + 200 * (int) first.k.bits + 100 * (addressed.bit)
for correct classifications and 200 * (int) first.k.bits + 100 * (addressed.bit) for
incorrect ones. Later, Wilson (1998) also solved the multiplexer problem with
a payoff distinguishing only between correct and wrong classifications (payoff
1000 and 0). Either variation may be coded into the perceptions of the ACS.
When coding the reward map, there are \( k + 2 \) additional bits which are always
0 in a given situation and change in the effects according to the reward map.
In order to supply always a change in the perceptions we start with one for
the lowest payoff possible instead of zero. When coding the payoff 1000/0, then
only one additional attribute is present that changes to one or to two when the
classification is wrong or correct, respectively.

One problem instance starts with the presentation of a random bit string
of length \( k + 2^k \) plus the additional attributes which are set to zero. Next, the
ACS chooses an action randomly (i.e. the classification) and the result is coded
into the perceptions, leaving the random bit string unchanged. After the ACS
learned from the consequences, one problem ends and a new random instance is
presented.
3.1 Trouble with Over-Specialization

The 11-Multiplexer problem was already investigated in Butz, Goldberg, and Stolzmann (2000b). Although the ACS solved the problem with a number of explored problems comparable to XCS, the generalization was not as good as in XCS. The population increased to a higher level and converged not as fast as the population in XCS. Further investigations showed that the higher population size was mainly caused by over-specialization at the beginning of the run. Because of the restriction in the structure of the mark the ACS is not able to find out which components are the most relevant ones to specialize and thus specializes all components at once. Since the GA is a niche GA and only works in specific action sets, these over-specialized classifiers hide in the population. They rarely match the current state and are therefore very seldom part of the GA. Thus, they stay in the population, although a correct, more general classifier was found long ago, which could subsume the over-specialized one(s).

There are a number of possibilities to fix this problem. One possibility is a population sweep where the accurate, more general classifiers absorb the over-specialized ones. Another possibility is to restrict the population size and execute a deletion in the population which preferably deletes over-specialized classifiers. These are global solutions tackling the problem from the top level. In this paper however, we approach the problem from the other side and consequently restrict the over-specialization. For doing that, the ACS has got the threshold \( u_{\text{max}} \), the maximal number of specialized attributes in the conditions that are predicted not to change. Restricting the number of attributes to specialize, the ACS will not generate over-specialized classifiers anymore and the population should converge further. This latter approach is applied herein.

Figure 1 shows the resulting performance of the ACS in the 11- and the previously unsolved 20-Multiplexer problem. The 'correct anticipations' measure is calculated by presenting a random instance of the problem, choosing the most probable classifiers for both classifications (i.e. the classifiers with the highest \( q \)), and monitoring if the effects were anticipated correctly by the classifiers. All results are averaged over ten runs. The parameter settings in this and the following experiments were \( b_q = b_r = 0.05, \theta_{GA} = 25, \chi = 0.0, \mu = 0.3, \theta_{as} = 20 \). Experiments with crossover did not influence the performance in this problem.

In both graphs we can observe how the population size and the knowledge depend on the constant \( u_{\text{max}} \). By limiting the number of specified unchanging attributes, the specialization pressure is decreased. When choosing a low threshold, the population stays small, but a complete knowledge is not necessarily reached because the GA over-generalizes and the ALP is too slow to recover. When choosing a higher threshold, the population size increases and the knowledge reaches 100%. However, when \( u_{\text{max}} \) is chosen too high, the convergence takes longer and complete knowledge is achieved more slowly since the ALP over-specializes and the GA is not able to generalize fast enough.

3.2 The Two Codings

The ACS is based on learning from anticipations. Thus, the more effects that distinguish a problem, the easier it should be to learn a problem for the ACS.
Fig. 1. The comparison of different $u_{max}$ values reveals the dependency of the size of the evolving population and the quality of the evolving knowledge.

Figure 2 shows exactly this in the 11- and 20-Multiplexer. When only one additional attribute is coded into the perceptions distinguishing between correct and wrong classification (i.e. the 1/2 coding), the ACS takes longer to evolve a complete knowledge and the population required is larger. When the whole reward map is coded into the perceptions, the different effects result in a faster partitioning of the population and thus enable the ACS to learn more quickly.

Moreover, it can be observed that the knowledge in the 1/2 coded problem does not reach 100% knowledge completely. Analyzing the problem revealed that sometimes 'parasitic' classifiers evolve. These classifiers are correct and maximally general but occupy half of the space of each of two good classifiers. Considering the 6-Multiplexer problem such a classifier would be 'l###l-r' (showing condition – action). The classifier occupies half of the space of classifier '11###1-1' and of classifier '10###1-1'. Once a parasitic classifier took over a niche the good classifiers get lost, and thus, half of their covered space is suddenly empty. This problem is currently unsolved and needs further investigation.

Fig. 2. The comparison of different codings shows that the ACS is able to increase performance when the complete reward map is coded into the perceptions.
However, the graphs show that the effect is minor and that the ACS is able to recover from this error, since the knowledge still reaches a very high level.

3.3 Comparing the ACS with XCS

In order to investigate the quality of the ACS performance further, this section gives a direct comparison to XCS. We emphasize that the comparison is one of apple and oranges. The ACS is basically learning C-R-E relations and thus an internal environmental representation, whereas XCS is learning C-R-payoff relations and thus a payoff-map of the environment. This difference results normally in a different area of application. However, the multiplexer problem helps to investigate generalization in both approaches.

The XCS results presented here are run with an XCS implementation which is based on the original Wilson (1995) paper. The GA is executed in the action set and subsumption deletion is used according to Wilson (1998). The constants are chosen as in the two mentioned papers. Similar to the XCS performance measure we use a payoff based performance measure for the ACS in the subsequent results. After each 'explore trial' one 'exploit trial' is executed. In this 'exploit trial' the better action is executed which is determined by a prediction array identical to XCS. In the prediction array, the reward prediction of each possible action is determined according to the quality $q$ and the reward prediction $r$ of each matching classifier with the according action:

$$Value(a) = \sum_{cl \in M_{sett}} (r_{cl} * q_{cl} * num_{cl}) / \sum_{cl \in M_{sett} \land A_{cl} = a} q_{cl} * num_{cl} \quad (2)$$

Since the learning mechanism in the ACS is based on the learning of effects we always give a payoff of 1000/0 to the ACS regardless of which coding is used. Figure 3 compares the performance in the reward map multiplexer task and the 1000/0 payoff learning task. It can be observed that both algorithms benefit from the reward map. Moreover, although the size of the population of the ACS increases to a slightly higher level than XCS in the beginning, it converges to a comparable level in all four tasks. The performance in classifying is similar to XCS. While the ACS beats the XCS performance in the beginning of a run, it takes the ACS slightly longer to converge to a 100% performance level. However, the convergence of the population size is very similar to the XCS convergence. This shows that the ACS is generating the desired accurate, maximally general classifiers.

4 Summary

This paper has studied the generalization capabilities of the ACS with a GA in the highly challenging multiplexer problem. Even when only one additional attribute created the perceptual causality, the ACS was able to scale up in the 11- and 20-Multiplexer task. Monitoring the reward learning, the performance was very similar to XCS's performance. Thus, transferring Wilson's conclusions to his generalization hypothesis (Wilson, 1995) and the investigations about the refinement to an optimality hypothesis (Kovacs, 1996), the ACS also generates accurate, maximally general classifiers and the population is converging.
to those classifiers. However, the GA is not working on its own but the process is a combination of a directed specialization due to the ALP and an indirected generalization due to the GA. Furthermore, the learned model is not a condition-action-payoff model but a condition-action-effect model of an environment. This generalized internal model gives the ACS a great advantage in comparison with all other approaches that form internal environmental representations.

5 Acknowledgments

The authors would like to thank Martin Pelikan and Stewart Wilson for valuable discussions and useful comments as well as the Automated Learning Group of NCSA at the University of Illinois in Urbana-Champaign for their support. The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, F49620-97-1-0050, and F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Additionally, research funding was provided by the

\[ \text{Fig. 3. The ACS and XCS in the multiplexer problem learning the reward-map and the 1000/0 payoff.} \]
German Research Foundation DFG. The views and conclusions contained herein are my own and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References


A New Bootstrapping Method to Improve Classification Performance in Learning Classifier Systems

John H. Holmes¹, Dennis R. Durbin¹,², Flaura K. Winston²

¹Center for Clinical Epidemiology and Biostatistics
University of Pennsylvania School of Medicine
Philadelphia, PA 19104 USA
{jholmes,ddurbin}@cceb.med.upenn.edu
²The Children’s Hospital of Philadelphia
Philadelphia, PA 19104 USA
flaura@mail.med.upenn.edu

Abstract. A new technique for improving the classification performance of learning classifier systems (LCS) was developed and applied to a real-world data mining problem. EpiCS, a stimulus-response LCS, was adapted to perform prevalence-based bootstrapping, wherein data from training and testing sets were sampled according to the prevalence of the individual classes, rather than randomly using the class distribution inherent in the data. Prevalence-based bootstrapping was shown to improve classification performance significantly on training and testing. Furthermore, this procedure was shown to enhance EpiCS’s classification performance on testing compared to a well-known decision tree inducer (C4.5) when similar bootstrapping procedures were applied to the latter.

1 Introduction

Learning classifier systems (LCS) are rule-based, message-passing systems that employ reinforcement learning and the genetic algorithm to learn rules that guide their performance in a given environment [6,8]. Knowledge is represented in the LCS by classifiers, which correspond to condition-action rules, wherein the condition is expressed as a vector of bits, integers, or real numbers. Classifiers are contained in a population of either fixed or variable size; this population is the LCS knowledge base, which evolves over time to represent a generalized model of the environment.

This investigation focused on supervised, two-class learning environments, where the LCS is trained with example cases having a known dichotomous classification, such as either diseased/not-diseased, or alive/dead. As a result, the LCS is informed by the training cases as to the overall class landscape, rather than having to discover the classes on its own. After a period of training has resulted in sufficient classification accuracy, the LCS is exposed to previously unseen cases with unknown class membership, to evaluate the system’s classification ability by applying to novel cases the generalized model of the environment developed during training.
The typical method for training a LCS is *bootstrapping*. This is a sampling method well known in the statistical sciences as a technique for reducing the variance of models for classification or prediction [5,19]. In the LCS domain, bootstrapping involves taking cases from a training set at random, with replacement, such that the system encounters each case many times over the training period. As the training period progresses, the LCS is exposed to increasing numbers of training cases, and partly as a result of this procedure, an optimally general model of the problem domain is constructed in the population of classifiers that comprise the LCS knowledge base. For problem domains where classes occur in equal proportions (in a two-class problem where positives and negatives are equal in number), this type of bootstrapping is quite acceptable, in that measures of classification accuracy are not affected by unequal class distributions. However, real-world problems seldom have this type of class distribution. As an example, consider cancer surveillance, where, even in the most devastating situation, it is extremely unlikely that the number of cancer cases would ever approximate that of non-cases.

The difficulty with this situation is that the system will be exposed to disproportionately small numbers of disease-positive cases. As a result of this, *classification pressure* is exerted on the system, causing evolving classification models to be more accurate in representing the dominant (disease-negative) class, at the expense of an accurate representation of the non-dominant (disease-negative) class. For example, if the non-dominant class is positive, and the dominant class is negative, classification pressure causes some positives to be classified as false negatives, simply because the classification models are biased by the surfeit of cases in the non-dominant class, which are negatives. There is a corresponding decrease in the true positive rate. At the same time, the effect on negatives is negligible, as they comprise the dominant class. This problem has been described in detail elsewhere [12,13]. While this phenomenon is an artifact of the mathematics for calculating true positive and negative rates, and is an annoyance in testing, it has serious consequences for model building during training.

“Committee-based” techniques such as bagging and boosting [1,3,7,14,17,18] have been used to improve model-building and classification performance. However, these techniques focus on ensembles of “problem cases” that are difficult to learn, often due to conflicts or contradictions in the data. While these types of cases no doubt exist in real world data, neither bagging nor boosting addresses the issue of unequal class prevalence as a cause for inaccurate classification.

A possible solution to this problem would ensure that the system is exposed to cases in equal proportion by class, while ensuring that, over time, it is exposed to all cases, regardless of class. This strategy should give the system the opportunity to develop a model that accurately generalizes to all of the classes in the environment. This paper describes such a technique, a new sampling methodology called *prevalence-based bootstrapping*. In prevalence-based bootstrapping, the distribution of the classes must be determined in advance of the start of actual sampling, so that a sampling weight can be determined for the dominant class. The derivation of the sampling weight is described in Section 2.1. For now, the weight is simply the ratio of dominant class cases to non-dominant cases. This approach ensures that the system is exposed to equal numbers of dominant and non-dominant cases at any one point in time, but over time, it is exposed to all cases in both classes.
This paper reports on the application of prevalence-based bootstrapping as a means to improve learning and classification accuracy in a LCS operating in a real-world clinical domain: the discovery of rules to predict the presence or absence of serious head injury in children involved in automobile crashes.

2 Methods

2.1 Testbed system: EpiCS

A stimulus-response LCS, EpiCS [9] was developed from the NEWBOOLE model [4] to meet the demands of epidemiologic data. The distinctive features of EpiCS include algorithms for controlling under- and over-generalization of data, a methodology for determining risk as a measure of classification [10], and the ability to use differential negative reinforcement of false positive and false negative errors in classification during training [12].

EpiCS was further modified in the following way to accommodate prevalence-based bootstrapping. First, a procedure was implemented to determine the distributions of the two classes in the data. This was performed by simply counting the number of positive and negative cases in the training and testing sets. Second, these counts were used to calculate the prevalence of each class (expressed as a proportion of the total number of cases) and in turn to determine the sampling weights. The weight for sampling from the non-dominant class was always 1: during prevalence-based bootstrapping, if a case was a member of the non-dominant class, it was always used. The weight for sampling from the dominant class was determined by dividing the prevalence of the dominant class by that of the non-dominant class. For example, if the prevalence of positive cases were 0.20 and that of negative cases 0.80, the weight would be:

\[
\text{Weight} = \frac{\text{Prevalence of Dominant Class}}{\text{Prevalence of Non-Dominant Class}} = \frac{0.80}{0.20} = 4.0
\]  

(1)

The weight corresponds to an \(n\)-sided coin, where \(n\) is the ratio of dominant-class cases to non-dominants. This weight would be used to determine the probability of sampling a dominant-class case from the data, given that this case was selected as a candidate by the random procedures used in traditional bootstrapping. The prevalence-based bootstrapping procedure calls for the system to generate a random number between 1 and the weight. In the example above, this random number would fall between 1 and 4, meaning that a dominant-class case would have a 25% chance of being selected. The effect of this procedure is to mitigate the natural classification pressure exerted by the dominant class on the non-dominant. Looked at another way, a non-dominant class case would have, over time, four times the probability of being selected, as compared to that if the bootstrap procedure were not used. That is, the non-dominant class could be over-sampled to the extent of the weight described in Formula (1). While this is effectively what happens in prevalence-based
bootstrapping, it is simpler to implement this approach from the sampling method described here. The pseudocode for this procedure, applied to the training phase, is shown below.

Determine sampling weight
While TrainingPhase
  Do
    Randomly select candidate training case
    If candidate case is member of non-dominant class
      Use as training case
    Else
      Generate random number between 1 and weight
      If mod(random number, selection_weight)=1
        Use as training case
      EndIf
    EndIf
  EndDo

A similar procedure was used in the testing phase, although each testing case was presented only once. Since it is known that classification pressure will artificially drive down the accuracy of classifications of the non-dominant class, it is important to keep the proportions of the two classes equal during testing. Thus, the bootstrap modification simply restricted the cases drawn for this phase to all training cases in the non-dominant class and an equivalent number of training cases from the dominant class, randomly selected without replacement. However, the procedure was repeated until all cases in the dominant class had been sampled.

2.2 Testbed Data: The Partners for Child Passenger Safety Project

The data for this investigation were captured by the Partners for Child Passenger Safety (PCPS) project, which is a long-term investigation into child occupant protection in automobile crashes [11]. The PCPS project uses State Farm Insurance Companies claims data from 15 states and the District of Columbia on automobile crashes involving at least one child less than 16 years of age.

From the more than 500 numeric data elements in the study, 47 were selected for their possible association with head injury. These elements were selected from a number of modules: passenger restraint (characteristics of the restraint devices and their usage, pertaining to an individual passenger); crash (characteristics of the crash event, such as point of impact, type of object struck, and estimated speed); kinematics (ejection from vehicle, contact with surfaces such as windows or dashboard, and evidence of occupant motion during the crash); and demographics (age and predisposing factors such as physical disability). As EpiCS does not yet support real-valued data representation, coding the data for use by EpiCS required transforming the dichotomous and ordered discrete values into binary representation. A
dichotomously-coded element was used to indicate the presence or absence of serious head injury defined as "serious" or worse by the Abbreviated Injury Scale (AIS) [2].

A total of 8,334 records comprised the pool of data. Training and testing sets were created from the pool by selecting records such that they were equal in size and mutually exclusive. In addition, care was taken to insure that positive and negative cases were distributed as equally as possible between the training and testing sets. The distribution of the sets is shown in Table 1.

| Table 1. Distribution of main PCPS head injury dataset. |
|---------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|                                 | Training        | Testing         |                 |                 |                 |                 |
| Positive | Negative | Total | Positive | Negative | Total |                 |                 |
| 208      | 3959     | 4167 | 207      | 3960     | 4167 |                 |                 |

2.3 Experimental Procedure

System initialization
The population in EpiCS was initialized with 5,000 randomly generated classifiers. The population size parameter was derived empirically by examining classification performance and classifier generalization during and at the end of training over a range of population sizes from 1,000 to 20,000. Low population sizes resulted in overgeneralization, while higher sizes resulted in poor generalization, due to excess or insufficient generalization pressure (respectively), a phenomenon described in [16]. The other parameters, such as crossover and mutation rates, penalty, and reward were set similarly to those used in [4].

Training
During training, cases were drawn from the training set using either the prevalence-based or traditional bootstrapping procedure described above; a single case presented to the system constituted one iteration. As these cases were drawn with replacement, an individual training case could be presented to the system many times during a training phase, which was defined as 100 iterations. A training epoch was defined as 10,000 iterations.

At every 100th iteration, the system moved from the training phase to the interim evaluation phase. During this phase, the learning ability of EpiCS was evaluated by presenting to the system for classification the taxon of cases drawn from the training set using a variant of the bootstrap procedure, as described above, or using serial sampling approach of the traditional bootstrap method for selecting testing cases. The decision advocated by EpiCS for a given testing case was compared to the known classification of the case. The decision type was classified in one of four categories: true positive, true negative, false positive, and false negative. From tallies of the four decision classifications, the performance evaluation measures were calculated (as described in Section 2.4) and written to a file for analysis.
Testing
At the completion of 10,000 iterations, the system moved into the testing phase, where the classification performance of EpiCS on the testing set was evaluated using the same method described above. The bootstrap method for this phase was the same as that used for the interim evaluation phase during training.

The entire cycle of training and testing comprised a single trial. A total of 20 trials using prevalence-based bootstrapping were performed for this investigation. Another 20 trials were performed without prevalence-based bootstrapping for comparison. These trials used the traditional method of bootstrapping, in which all cases were used, preserving the prevalence of the respective classes.

2.4 Evaluation Metrics
Two basic metrics were used to determine classification accuracy: sensitivity (the true positive rate, or the proportion of known positive cases classified by the system as positive) and specificity (the true negative rate, or the proportion of known negative cases classified by the system as negative). These metrics were used to derive a single metric, the area under the receiver operating characteristic curve, or AUC. Also included in this suite of metrics was the Indeterminant Rate, or IR, the proportion of cases that could not be classified by the system. These metrics are discussed in detail in [12,13].

2.5 Performance comparison method: Decision rule induction
A well-known decision tree induction program, C4.5 [15], was used to derive a set of classification rules from the training set that were applied to unseen cases in a testing set. C4.5, like any classification system, is affected by classification pressure when operating in environments with unequal class prevalence. In order to ensure comparability with the procedures applied to EpiCS, the prevalence-based bootstrap procedure was used to create 20 training-testing set pairs from the main dataset. These sets comprised equal numbers of positive and negative cases, and were used to derive 20 different decision tree models. From these models, the C4.5RULES procedure was used to create a parsimonious rule set for classification of the testing set, and the evaluation metrics discussed above were used to evaluate the classification accuracy of these models. In addition, the full training and testing sets were used to create a set of similar models, so that these could be compared to the EpiCS trials that used traditional bootstrapping.
3 Results

3.1 Training Epoch

Figure 1 illustrates the training epoch. The areas under the receiver operating characteristic curve (AUC) and the Indeterminant Rates (IR), obtained at each 100th iteration during training are plotted to demonstrate the difference between the bootstrapped and non-bootstrapped trials. As can be seen in the figure, the bootstrapped trials performed significantly better, achieving a low IR and high AUC within 1,000 iterations. The non-bootstrapped trials resulted in mediocre AUCs and IRs that were slow to resolve; even after 5,000 iterations, the IR for these trials averaged 12%. Although the shape of the curve of AUCs for the prevalence-based bootstrapped trials shows a continuing upward trend that appears asymptotic rather than fully convergent, it reaches 0.88 (a respectable AUC in most domains) within the first 1,000 iterations.

![Figure 1. Curves of Area under the Receiver Operating Characteristic Curve (AUC) and Indeterminant Rate (IR) averaged over all non-bootstrapped and bootstrapped trials.](image)

Table 2 shows the results of the testing phase averaged separately for all prevalence-based bootstrapped and traditionally bootstrapped trials. The prevalence-based bootstrapped trials performed significantly better on testing (p<0.001) on sensitivity, AUC, and IR. As expected, no difference in specificity was observed, given that this metric reflects the dominant (negative) class. The benefit of the low IR
is realized in that not only did the prevalence-based bootstrapped trials classify cases more accurately than did traditionally bootstrapped, but more testing cases were classifiable by the system. In addition, the prevalence-based bootstrapped trials performed significantly better on testing than C4.5 (p<0.0001), while the traditional bootstrapped trials were not significantly different between EpiCS and C4.5.

Table 2. Results obtained at testing averaged over all 20 non-bootstrapped and 20 bootstrapped trials. AUC=Area under the receiver operating characteristic curve; IR=Indeterminant rate. Numbers in parentheses represent one standard deviation.

<table>
<thead>
<tr>
<th>EpiCS Bootstrapped</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>AUC</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.83 (0.02)</td>
<td>0.90 (0.01)</td>
<td>0.87 (0.02)</td>
<td>0.03 (0.0)</td>
</tr>
<tr>
<td>EpiCS Non-bootstrapped</td>
<td>0.22 (0.08)</td>
<td>0.92 (0.01)</td>
<td>0.57 (0.04)</td>
<td>0.18 (0.09)</td>
</tr>
<tr>
<td>C4.5 bootstrapped</td>
<td>0.52 (0.05)</td>
<td>0.92 (0.01)</td>
<td>0.72 (0.03)</td>
<td>----</td>
</tr>
<tr>
<td>C4.5 non-bootstrapped</td>
<td>0.24 (0.08)</td>
<td>0.91 (0.02)</td>
<td>0.58 (0.04)</td>
<td>----</td>
</tr>
</tbody>
</table>

4 Discussion

This work investigated the application of a new, prevalence-based bootstrapping method for training and evaluating the learning and classification performance of EpiCS, a stimulus-response LCS. Rather than rely on the class distributions inherent in training and testing data, this method ascertained these distributions in order to derive a weight that determined how cases are sampled from data, prior to any sampling for training or testing. The purpose of this approach was to expose the LCS to equal numbers of cases from the classes, while ensuring that over time, the LCS was exposed to all cases. The goal of this approach was to improve classification performance during training and testing. The results of this investigation indicate that prevalence-based bootstrapping achieved this improvement, at least for the data used in this investigation.

In looking at the training phase (Figure 1), prevalence-based bootstrapping had a profound effect on reducing the IR, indicating that generalization was improved by this procedure. This was verified in looking at the classifier population after training, where the proportion of unique classifiers dropped from 98.2% (at initialization) to 28.9%. The improved generalization observed in the prevalence-based bootstrap trials is probably the result of decreased classification pressure, compared with that which is typically exerted in environments where classes are distributed unequally. For example, with classes equally distributed, it is unlikely that one class will be favored over the other in terms of representation in the classifier population. As a result, there is “breathing room” for classifiers representing both classes to evolve equally.

The new bootstrapping procedure had a substantial effect on the area under the receiver operating characteristic curve as well. First, the AUC for the prevalence-based bootstrapped trials was significantly higher than that for the traditional bootstrapped trials, a difference attained within the first 100 iterations, and maintained throughout the training epoch. Second, although the prevalence-based bootstrapped trials attained a higher AUC, the learning rate, as evidenced in the shallower slope of
the curve, was slower than that of the traditional bootstrapped trials. This effect could be due to the interim evaluation method, which caused all cases in the training set to be used in testing, even though the system would not have encountered all of these cases early on in the training epoch. It seems reasonable to assume that it will take longer to learn the concepts of cases when their presentation to the system is metered by the bootstrapping procedure. This explanation is borne out by the relatively stable performance of the traditionally bootstrapped trials, where all cases, in both classes, are eligible for use in training and interim evaluation. The relatively poor performance of the traditionally bootstrapped trials is due to the wholesale use of all cases in the training set at any point in time. However, there is little benefit to be realized in selecting this method for training EpiCS: its learning rate is somewhat faster, but its significantly poorer performance argues against its use, at least in these data.

The results obtained during the training epoch demonstrate clearly the effects of applying prevalence-based bootstrapping, but these do not tell the whole story of the advantages realized by this technique. Additional benefits are found in the classification performance seen on testing EpiCS with novel data.

Table 2 shows the superiority of the prevalence-based bootstrapped trials to classify previously unseen cases, not only over the traditional bootstrapped trials, but over C4.5 as applied to the same data using a similar form of bootstrapping to ensure comparability. The primary reason for this comparison was to demonstrate that the classification performance of EpiCS, which was comparable with traditional bootstrapping to C4.5, could be made significantly superior to C4.5 with prevalence-based bootstrapping. This could be an artifact of the data used here.

5 Conclusion

From the results of this investigation, it is clear that the use of a LCS in real-world domains, where class distributions may be uneven, should be approached cautiously. Wherever these types of data exist, the LCS should be a good candidate as a knowledge discovery tool, but only if the researcher is sensitive to the demands of the data. Unequal class distributions may lead to poor learning performance, as evidenced by the significantly higher IRs, and lower AUCs, found on the traditional bootstrapped trials. This appears to be driven by the natural bias in classification systems toward equal class distributions.

Based on prevalence, this bootstrapping method appears to address the issues associated with unequal class prevalence in a two-class, real-world problem, at least for the data used in this investigation. However, this approach could be extended to multi-class and even ordered-class domains, and this would be a fertile area for future research. A logical next step for LCS research is its application to knowledge discovery in databases (KDD) as a data mining methodology. The LCS will need to be able to address the complexities of data found in KDD problems, which certainly will include unequal class distributions. Further research into the application of prevalence-based bootstrapping to other, more complicated real-world problems will help determine if it is a sound technique for addressing this problem.
References

Towards Automatic Domain Knowledge Extraction for Evolutionary Heuristics

Márk Jelasity

Research Group on Artificial Intelligence,
University of Szeged, Hungary
jelasity@inf.u-szeged.hu
and
Leiden Institute of Advanced Computer Science,
Leiden University, The Netherlands

Abstract. Domain knowledge is essential for successful problem solving and optimization. This paper introduces a framework in which a form of automatic domain knowledge extraction can be implemented using concepts from the field of machine learning. The result is an encoding of the type used in most evolutionary computation (EC) algorithms. The approach focuses on whole problem domains instead of single problems. After the theoretical validation of the algorithm the main idea is given impetus by showing that on different subdomains of linear functions the method finds different encodings which result in different problem complexities.

1 Introduction and Motivation

Domain knowledge plays a key part in today’s machine learning applications. Though in many cases relatively simple heuristics combined with the brute force of available fast processors and millions of test samples seems to be the best available solution — like hidden Markov models used in speech recognition systems rather than an expert knowledge of phonemes [11], or simple Bayesian models employed in natural language processing applications instead of knowledge of grammar [6] — it is now generally accepted that for instance the performance of evolutionary heuristics depends heavily on the applied encoding and operators. In fact this is consistent with what the “no free lunch” theorems [12] tell us: there are no algorithms that are the best in each domain, so for the best performance in each domain one has to find the best algorithm for each separately. Hence the extraction of domain knowledge is essential.

Our previous research into the question also supports this view [4, 2, 3]. We have shown that the usual practice of simply characterizing domains by giving them labels such as “NP-hard” or “subset-sum problem” is not necessarily useful or even misleading. The actual structure and complexity of a domain (i.e. a set of functions defined over a common search space) depends on the source of these functions. For example a domain containing subset sum problems (i.e. “NP-complete” combinatorial optimization problems) may turn out to be a trivial domain due to the structure of the parameters of the particular functions in the domain. So even when a characterization is available (the function is not a black-box) the extraction of domain knowledge is still essential. In
Section 4 it will be shown that even when we know that a domain contains only linear functions the performance can still be significantly improved using domain analysis.

The problem is that extracting domain knowledge in general is quite a difficult problem; scientific researchers and engineers do this for a living and it is not one of the easiest jobs available. However, in systems where knowledge is explicitly and separately represented, it is possible to perform some kind of meta optimization over the domain of possible knowledge content. Evolutionary heuristics are good examples since knowledge is expressed in the encoding and operators while the basic algorithm remains the same. A lot of methods can be found in the literature that tackle the problem of dynamic problem structure analysis. A survey of methods using probabilistic models can be found in [8]. Other approaches concentrate on linkage detection [7]. A common feature of these methods is that they concentrate on single functions. Our goal here is different in that we would like to extract knowledge that characterizes a whole domain and is reusable and interchangeable. One area of research is relevant from this point of view, namely the work of Radcliffe [10]. Their basic ideas on the nature of knowledge to be extracted are not unlike those presented here (the differences being emphasized later on) but they did not tackle the problem of extracting knowledge automatically in a systematic way.

This paper introduces a framework in which automatic domain knowledge extraction is possible. In our case domain knowledge means the representation or encoding of the search space. Here, binary representations will be generated that are optimal in a sense to be described later. A binary representation is a mapping of the search space to a set of the binary strings \( \{0,1\}^n \). Though it is now widely accepted that an arbitrary binary representation is not necessarily better than an arbitrary non-binary encoding, our problem is a little different here. We are looking for the optimal binary representation in the space of all binary representations of a search space. Note that e.g. a search space of size \( 2^n \) has \( 2^n! \) different \( n \)-bit binary representations which is an enormous number. It is still possible that the optimal binary representation is not optimal in the space of all representations but here we do not tackle this problem.

In a binary representation every position of this string contains a 0 or a 1 which means that every position defines a concept over the search space. The term concept is used as in machine learning, i.e. a concept over a space is a subset of the space. Very briefly, our method is based on finding such concepts with the help of a measure over the concept space.

The outline of the paper is as follows. In Section 2 the basic concepts of the framework are defined. In Section 3 a useful property of the method is proved which supports applicability. Section 4 provides an illustrative but interesting example of the possible advantages of the approach on the class of linear functions. In Section 5 it will be shown that this method is in fact a generalization of some probabilistic methods used to model the distribution of good solutions of a function (see [8]). Finally, Section 6 discusses the possibilities and limitations of implementation of the framework in real-world, large-scale problems.
2 Framework

It is important to emphasize that this section will define only a general framework which may have many implementations depending on problem size, available time and data, etc. These details will be discussed in the following sections.

2.1 Basic Terms

Let us first define a problem. A problem is given by its search space, and a real valued objective function defined over it. The notation of the search space is \( S \). The notation of the function is \( f : S \rightarrow \mathbb{R} \). In other words \( f \in \mathbb{R}^S \). In evolutionary computation the objective function is usually called the fitness function. In this paper I will adopt this convention.

The problem domain is a subset of all possible fitness functions. The problem domain will be denoted by \( \mathcal{D} = \{ f_1, f_2, \ldots \} \subseteq \mathbb{R}^S \). This notion is crucial from our point of view. In practice the problem domain is given by the problem situation, e.g. a university which needs schedules for organizing its activity. The particular variables of the particular university — i.e. the number of employers, students, rooms, the sizes of rooms, the habits of each lecturer (who get up early/late, work at home/in their office etc.) and so on — will make the scheduling task special. The fitness functions in the domain will probably have a lot of features in common. At the same time, the scheduling task is an NP-complete combinatorial optimization problem in general. But this mathematical definition includes many more functions which are very diverse compared to the ones actually encountered at our university. To handle problem domains as an actual sample of functions and trying to describe them instead of using a given definition is therefore a main constituent of the philosophy of the present approach (see also [4, 2, 3]).

A concept over \( S \) is a subset of \( S \). The notation will be \( C \subseteq S \) while \( \overline{C} = S \setminus C \). In other words, using a function notation \( C \in \{0,1\}^S \). An encoding of \( S \) is given by an ordered list of concepts. The encoding will be denoted by \( C = (C_1, \ldots , C_n) \). Using this notation, the code of a solution \( x \in S \) is given by \( \mathcal{C}(x) = (C_1(x), \ldots , C_n(x)) \in \{0,1\}^n \). For the sake of simplicity these binary codes will be used throughout the paper noting that generalization is possible to other kinds of codes.

Next let us define two properties of encodings. The first is very important from a practical point of view: an encoding has to be invertible, i.e. given a code \( c \) of a solution, we should be able to effectively compute solutions \( x \in S \) for which \( \mathcal{C}(x) = c \). Note that it is possible that \( x \) is not unique. The second is related to the efficiency of the encoding. We want as few concepts as possible. To express this we call an encoding independent if its concepts are stochastically completely independent, i.e.

\[
\forall k, i_1, \ldots , i_k \quad Pr(x \in C_{i_1}, \ldots , x \in C_{i_k}) = Pr(x \in C_{i_1}) \ldots Pr(x \in C_{i_k})
\]

This seemingly contradicts other results from the GA literature, which report that non-coding segments (introns) may improve the search (e.g. [5]). This may be true under the assumption that the encoding is not optimal and so the genetic drift introduced by the small population-size has a larger impact. With optimal encodings which will be defined later on the genetic drift is a smaller problem while with few concepts the search space size reduces significantly.
2.2 Automatic Generation of Codes

The task is to generate the optimal encoding for a problem domain. If we define the notion of optimality then the problem reduces to a search problem over the possible encodings.

First let us define the optimality of a concept over a domain. We need a concept that separates good and bad solutions as clearly as possible in all of the functions in the domain. Good (or bad) solutions are defined as being in the upper half (or the lower half) of the search space with respect to a given fitness function. Let us denote the concept representing exactly the good solutions for the fitness function \( f \) by \( G_f \). Clearly every fitness function will define a different notion of good and bad solutions. For measuring the separation of good and bad solutions over a given fitness function \( \text{information gain} \) is an ideal choice. Information gain is a measure of "goodness" of cutting a space. Before cutting the space, the entropy of the space tells us how many bits are needed on average to encode if a random solution is good or bad. In the worst case one bit is needed (if the number of good solutions equals the number of bad ones) and in the best case no information is needed (0 bits) if all the solutions are good or bad. After cutting the entropy of the two resulting subspaces can be calculated. If the cut is good, these entropies will be smaller than the original entropy of the whole space. The difference of the average of the entropies of the two half spaces and the original entropy is the gain.

We use information gain as defined in the classical ID3 algorithm [9]. For a given fitness function \( f \) from the domain the information gain of a concept \( C \) is defined as follows:

\[
\text{gain}(G_f, C) = E\left(\frac{|G_f|}{|S|}\right) - \frac{|C|}{|S|} E\left(\frac{|G_f \cap C|}{|C|}\right) - \frac{|\bar{C}|}{|S|} E\left(\frac{|G_f \cap \bar{C}|}{|\bar{C}|}\right)
\]

where function \( E \) is the entropy defined by \( E(p) = -p \ln p - (1-p) \ln(1-p) \). Here \( p \) is the proportion of a given concept over the space under consideration. The natural logarithm was chosen because natural logarithm is equivalent to \( \log_2 \) as a measure of information according to information theory but our formulas will become simpler using \( \ln \). \( E(0) = E(1) = 0 \) while \( E(0.5) \) is maximal. This means that the information gain is maximal if \( C = G_f \) or \( \bar{C} = G_f \), and minimal (0) if \( C \) and \( G_f \) are independent. The measure we are seeking will be the average information gain of the concept over the functions in the domain. This measure is denoted by \( \text{gain}(C) \). This means that a concept is an \textit{optimal concept of the domain} \( D \) if it maximizes the average information gain over \( D \).

Since a useful encoding contains several concepts we need a method for finding additional concepts while preserving the mutual independence between the concepts. A good heuristic for doing this is to find the concepts iteratively, one by one, and then applying the definition of optimality to the subdomains defined by the inverse sets of the possible codes determined so far. For example two concepts define four possible codes. The inverse sets of these codes yield a classification of the search space defining four subsets. These subsets define four subdomains by restricting the functions of the original domain. An optimal concept can be found in each of these subdomains. Now let the third concept of the encoding be the union of these four optimal concepts. The rationale behind this heuristic is that this recursive construction ensures independence if
the optimal concepts divide the search space in two equal parts. Of course this will not be true in general.

This method provides us with a definition of the information gain of the concept $C_{i+1}$ given that $C_1, \ldots, C_i$ are known (denoted by $\text{gain}(C_{i+1} | C_1, \ldots, C_i)$). For this let us take the information gain values of $C_{i+1}$ restricted to each of the subspaces defined by the known concepts as described above and let the information gain of $C_{i+1}$ be the weighted average of these gains where the weights are proportional to the sizes of the corresponding subspaces. The algorithm for finding the optimal encoding is given in Figure 1. The algorithm stops when the gain of the new concept is not greater than the optimal information gain over a domain containing only random functions.

\[
\begin{align*}
C_1 &= \arg\max \text{gain}(C) \\
g &= \text{gain}(C_1) \\
i &= 1 \\
\text{while}(g > \text{random gain}) &\quad C_i = \arg\max_C \text{gain}(C | C_1, \ldots, C_{i-1}) \\
g &= \text{gain}(C_i | C_1, \ldots, C_{i-1}) \\
i &\leftarrow i + 1
\end{align*}
\]

Fig. 1. The algorithm for finding the optimal encoding.

3 Theoretical Foundations

In this section I will show that the algorithm described in Section 2 is optimal in an important sense: random domains are never divided by any concept if some assumptions hold. This means that the subdomains of the original domain defined by the inverses of the codes are either random or empty. We say that a subdomain is random if it contains only random functions i.e. the values of the functions are drawn from the same distribution. Besides this the random subdomains are maximal i.e. every larger domain becomes non-random. In other words it is impossible to gain more information from the space by refining the encoding and the information contained in the encoding cannot be expressed using fewer random classes.

According to our algorithm we have to find the optimal concept on a domain, the concept with the maximal information gain. Let as assume that our domain with search space $S$ contains a random subdomain $S_2$ (see Figure 2 for an illustration). I will show that for every concept that splits this random space there exists another concept which has a larger gain and which does not split $S_2$.

Now let us choose an arbitrary concept $C$. The dotted line in Figure 2 is the boundary of the concept. The subspace $S_1 = C \setminus S_2$ is the non-random part of $C$ and $S_3 = \overline{C} \setminus S_2$ is the non-random part of $\overline{C}$. The sizes of the classes are $|S_i| = N_i$, $i = 1, 2, 3$. $\pi_1$ is the value for which

\[
E(\pi_1) = \frac{1}{|D|} \sum_{f \in D} E\left(\frac{|G_f \cap C|}{|C|}\right)
\]
There are two such values since $E(p) = E(1 - p)$. Let $\pi_1$ be the smaller one. With a similar definition of $\pi_3$ the gain of $C$ now can be written as

$$gain(C) = K - \frac{|C|}{|S|}E(\pi_1) - \frac{|C|}{|S|}E(\pi_3)$$

where $K$ is a constant independent of $C$. Using this value we introduce a simplification assumption:

$$gain(C_d) = K - \frac{|C_d|}{|S|}E(p_1(d)) - \frac{|C_d|}{|S|}E(p_2(d)) \quad (1)$$

where

$$p_1(d) = \frac{\pi_1N_1 + 0.5d}{N_1 + d}, \quad p_2(d) = \frac{0.5(N_2 - d) + \pi_3N_3}{N_2 - d + N_3}$$

and $C_d$ is any concept that was created from $C$ by adding $d$ elements from $S_2$. The point is that we replace the average of the entropies with the entropy of the average of $|G_f \cap C|/|C|$ over the domain. This introduces some error since it is possible only for linear functions of probability variables and entropy is not linear. This error depends on the actual distribution of $|G_f \cap C|/|C|$ over the domain. The lower the variance the higher the accuracy of the approximation. Furthermore, recall that we have chosen the smaller values for $\pi_1$ and $\pi_3$ so we use only the first half of the entropy function (over the interval $[0, 0.5]$) and here (apart from the neighborhood of the ends of the interval) it is not very far from linearity, so the accuracy depends also on the actual values of $\pi_1$ and $\pi_2$.

Now we can prove the following theorem:

**Theorem 1.** Using the assumption in (1)

$$C_i = \arg \max_{C \in \{C_0, ..., C_{N_2}\}} gain(C)$$

holds only for $i = 0$ or $i = N_2$. 
Proof. We are looking for the maximum of the information gain

$$\text{gain}(C_d) = K - \frac{(N_1 + d)E(p_1(d)) + (N_2 - d + N_3)E(p_2(d))}{N_1 + N_2 + N_3}$$

The problem is equivalent to finding the minima of the counter of the fraction, the average entropy. I will show that this function is concave on the interval $[0, N_3]$ which directly proves the theorem. It is sufficient to show that the first term $(N_1 + d)E(p_1(d))$ is concave, the other term has a symmetrical structure and the sum of concave functions remains concave. The second derivative of the first term is

$$\frac{-1}{2(1 - \pi_1)N_1 + d} + \frac{-1}{\pi_1 N_1 + 0.5d}$$

which is negative so the proof is complete.

This theorem means that either $S_2$ is included in the optimal concept or it is excluded completely. Applying this to every subdomain that arises during the running of the algorithm we get the result mentioned in the introduction of this section. It is very interesting to briefly relate this finding to Radcliffe's notion of a good encoding [10]. According to his model, the equivalence classes of the encoding should have a low fitness variance. This can be applied not only to single functions but to domains as well since it is possible to take e.g. the average of the variances of the functions of the domain. This is a special case of our approach since if the low variance property holds over a subdomain then according to our approach it will be a good candidate for being an optimal concept since all the solutions will tend to be good or bad due to low variance and therefore the entropy will tend to be low. However in the case of random domains the variance is not necessarily low.

4 Structure in the Linear Domain

For illustration of the potentials of the approach let us take a closer look at a domain of special significance: the linear functions over the binary search space $S = \{0, 1\}^n$. A function $f : S \rightarrow \mathbb{R}$ is linear with the coefficient vector $a \in \mathbb{R}^n$ if

$$f(x) = a^T x = \sum_{i=1}^{n} a_i x_i$$

In this section three subdomains of the general linear functions will be studied. Every subdomain will have prototypical functions, but the definitions are intended to be fuzzy. The closer examination of these subdomains is useful for two reasons. The first is that we will see that on certain subdomains the efficiency of the search can be significantly improved. The second is that this discussion will illustrate a major point of this work: different domains may have dramatically different structure and thus different optimal encodings even if the mathematical description of the functions of the domains have the same form.

Some claims of the section are based on experimental data. In all the experiments 8 bit domains were used and a concept was implemented as an explicit characteristic
function (i.e. a list of 256 truth values). The concepts were optimized using a simple multistart hillclimber run until 10000 evaluations restarted when no one-bit change resulted in improvement. The other specific details are given in the subsections.

4.1 Orderable Problems

The coefficient vector of an orderable function contains numbers that differ in their order of magnitude significantly. In other words the coefficients (and thus the bits of a solution) can be ordered according to dominance. Prototypical examples are vectors with $|a_i| = 2^i, (i = 1, \ldots, n)$. It is easy to see that for orderable domains the optimal encoding will be the collection of schemata of length 1. As an additional benefit, the dominance order is also given by the algorithm.

4.2 Counting Problems

Here the coefficients do not differ in magnitude and they have the same sign. The coefficients of the prototypes of such problems are equal to a given constant: $a_i = c, (i = 1, \ldots, n)$. We have run experiments with domains containing 100 functions where the coefficients of a particular function were drawn from $[100, 120]$ (or $[-120, -100]$) to introduce some noise. Note that the value of solutions which have the same number of 1s is similar. Thus they generate random subdomains in the sense of Section 3.

The experiments confirmed our theoretical assumptions in that the concepts found during search never divided such a subdomain in any single run, only when the whole domain to divide was random. Surprisingly (to me), when trying to divide such a random domain the algorithm did find structure consistently. Closer analysis showed that this structure is due to the noise we introduced and can approximately be translated into an additional heuristic which says that in a space of solutions containing the same number of 1s divide the space using the bit which has the smallest coefficient on average.

Note that the length of the optimal encoding is proportional to $\log n$ so a significant reduction of the search space can be achieved while the fitness variance of the inverse sets of the codes is low therefore this subdomain with the optimal encoding is much easier than the general linear domain.

4.3 Hamming Problems

Here the coefficients do not differ in magnitude but they may have different signs. A prototypical example could be $a_i = (-1)^n, (i = 1, \ldots, n)$. The value of a Hamming function depends on the Hamming distance from a given binary vector.

Experiments were run using a 100 function domain where the coefficients of a particular function were drawn from $[100, 120]$ and their sign was random. The maximal information gain of the first concept that was found by the hillclimber was 0.048 with a variance of 0.003 (from 10 experiments). This value is quite low given that on completely random domains the expected maximal gain is around 0.01 according to our simulations, and in the case of counting problems this value is 0.39 on average. Analyzing the optimal first concepts we can define the following heuristic: divide the space
according to a one bit schema. Applying this heuristic explicitly we get a gain of 0.046 on average with a variance of 0.002 (10 experiments).

The conclusion is that the optimal encoding is the natural encoding as in the case of orderable problems but the information gain is significantly lower. This indicates that Hamming problems are harder than orderable problems since the fitness variance is much larger and the problems are much more sensitive to sampling error and genetic drift.

5 Probabilistic Models

The approach presented here can be considered as a generalization of search techniques that use dynamic probabilistic models to generate good solutions [8]. A probabilistic model of the good region of the space has a close relationship to our notion of concept. As mentioned earlier, a concept has to be invertible; we have to be able to generate solutions that satisfy a given concept. A probabilistic model is in fact a fuzzy concept which is of course invertible.

A practical implementation of the algorithm applied to a domain containing only a single function may be very similar to algorithms using probabilistic models since during the recursive building of the optimal encoding the gain of new concepts can be evaluated on solutions generated using the inverses of available codes. Furthermore — as a trivial extension — every subdomain can be labeled positive if the functions over it contain good solutions consistently (recall that large information gain requires only homogeneity), and emphasis can be moved to explore those regions further, even if the domain contains several functions.

6 Conclusions and Future Work

The purpose of this paper was to motivate, to theoretically ground and to illustrate an automatic encoding generation technique. We have seen that the method cuts search spaces along their “natural joints” in the sense that random domains are never cut in half. This also means — considering the structure of the algorithm as well — that the non-empty inverses of the optimal codes define either random domains or low fitness variance domains. It was demonstrated that even in the case of the linear functions three subdomains can be defined that have significantly different complexity. This also implies that similar or identical mathematical structure is not necessarily sufficient to characterize a domain: the distribution of the parameters of the functions is also essential [4, 2, 3].

Here I would like to touch on some problems of practical, real world applications and its limitations. One main problem to solve when implementing the system is to chose the actual representations of the abstract notion of concept. In the case of big spaces this representation is naturally a function class. The literature on machine learning provides us with an endless number of opportunities, the class of feedforward artificial neural networks (ANNs) is a good example. The only important constraint is the invertibility condition of the encoding.

Another important issue is the bias introduced by the chosen representation. When restricting ourselves to a specific function class we risk the possibility that we cannot
describe the structure of the domain under consideration. For example if parity of bits plays an important role in a binary domain then there is practically no chance to capture this using feedforward ANNs. However this is not a specific problem of the present approach: it is the problem of machine learning in general.

Finally, for finding good concepts samples of the functions in the domain are needed. Two natural approaches seem to be reasonable. The first is to use the trajectories that were produced by search algorithms on the functions of the domain. The other is to recursively generate new solutions based on the available concepts and to evaluate them. Both methods assume a larger time-scale than ordinary optimization methods but the output, the interchangeable and reusable knowledge about important problem domains may pay off in the long term.

References

New Algorithms and Metaphors
Expanding from Discrete to Continuous Estimation of Distribution Algorithms: The IDEA

Peter A.N. Bosman and Dirk Thierens

Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{peterb, Dirk.Thierens}@cs.uu.nl

Abstract. The direct application of statistics to stochastic optimization based on iterated density estimation has become more important and present in evolutionary computation over the last few years. The estimation of densities over selected samples and the sampling from the resulting distributions, is a combination of the recombination and mutation steps used in evolutionary algorithms. We introduce the framework named IDEA to formalize this notion. By combining continuous probability theory with techniques from existing algorithms, this framework allows us to define new continuous evolutionary optimization algorithms.

1 Introduction

Algorithms in evolutionary optimization guide their search through statistics based on a vector of samples, often called a population. By using this stochastic information, non-deterministic induction is performed in order to attempt to use the structure of the search space and thereby aid the search for the optimal solution. In order to perform induction, these samples are combined so as to generate new solutions that will hopefully be closer to the optimum. As this process is iterated, convergence is intended to lead the algorithm to a final solution.

In the genetic algorithm [11, 14] and many variants thereof, values for problem variables are often exchanged and subsequently individually adapted. Another way of combining the samples is to regard them as being representative of some probability distribution. Estimating this probability distribution and sampling more solutions from it, is a global statistical type of inductive iterated search. Such algorithms have been proposed for discrete spaces [2–4, 12, 13, 15, 17, 19, 21], as well as in a limited way for continuous spaces [5, 10, 15, 22, 23]. An overview of this field has been given by Pelikan, Goldberg and Lobo [20].

Our goal in this paper is to apply the search for good probability density models to continuous spaces. To this end, we formalize the notion of building and using probabilistic models in a new framework named IDEA. We show how we can adjust existing techniques to be used in the continuous case. We thereby define new evolutionary optimization algorithms. Using a set of test functions, we validate their performance.
The remainder of this paper is organized as follows. In section 2, we present the IDEA framework. In section 3 we describe a few existing algorithms that build and use probabilistic models. In section 4 we state some derivations of probability density functions (pdfs). We use the described algorithms and pdfs within the IDEA in our experiments in section 5. Topics for further research are discussed in section 6 and our final conclusions are drawn in section 7.

2 The IDEA

We write $a = (a_0, a_1, \ldots, a_{|a|-1})$ for a vector $a$ of length $|a|$. The ordering of the elements in a vector is relevant. We assume to have $l$ random variables available, meaning that each sample point is an $l$ dimensional vector. We introduce the notation $a(c) = (a_{c_0}, a_{c_1}, \ldots, a_{c_{|a|-1}})$ Let $\mathcal{L} = (0, 1, \ldots, l - 1)$ be a vector of $l$ numbers and let $Z = (Z_0, Z_1, \ldots, Z_{l-1})$ be a vector of $l$ random variables. We assume that we have an $l$ dimensional cost function $C(z(\mathcal{L}))$ which without loss of generality we seek to minimize. Without any prior information on $C(z(\mathcal{L}))$, we might as well assume a uniform distribution over $Z$. Now denote a probability distribution that is uniformly distributed over all $Z$ with $C(z(\mathcal{L})) \leq \theta$ and that has a probability of 0 otherwise, by $P^\theta(Z)$. In the discrete case we have:

$$P^\theta(Z)(z(\mathcal{L})) = \begin{cases} \frac{1}{|\{z(\mathcal{L})|C(z(\mathcal{L})) \leq \theta\}|} & \text{if } C(z(\mathcal{L})) \leq \theta \\ 0 & \text{otherwise} \end{cases}$$ (1)

Note that if we find $P^\theta^*(Z)$ where $\theta^* = \min_{z(\mathcal{L})}\{C(z(\mathcal{L}))\}$, a single sample drawn from $P^\theta^*(Z)$ provides an optimal solution $z(\mathcal{L})^*$. A probability distribution is made up of a probability density structure (pds) and a probability density function (pdf) for each element in the pds. In graphical models literature, a pds is also called a factorization. Let $a \cup b$ be the splicing of $a$ and $b$ such that the elements of $b$ are placed behind the elements of $a$, giving $|a \cup b| = |a| + |b|$. Using graphical modelling [9], we can denote any non-clustered pds with conditional probabilities $P(Z(a)|Z(b)) = P(Z(a \cup b))/P(Z(b))$. We let $\pi(\cdot)$ be a function that returns a vector $\pi(i) = (\pi(i)_0, \pi(i)_1, \ldots, \pi(i)_{|i|-1})$ of indices denoting the variables that $Z_i$ is conditionally dependent on. We call the graph that results when taking the $Z_i$ as nodes and having an arc from node $Z_i$ to node $Z_j$ if and only if $i \in \pi(j)$, the pds graph. The only required condition to be able to express any non-clustered pds using conditional probabilities, is that this graph needs to be acyclic. By using a permutation vector $\omega$, the definition of a pds $(\pi, \omega)$ that models conditional factorizations, can be formalized as follows:

$$P(\pi, \omega)(Z) = \prod_{i=0}^{l-1} P(Z_{\omega_i}|Z(\pi(\omega_i)))$$ (2)

such that $\forall i \in \mathcal{L} \ (\omega_i \in \mathcal{L} \land \forall k \in \mathcal{L}(i) (\omega_i \neq \omega_k))$

$\forall i \in \mathcal{L} \ (\forall k \in \pi(\omega_i) (k \in \{\omega_{i+1}, \omega_{i+2}, \ldots, \omega_{l-1}\}))$

Each $P(Z_{\omega_i}|Z(\pi(\omega_i)))$ from equation 2 is a special case multivariate conditional pdf. This means that any such conditional pdf along with a pds $(\pi, \omega)$
defines a probability distribution over \( Z \). In general, we denote a pds by \( f \). The pds is constrained to be of a certain form. For instance, in the case of equation 2, the constraints impose \( f \) to describe a directed acyclic graph. We denote the constrained space of all possible structures by \( \mathcal{C} \). A probability distribution over \( Z \) is then formally denoted by \( P_f(Z) \), \( f \in \mathcal{C} \).

Denote the largest function value of a selection of samples at iteration \( t \) by \( \theta_t \). We find a pds and estimate each pdf to best approximate \( P^{\theta_t}(Z) \). We can then sample from the resulting probability distribution to get more samples. By formalizing this rationale in an iterative algorithm, we define the Iterated Density Estimation Evolutionary Algorithm (IDEA):

| IDEA\((n, \tau, m, \text{sel}(), \text{rep}(), \text{ter}(), \text{sea}(), \text{est}(), \text{sam}())\) |
|-----------------|-----------------|-----------------|
| Initialize an empty vector of samples | \( \mathcal{P} \leftarrow () \) |
| Add and evaluate \( n \) random samples | \textbf{for} \( i \leftarrow 0 \) to \( n - 1 \) \textbf{do} |
| Initialize the iteration counter | \( \mathcal{P} \leftarrow \mathcal{P} \cup \text{NEWRANDOMVECTOR()} \)
| Iterate until termination | \( c[\mathcal{P}_i] \leftarrow C(\mathcal{P}_i) \)
| Select \( \lceil \tau n \rceil \) samples | \( t \leftarrow 0 \)
| Set \( \theta_t \) to the worst selected cost | \textbf{while} \( \neg \text{ter}() \) \textbf{do} |
| Search for a pds \( f \) | \( (z^0(L), z^1(L), \ldots, z^{\lceil \tau n \rceil - 1}(L)) \leftarrow \text{sel}() \)
| Estimate each pdf in \( \hat{P}_f(Z) \) | \( \theta_t \leftarrow c[z^{\ell}(L)] \) such that |
| Create an empty vector of new samples | \( \forall i \in \mathcal{N}, \{c[z^i(L)] \leq c[z^{\ell}(L)]\} \) |
| Sample \( m \) new samples from \( \hat{P}_f(Z) \) | \( \mathcal{O} \leftarrow () \)
| Replace a part of \( \mathcal{P} \) with a part of \( \mathcal{O} \) | \textbf{for} \( i \leftarrow 0 \) to \( m - 1 \) \textbf{do} |
| Evaluate the new samples in \( \mathcal{P} \) | \( \mathcal{O} \leftarrow \mathcal{O} \cup \text{sam}() \)
| Update the generation counter | \textbf{for} each unevaluated \( \mathcal{P}_i \) \textbf{do} |
| Denote the required iterations by \( t_{\text{end}} \) | \( c[\mathcal{P}_i] \leftarrow C(\mathcal{P}_i) \)
| | \( t \leftarrow t + 1 \)

In the IDEA framework, we have that \( \mathcal{N}_\tau = (0, 1, \ldots, \lceil \tau n \rceil - 1), \tau \in [\frac{1}{n}, 1] \), \( \text{sel}() \) is the selection operator, \( \text{rep}() \) replaces a subset of \( \mathcal{P} \) with a subset of \( \mathcal{O} \), \( \text{ter}() \) is the termination condition, \( \text{sea}() \) is a pds search algorithm, \( \text{est}() \) estimates each pdf and \( \text{sam}() \) generates a single sample from \( \hat{P}_f(Z) \). The notation \( \hat{P}() \leftarrow \hat{P}_f \) means that \( \hat{P}() \) is one of the pdfs that is implied by the model \( f \).

The IDEA is a true evolutionary algorithm in the sense that a population of individuals is used from which individuals are selected to generate new offspring with. Using these offspring along with the parent individuals and the current population, a new population is constructed. By referring to the iterations in the IDEA as generations, the evolutionary correspondence is even more obvious.

Note that in the IDEA, we have used the approximation notation \( \hat{P}^{\theta_t}(Z) \) instead of the true distribution \( P^{\theta_t}(Z) \). An approximation is required because the determined distribution is based upon samples and the underlying density model is an assumption on the true model. This means that even though we might achieve \( \hat{P}^{\theta_t}(Z) = P^{\theta_t}(Z) \), in general this is not the case.
If we set $m$ to $(n - \lceil \tau n \rceil)$, $sel()$ to selection by taking the best $\lceil \tau n \rceil$ vectors and $rep()$ to replacing the worst $(n - \lceil \tau n \rceil)$ vectors by the new sampled vectors, we have that $\theta_{k+1} = \theta_k - \varepsilon$ with $\varepsilon \geq 0$. This assures that the search for $\theta^*$ is conveyed through a monotonically decreasing series $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{\text{end}}$. We call an IDEA with $m$, $sel()$ and $rep()$ so chosen, a monotonic IDEA.

If we set $m$ in the IDEA to $n$ and set $rep()$ to replace $\mathcal{P}$ with $\mathcal{O}$, we obtain the EDA by Mühlenbein, Mahnig and Rodriguez [17]. In the EDA however, the threshold $\theta_\varepsilon$ cannot be enforced. Note how EDA is thus an instance of IDEA.

### 3 Probability density structure search algorithms

In order to search for a pds, a metric is required that guides the search. In effect, this poses another optimization problem. The metric we use in this paper is a distance metric to the full joint pds $(\pi^+, \omega^+)$, $\forall i \in \mathcal{L} \omega^+ = i \land \pi^+(i) = (i + 1, i + 2, \ldots, l - 1)$. The distance metric is defined by the Kullback-Leibler (KL) divergence. We write $\mathcal{Y}$ instead of $\mathcal{Z}$ from now on to indicate the use of continuous random variables instead of either the discrete or continuous case. Using our definitions, the KL divergence can be written as [7]:

$$D(\hat{P}_{(\pi^+, \omega^+)}(\mathcal{Y}) || \hat{P}_{(\pi, \omega)}(\mathcal{Y})) = -\frac{1}{I} \sum_{i=0}^{I-1} h(\hat{P}(Y_{\omega_i}|Y(\pi(\omega_i)))) \quad (3)$$

Let $a \subseteq \mathcal{L}, b \subseteq \mathcal{L}$ where $a \subseteq \mathcal{L}$ means that $a$ contains only elements of $\mathcal{L}$. In equation 3, $h(Y(a))$ is the multivariate differential entropy and $h(Y(a)|Y(b))$ is the conditional differential entropy. Let $dy(a) = \prod_{i=0}^{[a]-1} dy_i$ be shorthand notation for the multivariate derivative. We then have:

$$h(P(Y(a))) = -\int P(Y(a))(y(a))ln(P(Y(a))(y(a)))dy(a) \quad (4)$$

$$h(P(Y(a)|Y(b))) = h(P(Y(a \cup b))) - h(P(Y(b))) \quad (5)$$

As the term $h(\hat{P}_{(\pi^+, \omega^+)}(\mathcal{Y}))$ in equation 3 is constant, an algorithm that searches for a pds can use the KL divergence by minimizing the sum of the conditional entropies imposed by $(\pi, \omega)$. This will cause the pds search algorithm to search for a pds as close as possible to $(\pi^+, \omega^+)$ subject to additional constraints.

The probabilistic models used in previously proposed algorithms range from lower order structures to structures of unbounded complexity. It has been empirically shown by Bosman and Thierens [6] that a higher order pds is required to solve higher order building block problems. We shortly state three previously introduced pds search algorithms that we use in our experiments.

In the univariate distribution, all variables are regarded independently of each other. The PBIL by Baluja and Caruana [2], the cGA by Harik, Lobo and Goldberg [13], the UMDA by Mühlenbein and Paafß [18], and all known approaches in the continuous case prior to the IDEA [10, 22, 23], use this pds. It can be modelled by $\forall i \in \mathcal{L} \pi(i) = () \land \omega_i = i$, giving: $\hat{P}_{(\pi, \omega)}(\mathcal{Z}) = \prod_{i=0}^{l-1} \hat{P}(Z_i)$. 
In the MIMIC algorithm by De Bonet, Isbell and Viola [4], the pds is a chain which is constrained to $\pi(\omega_{i-1}) = () \land \forall i \in \mathcal{L}_{(i-1)} \{ \pi(\omega_i) = (\omega_{i+1}) \}$, giving 
$$
\dot{P}(\pi, \omega)(Z) = (\prod_{i=0}^{t-2} \dot{P}(Z_{\omega_i}|Z_{\omega_{i+1}})) \dot{P}(Z_{\omega_t}).
$$
To find the chain, an $O(t^2)$ greedy approximation algorithm is used in MIMIC to minimize the KL divergence.

If the pds is constrained so that in addition to having an acyclic pds graph, each node may have at most $\kappa$ parents, the pds is constrained to $\forall i \in \mathcal{L} \{ |\pi(i)| \leq \kappa \}$. This general approach is used in the BOA by Pelikan, Goldberg and Cantú-Paz [19], as well as the LFDA by Mühlenbein and Mahnig [16] and the EBNA by Larrañaga, Etxeberria, Lozano and Peña. In the case of $\kappa = 1$, a polynomial time algorithm can be used to minimize the KL divergence [5]. In the case of $\kappa > 1$, a greedy algorithm is used that iteratively adds arcs to the pds graph.

There are other special case algorithms, such as the optimal dependency trees approach by Baluja and Davies [3] and the ECGA by Harik [12]. Like the LFDA, the ECGA uses minimum description length as a search metric. This metric has the advantage that the resulting pds will not be overly complex. Using the KL divergence, this can only be influenced by adjusting $\kappa$ because the KL divergence is merely a distance measure from a certain pds to $(\pi^+, \omega^+)$. We only regard the three described (see also) algorithms in combination with the KL divergence metric. Note that using the KL metric and the $(\pi^+, \omega^+)$ pds is merely an instance of the IDEA framework. This is also the case for using a certain pdf. The framework is to be seen separately from the algorithms that can be modelled by it.

4 Probability density functions

Next to the pds search algorithms from section 3, we require to specify a pdf to use. It follows from sections 2 and 3 that we require to know the multivariate differential entropy as well as the conditional pdf. In this section, we specify two well known pdfs that we use in our experiments within the IDEA framework.

A widely used parametric pdf is the normal pdf. Let $\mathcal{S} = (y^0, y^1, \ldots, y^{|\mathcal{S}|-1})$ be the set of selected samples. The sample average in dimension $j$ is then $\overline{Y}_j = \frac{1}{|\mathcal{S}|} \sum_{i=0}^{|\mathcal{S}|-1} y^i_j$. The sample covariance matrix over variables $Y(a)$ is $S = \frac{1}{|\mathcal{S}|} \sum_{i=0}^{|\mathcal{S}|-1} (y^i(a) - \overline{Y}(a))(y^i(a) - \overline{Y}(a))^T$. Let $s'_{ij} = S^{-1}(i, j)$. The conditional pdf and the entropy can be stated as follows [5]:

$$
\int \mathcal{N}(y_{\omega_0}|y(a - \omega_0)) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y_{\omega_0} - \mu)^2}{2\sigma^2}}
$$

where $\sigma = \frac{1}{\sqrt{s'_{00}}}$, $\mu = \frac{Y_{\omega_0}s'_{00} - \sum_{i=0}^{|\mathcal{S}|-1} (y_{\omega_i} - \overline{Y}_{\omega_i})s'_{i0}}{s'_{00}}$

$$
h(Y(a)) = \frac{1}{2} (|a| + \ln((2\pi)^{|a|} \det(S)))
$$

The non-parametric normal kernels pdf places a normal pdf over every available sample point. Let $s_i$ be a fixed standard deviation in the $i$-th dimension. The conditional pdf and the entropy can then be stated as follows [8]:

$$
\int \mathcal{N}(y_{\omega_0}|y(a - \omega_0)) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y_{\omega_0} - \mu)^2}{2\sigma^2}}
$$

where $\sigma = \frac{1}{\sqrt{s'_{00}}}$, $\mu = \frac{Y_{\omega_0}s'_{00} - \sum_{i=0}^{|\mathcal{S}|-1} (y_{\omega_i} - \overline{Y}_{\omega_i})s'_{i0}}{s'_{00}}$

$$
h(Y(a)) = \frac{1}{2} (|a| + \ln((2\pi)^{|a|} \det(S)))
$$
\[ f_{\mathcal{N}_k}(y_{a_0}|y(a - a_0)) = \sum_{i=0}^{S-1} \nu_i \frac{1}{s_{a_0} \sqrt{2\pi}} e^{-\frac{(y_{a_0} - y_i)^2}{2s_{a_0}^2}} \]  
\[ = e^{-\frac{\sum_{j=1}^{S-1} (y_{a_j} - y_i)^2}{2s_{a_j}^2}} \]  
where \( \nu_i = \frac{-\sum_{j=1}^{S-1} (y_{a_j} - y_i)^2}{\sum_{k=0}^{S-1} e^{-\frac{\sum_{j=1}^{S-1} (y_{a_j} - y_k)^2}{2s_{a_j}^2}}} \) 

\[ h(Y(a)) = \]  
\[ \frac{1}{2} \ln \left( |S|^2 (2\pi)^{|\alpha|} \prod_{j=0}^{a-1} s_{a_j}^2 \right) - \int f_{\mathcal{N}_k}(y(a)) \ln \left( \sum_{i=0}^{S-1} e^{-\frac{\sum_{j=1}^{S-1} (y_{a_j} - y_i)^2}{2s_{a_j}^2}} \right) dy(a) \]  

An alternative pdf to the two described above, is the histogram pdf. Using this pdf however does not scale up very well [7] and leads to an exponential iteration running time in the order of \( r^n \) where \( r \) is the amount of bins to use in each dimension. The normal pdf is very efficient but very cluster insensitive. The normal kernels pdf is very sensitive to clusters but may very quickly overfit the data. In addition, the running time each iteration for using the latter pdf tends to be a lot greater than for the normal pdf.

5 Experiments

We have used the following continuous function optimization problems:

| \( C_1 \) | \[ \frac{1}{4000} \sum_{i=0}^{l-1} (y_i - 100)^2 - \prod_{i=0}^{l-1} \cos \left( \frac{y_i - 100}{\sqrt{i+1}} \right) + 1 \] | \([-600, 600]^l\) |
| \( C_2 \) | \( \gamma_i = \frac{24}{1000} (i + 2) - y_i \) | \([-3, 3]^l\) |
| \( C_3 \) | \( \gamma_0 = Y_0, \gamma_i = y_i + \gamma_{i-1} \) | \([-3, 3]^l\) |
| \( C_4 \) | \( \gamma_0 = Y_0, \gamma_i = y_i + \sin(\gamma_{i-1}) \) | \([-3, 3]^l\) |

Function \( C_1 \) is Griewank's function and \( C_2, C_3 \) and \( C_4 \) are test functions by Baluja. For the latter three functions we have to maximize \( 100/(10^{-5} + \sum_{i=0}^{l-1} |\gamma_i|) \). Griewank's function should be minimized.

We use monotonic IDEAs with the normal pdf and the normal kernels pdf. Furthermore, we use the KL metric and truncation selection. The amount of available samples \( |\tau n| \) strongly influences the effectiveness of density estimation. We expect a better performance if this amount goes up. Therefore, we fix \( \tau \) and increase \( n \). To be more precise, we use the rule of thumb by Mühlenbein and Mahnig [16] for FDA and set \( \tau \) to 0.3. If all of the solutions differed by less than \( 5 \cdot 10^{-7} \), termination was enforced. The \( s_i \) standard deviation parameters for the normal kernels pdf were determined as \( (\alpha \cdot \text{range}^i)/|\tau n| \) with \( \alpha = 3 \) for \( C_1 \).

Our results are presented using the notion of relative function evaluations \( RFE = n_e \tau T \), where \( n_e \) is the required amount of function evaluations. Let
FT(x) be the time spent to perform x random function evaluations and let TT be the average total algorithm time spent including the n_e function evaluations. The relative time is defined as RT = (TT - FT(n_e))/FT(n_e). The RFE metric is a cpu independent fair running time comparison.

Functions C_1 and C_2 can be optimized efficiently by determining a value for each variable separately. This is not the case for functions C_3 and C_4. We therefore only used the univariate distribution on C_1. In figure 1, the scalability on C_1 and C_3 is shown, computed over 20 runs. We only used the normal pdf on C_3. The RFE at the minimal value of n at which C_1 was minimized to a value to reach (VTR) of 10^{-6} in all runs is shown on a linear scale. The computation time scales approximately linearly for the IDEA variants. For l \in \{250, 300\}, the VTR was never reached for the normal kernels. By allowing \alpha to vary adaptively, thereby obtaining a more flexible density estimator, this might be overcome. For C_3, we used a VTR of 5. The results are shown on a logarithmic scale. The true pds of C_3 can be seen to be the full joint distribution. Using this pds scales up polynomially, whereas using the univariate model scales up exponentially. Using the graph search with \kappa = 1 seems to scale up polynomially, but tests for a larger value of l need to be run to be sure.

![Fig. 1. Results on C_1 (left, linear) and C_3 (right, logarithmic) for increasing dimension.](image)

We compared the IDEA using the normal pdf to Evolution Strategies [1] (ES) on C_1. The ES has a (\mu, \lambda) strategy with \lambda = 7\mu and independent mutations using either individual standard deviations n_\sigma = l or a single standard deviation n_\sigma = 1. We initialized the standard deviations to 3.0 and used \tau_\sigma = 1/\sqrt{2l} and \tau_\mu = 1/\sqrt{2l}. Table 2 shows the success rate (SR), which is the relative amount of times the VTR of 10^{-6} was reached. The results are computed over 10 runs. The parameters for the ES are indicated by (n_\sigma, \mu, \lambda) and for the IDEA by (n, \tau). We allowed 10^6 evaluations if l = 10 and 10^8 evaluations if l = 300. In all cases except for ES (l, 30, 210) for l = 300 and IDEA (500, 0.3) for l \in \{10, 300\}, premature convergence resulted in SR < 100%. In the other two cases, the maximum of evaluations was reached before convergence in unsuccessful runs.

The continuous PBIL approach by Sebag and Ducoulombier [22] was tested on functions C_2, C_3 and C_4. We used IDEAs with the KL metric and the normal pdf in combination with the univariate distribution, the chain search algorithm, the exact graph search algorithm for \kappa = 1 and a fixed chain according to the
function definitions, so $\omega_i = l - i - 1$, $\pi(0) = ()$ and $\pi(i) = i - 1$, $i \geq 1$. For a given pdf, results obtained with this pdf is an upper bound for any pdfs in which each node may have at most a single parent. We also tested the full joint pdf. We have used $l = 100$, a maximum of $2 \cdot 10^5$ evaluations and we have averaged the results over 20 runs. We increased $n$ in steps of 25 to find the best value for $n$. Repeating earlier reported results [22], table 3 indicates that our approaches perform better. We note that using the full joint pdfs requires very large values for $n$. As a result, the amount of generations is very small since we are allowed only $2 \cdot 10^5$ evaluations. This strongly influences the results.

<table>
<thead>
<tr>
<th>Method</th>
<th>$C_2$, $l = 100$</th>
<th>$C_3$, $l = 100$</th>
<th>$C_4$, $l = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10 + 50)-ES</td>
<td>399.07</td>
<td>2.91</td>
<td>7.56</td>
</tr>
<tr>
<td>PBIL (Binary)</td>
<td>16.43</td>
<td>2.12</td>
<td>4.4</td>
</tr>
<tr>
<td>PBIL (Gray)</td>
<td>366.77</td>
<td>2.62</td>
<td>5.61</td>
</tr>
<tr>
<td>PBIL$_{C}$</td>
<td>4803</td>
<td>4.76</td>
<td>11.18</td>
</tr>
<tr>
<td>IDEA NoU</td>
<td>9999999.87</td>
<td>4.51</td>
<td>13.40</td>
</tr>
<tr>
<td>IDEA NoC</td>
<td>9999999.90</td>
<td>5.30</td>
<td>14.85</td>
</tr>
<tr>
<td>IDEA NoF</td>
<td>9999999.96</td>
<td>7.50</td>
<td>22.73</td>
</tr>
<tr>
<td>IDEA NoF</td>
<td>9999999.88</td>
<td>4.51</td>
<td>13.40</td>
</tr>
<tr>
<td>IDEA NoF</td>
<td>1.81</td>
<td>4.51</td>
<td>13.40</td>
</tr>
</tbody>
</table>

Fig. 3. Results on $C_2$, $C_3$ and $C_4$ in 100 dimensions.

6 Discussion

Selecting what value of $\alpha$ to use for the normal kernels pdf is dependent on the optimization problem. The value of $\alpha$ determines the smoothness of the fit [7], which makes it intuitive that increasing $\alpha$ on smooth functions should give better results. In our experiments, we empirically determined $\alpha$, but it is worthwhile to investigate how to use $\alpha$ adaptively.

Using the normal kernels pdf quickly tends to overfit the sample vector. This can somewhat be regulated by $\alpha$, but not entirely. The tendency to overfit became apparent as the approach was highly sensitive to the value of $[\pi n]$. Using the normal pdf on the other hand almost always underfits a sample vector. The normal mixture pdf with regularization is therefore an interesting trade-off that seems very worthwhile to investigate next for multiple reasons.

In this paper, we have used the KL metric. The drawback of this metric is that it is merely a distance metric to the fully joint distribution. This means that unless the additional constraints on the pdfs are very specific, a pdfs search algorithm will most likely not result in using the problem structure in an effective
way. Therefore, metrics such as the minimum description length or other direct conditionality tests are strongly worth investigating. The algorithms tested in this paper should be seen independently from the IDEA framework.

When we increase the size $l$ of a problem, we are estimating a probability distribution in a highly dimensional space. Using the full joint pdfs in that case poses problems for an IDEA. Estimating joint pdfs in highly dimensional spaces can require a large amount of time as well as many samples to justify the estimation because of the curse of dimensionality. However, the assumption that the cost function is built up of bounded lower order interactions between the problem variables is usually made. This implies that the actual pdfs that are being estimated, are of a lower order. If the optimization problem is not built up of such bounded lower order building blocks, using IDEAs is potentially a non-scalable approach, depending on the pdf that is used and the problem definition.

7 Conclusions

We have used the algorithmic framework IDEA for modelling iterated density estimation evolutionary algorithms. These algorithms make use of density estimation techniques to build a probability distribution over the variables that code a problem in order to perform optimization. To this end, a probability density structure must be found and subsequently be used in density estimation. For a set of existing search algorithms, we have applied and tested them in the IDEA framework using two different density estimation models.

The experiments indicate that building and using probabilistic models for continuous optimization problems is promising. This, in combination with its modelling capabilities, shows that the IDEA is general, applicable and effective.

References


A New Genetic Algorithms Working on State Domain Order Statistics

Delahaye Daniel, Puechmorel Stephane
puechmor@recherche.enac.fr
Tel : + 33 5 62 17 41 52
ENAC Math Department
7, Avenue Edouard Belin
31055 Toulouse, France

Abstract. This paper presents a new concept of Genetic Algorithm in which an individual is coded as a domain of the state space and is evaluated with the help of order statistics. For this first version only continuous criteria has been investigated. An hypercube domain of the state space is associated with each individual and is randomly sampled according to a distribution for which asymptotic extremes are known. Regular fitnesses are computed for all the samples in each domain and are combined to produce a prospectiveness criterion. A regular GA and this new GA are compared on classical N dimensional functions such as Sphere, Step, Ackley, Griewank for different values of N. A final comparison is given on the classical Lennard-Jones Molecular Conformation problem with 30 atoms.
For both versions, a regular GA has been used; the first one works on state points and the other one on state domains. For all tests, and for the same number of criterion evaluations, this new algorithm performs much better than the classical one.

Keywords: order statistics, state domains, genetic algorithm.

1 Introduction

At the beginning of GA, binary coding was mainly used to encode optimization problem (a good description of this kind of coding may be found in [2]). This coding is easy to manipulate and is very adapted for discrete problems with binary decision variables. The schema theory can be applied to this kind of GA and produce some convergence theorems. Unfortunately, this coding is not adapted for real vector optimization problems and direct real coding has been developed [4]. Usually GAs work on state points and evaluate fitnesses on those points with succession of exploration and exploitation phases. The exploitation phase is guided by the selection and the exploration phase is guided by the crossover and mutation operators.

In order to enforce and speed up those two phases, the present algorithm works on state domains instead of state points. The principle is the same as the
one used for the Branch and Probability Bound [6]. For this B&B, the global state domain is split into smaller parts which are evaluated with the help of order statistics. The probability that the global optimum belongs to a domain is then used to guide the branching phase.

The present GA works the same way by coding each individual by an N dimensional hypercube and uses order statistics to produce the associated prospectiveness and then guide the exploration or the exploitation.

In a first part, a brief description of order statistics is given. The second part describes how order statistics may be used to strongly enhance the performance of a regular GA and how coding, fitness evaluation and operators have been implemented. Finally, the third part gives some comparisons of both algorithms on hard to optimize mathematical functions.

2 Order Statistics

Let \((x_1, \ldots, x_N)\) be a sample of \(N\) iid random variables. The orders statistics associated with that sample is \((\epsilon_1, \ldots, \epsilon_N)\) where \(\epsilon_i\) is the \(i\)-th largest value of \((x_1, \ldots, x_n)\). It's well known from classical extreme statistics theory[3] that under mild assumptions the distribution of the max of \(N\) iid random variable with common density function will converge point wise to one of the Frechet, Weibull or Gumbell distributions. More precisely, one may find two sequences \((a_n)_{n \in \mathbb{N}}, a_n > 0\) and \((b_n)_{n \in \mathbb{N}}\) such that :

\[
\forall x \lim_{n \to +\infty} F^n(a_n x + b_n) = G(x)
\]

where \(G\) is one of :

\[
G_{1\alpha} : x \mapsto \exp(-x^{-\alpha}) \quad \text{Frechet} \\
G_{2\alpha} : x \mapsto \exp(-(-x)^{-\alpha}) \quad \text{Weibull} \\
G_3 : x \mapsto \exp(-\exp(-x)) \quad \text{Gumbell}
\]

where \(\alpha\) is a positive real number named “Tail index”. In most cases \(\alpha\) has to be estimated, which requires a considerable amount of samples to be accurate, but for optimization purpose we will see that this value can be set a priori. Now, if we let \(M(F) = \inf \{x | F(x) < 1\}\) we have the following result[5] :

- If \(M(F) = +\infty\) , \(\lim_{t \to +\infty} [1 - F(tx)]/[1 - F(t)] = x^{-\alpha}\) for \(x > 0\)
- If \(M(F) < +\infty\) , \(\lim_{t \to 0} [1 - F(M(F) - tx)]/[1 - F(M(F) - t)] = (-x)^{\alpha}\) for \(x < 0\)
- As a limiting case , \(\lim_{t \to M(F)} [1 + F(T + g(t)x)]/[1 - F(t)] = e^{-x}\) for \(x \in \mathbb{R}\) and :

\[
g(t) = \int_t^{M(F)} (1 - F(u))du/(1 - F(t))
\]

which gives necessary and sufficient conditions for weak convergence of distributions.
If we now go back to the problem of maximizing a function \( f : \mathbb{R}^n \to \mathbb{R} \) and assuming that \( f \) can be approximated by a definite negative quadratic form in a sufficiently small ball centered at each point of its domain, then one can prove that conditions yielding to a Weibull limiting distribution are fulfilled and that \( \alpha = 2/n \). If the function \( f \) is exactly a definite negative quadratic form:

\[
f : x \mapsto \langle x, Hx \rangle
\]

and if samples are drawn according to a probability measure \( \mu \), then we have:

\[
F : t \mapsto 1 - \mu (^t Q \Lambda^{-1/2} B(0, t^{1/2}))
\]

where \( Q \) and \( \Lambda \) are respectively orthogonal and diagonal matrices arising from the decomposition:

\[
H = ^t Q \Lambda Q
\]

and the notation \( B(0, t) \) stands for the open ball of center 0 and radius \( t \). Now by linearity, we have directly the de Haan condition[1]:

\[
(1 - F(ut))/(1 - F(t)) = u^{n/2}
\]

for \( u > 0 \). Of course, this result can be straightforwardly extended to the case of a non zero maximum \( M(F) \), yielding again to a Weibull limiting distribution for the maximum:

\[
\lim_{N \to +\infty} F^N(M(F) + a_N x) = \exp\{-(-x)^{2/n}\}
\]

with:

\[
a_N = M(F) - \inf\{t | 1 - F(t) \leq N^{-1}\} + o(1)
\]

If the function \( f \) has several equivalent maxima, each of them satisfying the de Haan condition locally, the Weibull distribution with \( 2/n \) as tail index is again the limiting distribution of the maximum.

In the following, we will assume that the previous quadratic model is valid, at least locally so that we may take \( \alpha = 2/n \). Note anyway that after a sufficient number of samples has been drawn, it’s possible to have a sufficiently accurate estimator of \( \alpha \) which may speed up the algorithm. This refinement has not been implemented in our test algorithm but will be in the final version. Several estimators exists for \( \alpha \) and can be found in [6].

3 Prospectiveness

In the following sampling will be done on an hyper-rectangle \( D \) defined by its principal diagonal \((x_1, x_2)\) with \( x_1, x_2 \in \mathbb{R}^n \). Let \((\epsilon_1, \ldots, \epsilon_N)\) be the order statistics associated with sample \((f(x_1), \ldots, f(x_N))\), the points \(x_1, \ldots, x_N\) being drawn from \( D \) using uniform sampling (Markovian sampling may be used too and gives the same limiting distribution). The prospectiveness criterion of
the domain $D$ is an estimation of the probability that the true maximum of $f$ occurs in $D$. It can be computed using the first order statistics by:

$$c(D) = \left(1 - \left(\frac{M - \epsilon_1}{M - \epsilon_k}\right)^{2/n}\right)^k$$

with $k = \min(5, N/10)$ (this value is based on numerical experiments, see [6]) and $M$ the maximum value of $f$ observed so far (this value is updated after each generation of the GA).

A high prospectiveness indicates that the domain must be exploited while a low value shows that the domain is unpromising and must be either dropped or expanded.

## 4 GA and order statistics

### 4.1 Introduction

The general scheme of our GA is quite the same as a regular GA: it first generates an initial population, applies a selection to identify the best individuals and diversify the population by applying operators. The main difference come from the coding, and the fitness evaluation.

This GA works on state domains instead of state points and is able to identify the prospectiveness of a domain by computing order statistics. Then a low fitness will be given to a domains with a low prospectiveness. The power of order statistics enable to evaluate "large" domains with a "few" samples on it and produces a fitness which summarize the property of an entire zone.

It may be noticed that this improvement may be adapted to any GA which is working on continuous state space. It is independent of the selection, the scaling, the sharing etc....

The main adaption have to be done on the coding, the operators and on the fitness evaluation.

### 4.2 Coding

The chromosomes used for our GA are hypercubes of a N-dimensional state space which are encoded with two points on a diagonal (see figure 1).

After the building of a chromosome, only the field $P_1$ and $P_2$ are updated. The other fields will be addressed by the evaluation of the associated fitness and will be used by the operators in order to enhance the exploitation and the exploration. During the evaluation, the domain is sampled and order statistics are computed.

$\epsilon_1$-$\epsilon_5$ are the five first statistics, $P_m$ is the position of the max in the domain and $C$ is the relative confidence of the first statistic.
4.3 Operators

**Crossover** Different crossover operators have been implemented but the one which produce the best results uses the geometrical properties of the parent domains (see figure 2).

Two parent domains $P_1$ and $P_2$ are randomly selected and 2 situations have to be investigated:

1. the two parents share a common sub-domain (left part of figure 2). This shared sub-domain (shaded area) becomes one of the children ($C_1$) and the second one is given by the smaller domain which encompass the two parents (dashed line).

2. the two parents are independents (right part of figure 2). An “inter-sub-domain” is then built (shaded area) to create the first child $C_1$ and, as in the
first case, the second child is given by the smaller domain which encompass
the two parents (dashed line).

Depending on the parents, this crossover enhance both, exploration and ex­
ploration.

**Mutation** For 20% of mutation a full new drawing of the domain is applied
and in the others cases the mutation operator is guided by the prospectiveness
$C$ which is computed during the fitness evaluation. $C \in [0, 1]$, and is maximum
when the confidence about the computed max is maximum. So, when $C$ is close
to “1” one has to enforce the exploitation (centering on $P_m$ and contraction).
On the other side, when $C$ is close to “0”, the exploration has to be enforced
(the operator draws a new domain). Finally when $C$ is not on extrema (0 or
1) a compromise between exploration and exploitation is done (centering and extension).

An example of the three previous situations is given on figure(see figure 3).

![Mutation Operator Diagram](image-url)

**Fig. 3. Mutation operator**

### 4.4 Fitness Evaluation

In order to apply the selection, individuals have to be evaluated with a fit­
ness function. When a chromosome has to be evaluated, its domain is randomly
sampled (30 draws) with the original criterium function (the one we want to op­
timize). The order statistics and the associated confidence are then computed.
The fitness given to the chromosome is then computed with the help of the observed max and the confidence.

5 Test Functions

Different test functions have been used in order to compare our method with a classical GA:

- **Sphere function**: \( f_1(x) = \sum_{i=1}^{N} x_i^2 \), \(-50.0 \leq x_i \leq 50.0\)
- **Step function**: \( f_2(x) = \sum_{i=1}^{N} \left( x_i + 0.5 \right)^2 \), \(-50.0 \leq x_i \leq 50.0\)
- **Ackley function**: \( f_3(x) = -c_1 \cdot \exp \left( S_1(x) \right) - \exp \left( S_2(x) \right) + c_1 + c \) with
  \[
  S_1 = -c_2 \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2} \\
  S_2 = \frac{1}{N} \sum_{i=1}^{N} \cos \left( c_3 x_i \right) \\
  c_1 = 20 \quad c_2 = 0.2 \quad c_3 = 2\pi - 30.0 \leq x_i \leq 30.0
  \]
- **Griewank function**: \( f_4(x) = \frac{1}{400N} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \)
  \(-600.0 \leq x_i \leq 600.0\)
- **Rosenbrook function**: \( f_5(x) = \sum_{i=0}^{N-1} 100 \cdot (x_i^2 - x_{i+1})^2 + (1 - x_i)^2 \)
  \(-30.0 \leq x_i \leq 30.0\)
- **Lennard-Jones function**: This function is coming from the famous Lennard-Jones Molecular Configuration problem: the problem of finding the structure or relative positions of a cluster of atoms that minimizes the potential energy of the structure. The Lennard-Jones problems assume that the potential energy of the molecule is given by the sum of the pairwise interaction between atoms. The position of atoms being given in the three-dimensional space, a problem with \( K \) atoms has \( N = 3K \) real variables to be optimized.
  The potential energy is then given by the following function:
  \[
  f_6(x) = \sum_{i=0}^{N/3} \sum_{j=0}^{i-1} \left[ \frac{1}{d_{ij}^6} - 2 \cdot \frac{1}{d_{ij}^3} \right] \\
  d_{ij} = (x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2 + (x_{i+2} - x_{j+2})^2 \\
  \min f_5(x) = -128.287 \\
  N = 90 \text{(30 atoms)} \\
  -2.0 \leq x_i \leq 2.0
  \]
  All the function have to be minimized and have their minimum at 0 unless the Lennard-Jones function for which only an experimental min is used (best known min=-128.287 for 30 atoms). It must be noticed that both algorithm use the same selection scheme (stochastic remainder without replacement which is not the best) and do not use any scaling or sharing operators. From this point of view both algorithm may be still enhanced.
Our goal being to compare the influence of domain chromosome and order statistics we wanted them to work exactly the same way form the selection point of view.

The number of evaluations being different at each generation for those two algorithms, the number of generation has been adapted in order to maintain the same number of evaluations for all experiments.

It must be noticed that the following curves have been adjusted in order to represent both result on the same graph. Those adjustments have been done on both axis. The “x” axis address the number of evaluations for our GA and must be scaled for the standard GA (x 20). The “y” axis represent the fitness given by both algorithms. The given results given are so different that a logarithm scale has been used to see both curves.

The parameters used for our GA are the following:

<table>
<thead>
<tr>
<th>individuals 100</th>
<th>generations 500</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability of crossover 0.4</td>
<td>probability of mutation 0.3</td>
</tr>
</tbody>
</table>

For the Rosenbrook, Lennard-Jones the number of generation has been extended to 1500 and 2500 respectively. The experiments have been done on a PentiumII 300 MHz and last 7 minutes for N=200 and 14 minutes for N=2000 (N:dimension of the state space). It must be noticed that other experiments has been done for the same functions with the optimum moved in the state space (without symmetries) and the given results are quite the same.

<table>
<thead>
<tr>
<th>Function</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard AG - N=200</td>
<td>10621</td>
<td>11598</td>
<td>11.98</td>
<td>20.11</td>
</tr>
<tr>
<td>Domain AG - N=200</td>
<td>2.28</td>
<td>0</td>
<td>0.32</td>
<td>0.96</td>
</tr>
<tr>
<td>Standard AG - N=2000</td>
<td>9106</td>
<td>8.7105</td>
<td>20.45</td>
<td>165.8</td>
</tr>
<tr>
<td>Domain AG - N=2000</td>
<td>622</td>
<td>222</td>
<td>3.38</td>
<td>1.11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>$f_5$ N=200</th>
<th>$f_6$ N=90 (30 Atoms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard AG</td>
<td>15.6106</td>
<td>-77.21</td>
</tr>
<tr>
<td>Domain AG</td>
<td>254</td>
<td>-125.9</td>
</tr>
</tbody>
</table>

6 Conclusion

This paper has given a good application of the gain which could be given by the mix of different methods. On one side, the main advantage of order statistics for optimization is their ability to summarize the properties of an entire domain with a “small” sample. On the other side, the evolution process of GA is able to build the most adapted chromosome to environment given by the fitness landscape. The mix of both methods really increase the performances of GA by guiding the exploration and exploitation phases. For all tests, the results given by this new GA, are much better than the ones given by a standard GA.
(a) Sphere Dimension 2000

(b) Step Dimension 2000

(c) Ackley Dimension 2000

(d) Griewank Dimension 2000
It must be noticed that this algorithm may be still improved the following way:

- a better selection scheme may be used;
- the order statistics may also be used in order adapted the drawing random law in the domains;
- pools of samples may be used for different domains which could be randomly open on it. Those pools may be updated every $K$ generations.

References

A Factorized Distribution Algorithm Using Single Connected Bayesian Networks

Alberto Ochoa¹,², Heinz Muehlenbein¹, and Marta Soto²

¹ Real World Computing Partnership
Theoretical Foundation GMD Laboratory
GMD–Forschungszentrum Informationstechnik
53754 St. Augustin, Germany
muehlenbein@gmd.de

² Institute of Cybernetics, Mathematics and Physics
Center of Mathematics and Theoretical Physics
ICIMAF, CP 10400, La Habana, Cuba
{ochoa, mrosa}@cidet.icmf.inf.cu

Abstract. Single connected Factorized Distribution Algorithms (FDA-SC) use factorizations of the joint distribution, which are trees, forests or polytrees. At each stage of the evolution they build a polytree from which new points are sampled. We study empirically the relation between the accuracy of the learned model and the quality of the new search points generated. We show that a change of the learned model before sampling might reduce the population size requirements of sampling.

1 Introduction

In ([10],[11]) we have investigated the use of search distributions in evolutionary algorithms. Search distributions generalize the concept of creating new points by genetic operators working on strings. There exist many theoretical advantages for this approach. The only difficulty is to find a good search distribution suited for optimization. We have shown that for Additively Decomposed Functions (ADF) a search distribution can be computed which is factorized into a product of marginal and conditional distributions. The factorization is derived from the structure of the function. For the corresponding algorithm, the Factorized Distribution Algorithm FDA, convergence to the optima could be shown [11, 13].

The computational complexity of the algorithm depends on the number of variables needed to specify the distribution. The factorization derived from the structure of the function might use more variables than necessary. There have been techniques developed in the theory of graphical models which learn a factorization from the data. A number of algorithms use this approach ([12, 14, 17, 6]). They do not restrict the structure of the search distributions, but work with fairly general Bayesian networks. In order to reduce the complexity of the network, the number of parents has to be restricted.
Another possibility to reduce the complexity of the network is to restrict the allowed networks to a subclass of Bayesian networks. Our single connected Factorized Distribution Algorithm (FDA-SC) uses polytree factorizations of the distribution of the best point selected at each generation. We have investigated one member of this class, the so called Polytree Approximation Distribution Algorithm (PADA) in [18]. We have found that this algorithm performs very well on a wide range of functions hard to optimize for a genetic algorithm.

In this paper we continue to explore the usage of polytrees. The outline of the paper is as follows. Section 2 presents the Polytree Approximation Distribution Algorithm (PADA) together with some background material. Section 3 presents our ideas about the transformation of the learned polytree. It introduces a sampling operator which implements one of the possible transformations. Then we present our test functions and discuss the numerical results.

2 The Polytree Approximation Distribution Algorithm

2.1 Polytree Bayesian Networks

Research on Factorized Distribution Algorithms is based on two major fields: Bayesian networks and Evolutionary Computation. A Bayesian network is a directed acyclic graph (DAG) containing nodes, representing the variables, and arcs, representing probabilistic dependencies among nodes.

Bayesian networks can be grouped according to their connection structure. An important subclass are single connected (SCG) graphs: these are graphs where no more than one (undirected) path connects every two variables. Examples of SCGs are chains, trees, forests and polytrees. The last type of graph includes the previous ones. A polytree factorization can be written as:

\[ p(x) = \prod_{i=1}^{n} p(x_i \mid x_{i1}, x_{i2}, \ldots, x_{ir}) \]

where \( x_{ij} \) is the value of the j-th direct parent of variable \( X_i \). In a polytree we require that the parents of any variable are mutually independent. Hence, for all \( i \) we have:

\[ p(x_i) = \prod_{j=1}^{r} p(x_{ij}) \]

where \( p(x_i) \) is the joint distribution of the parent set.

Single connected networks can be recovered from data using simple and efficient learning procedures. The approach used by our algorithm computes a list of dependence/independence weighted assertions. From this list a graph is constructed that satisfies as much as possible the assertions on the list.

2.2 The Algorithm

The Polytree Approximation Distribution Algorithm (PADA) has been designed for the class of single connected Bayesian networks [18]. The rationale behind
the decision of using polytrees for the Factorized Distribution Algorithm class comes from two different sources. First we have been motivated by the impressive performance of the Univariate Marginal Distribution Algorithm (UMDA) [9] even for many non-linear problems. Secondly, from the representation, inference and learning point of view, polytrees have been proved to lie in a region of computational complexity, where the tradeoff between computational cost and goodness of results is promising. For example, in [1] authors compare empirically the performance of both polytrees and general (multiply connected) Bayesian networks models learned from data. In the classification task they explored, they found that polytrees do almost the same job, despite their inability to capture high order independence relationships.

Using polytrees, we gain in efficiency and simplicity in the procedures for sampling and learning the networks. The price we have to pay for using polytrees is a less expressive power. The kind of dependency relations that may be represented, is more restricted for polytrees than for general multiply connected networks.

The Polytree Approximation Distribution Algorithm (PADA) is a FDA optimization algorithm and therefore it shares with the FDA, the general structure described in table 1. In step 2 of the algorithm, PADA uses the LPA algorithm, which is introduced in the next section, to learn a polytree approximation of the probabilistic structure of the selected set of points. Once the algorithm knows the structure, it does parametric learning using Laplace estimation [7].

| step 0: | $t \leftarrow 1$. Generate N points randomly. |
| step 1: | Get a selected set S with M points. $(M < N)$, according to selection method |
| step 2: | Learn $p^s(x,t)$ using the LPA algorithm possibly make a suitable transformation |
| step 3: | Generate N points according to: $p(x,t+1) = p^s(x,t)$ |
| step 4: | $t \leftarrow t + 1$. If the termination criteria are not met, goto to step 1. |

Table 1. PADA.

2.3 The Learning Algorithm LPA

The \textit{LPA} algorithm for learning polytrees from data is a revised version of the algorithm presented in [4] and used in the preliminary version of PADA [18]. It uses a graded dependency relation that measures the degree of dependency between two variables. The basic idea is to preserve the edges representing the strongest dependency relations, but with the restriction that the structure must be singly connected. To do this, the algorithm uses Chow-Liu’s algorithm for building the skeleton (the undirected graph that supports the polytree). Chow-Liu’s algorithm uses a Maximum Weight Spanning Tree (MWST) to obtain the
skeleton of the polytree, where the weight of an edge \( \text{DepG}(\alpha, \beta) \), is a measure of the dependency degree between the linked variables. In our current implementation, we use the Kullback-Leibler cross entropy measure.

\[
\begin{align*}
\text{Dep}(x, y) &= \sum p(x, y) \ln \left( \frac{p(x, y)}{p(x) \cdot p(y)} \right) \\
\text{Dep}(x, y \mid z) &= \sum p(x, y, z) \ln \left( \frac{p(x, y, z) \cdot p(z)}{p(x, y, z) \cdot p(x, z)} \right) \\
\text{DepG}(x, y) &= \min(\text{Dep}(x, y), \text{Dep}(x, y \mid z)), z \in U
\end{align*}
\]

Therefore, we can use marginal \( \text{Dep}(x, y) \), conditional \( \text{Dep}(x, y \mid z) \) and global \( \text{DepG}(x, y) \) dependency measures. Once we have constructed the skeleton a procedure tries to direct the edges of the skeleton by using the following scheme: in a head to head pattern \( \alpha \rightarrow \gamma \leftarrow \beta \), the instantiation of the head to head node \( \gamma \) should normally increase the degree of dependency between the variables \( \alpha \) and \( \beta \), whereas in a non head to head pattern (such as \( \alpha \leftarrow \gamma \rightarrow \beta \), \( \alpha \rightarrow \gamma \leftarrow \beta \) or \( \alpha \leftarrow \gamma \leftarrow \beta \)), the instantiation of the middle node \( \gamma \) should produce the opposite effect, decreasing the degree of dependency between \( \alpha \) and \( \beta \). The idea is to compare the degree of dependency between \( \alpha \) and \( \beta \) after the instantiation of \( \gamma \) with the degree of dependency between \( \alpha \) and \( \beta \) before the instantiation of \( \gamma \).

**Edge Orientation:** For each subgraph \( \alpha \rightarrow \gamma \leftarrow \beta \in \text{skeleton} \), if \( \text{Dep}(\alpha, \beta \mid \gamma) > \text{Dep}(\alpha, \beta) \) then \( \alpha \rightarrow \gamma \leftarrow \beta \)

Edges that are not oriented after the above test, are directed at random without introducing new head to head connections. In table 2 the learning algorithm is shown. In step 4, the constraint on the number of edges guarantees that the skeleton is kept without cycles. The combination of the two measures is intended to give a global dependency measure.

The values \( \epsilon_0 \) and \( \epsilon_1 \) used in steps 1 and 2, are thresholds, which means that edges with values below them are considered independent. In the usual Bayesian networks applications, they are simply small positive numbers. However, in optimization, under certain circumstances, the performance of the algorithm could be sensible to even relative small departures from their true values. Both values depend on the population size. For large population sizes they are almost the same. We can easily calculate these values in an uniform random population.

The general scheme given in table 2 can be easily modified to accommodate different variants of the algorithm. The resulting algorithms have either quadratic or cubic complexity on the number of independence tests. Also, they can output trees or polytrees.

### 3 Transformation of the Learned Graph

The idea of transforming the learned graph before sampling, was suggested in [18]. There an optional pruning step was considered in the original scheme of PADA. Here we present a more general solution. The resulting algorithm we call FDA-SC.
0: Start with an empty graph $G$ and one empty list $L$.

1: for every pair of nodes $\alpha, \beta \in U$
do
   1.1 Compute $\text{Dep}(\alpha, \beta | \emptyset)$.
   1.2 If $\text{Dep}(\alpha, \beta | \emptyset) > \epsilon_0$, insert the edge $(\alpha, \beta)$ in $L$.

2: for every $(\alpha, \beta) \in L, \gamma \in U$ do
   2.1 Compute $\text{Dep}(\alpha, \beta | \gamma)$
   2.2 If $\text{Dep}(\alpha, \beta | \gamma) < \epsilon_1$, remove the edge $(\alpha, \beta)$

3: for every $(\alpha, \beta) \in L, \gamma \in U$
do
   compute $\text{Dep}_G(\alpha, \beta)$
   Rank $L$ in decreasing order of $\text{Dep}_G$.

4: repeat Add to $G$ the next edge in $L$,
   with the largest $\text{Dep}_G$, until a maximum of $n - 1$ edges,
   unless it creates a cycle.

5: for each subgraph $\alpha - \gamma - \beta \in G$ do
   If $\text{Dep}(\alpha, \beta | \gamma) > \text{Dep}(\alpha, \beta | \emptyset)$,
   $\alpha - \gamma - \beta \Rightarrow \alpha \rightarrow \gamma \leftarrow \beta$.

6: Direct the remaining edges without introducing new head to head connections.

Table 2. The learning algorithm $LPA$.

The question is: Once we have learned the corresponding graph of a given selected set of points, can we modify it, in order to get a better graph for the sampling process?

A better graph from the optimization point of view, could mean for example a graph which generates points with a larger response to selection [9], keeping the genotype variance relatively high. Also, a better graph could mean a graph which requires less points to estimate its parameters reliably. In the following we shall assume the last interpretation.

The sample size necessary to estimate correctly the parameters of a Bayesian network, is exponential in the maximum number of parents the network has. Therefore, the sample size is highly sensitive to this number. But it is known that different graphs can perform almost the same when they are used for a sampling distributions. For this reason we propose to use a restricted model class.

4 Reducing the Computational Complexity of Sampling

To the best of our knowledge, all experimental algorithms based on Bayesian networks use the same sampling algorithm. It is a Monte Carlo simulation method introduced in [8]. The method is called Probabilistic Logic Sampling (PLS). Given an ancestral ordering of all the variables (parents go before children), the
method samples $X_i$ using $P(X_i \mid \pi_{X_i})$. One obvious problem of this method, is that it will require an exponential number of points in the number of parents, to estimate the conditional probabilities correctly.

Both in the process of learning and sampling, algorithms that learn general Bayesian networks need a population size, which is exponential in the number of parents. This is important to get reliable estimates of the conditional probabilities. FDA-SC is in a different situation.

FDA-SC has the same computational complexity as other new developed Bayesian optimization algorithms: $O(n^3)$. But it can even be modified to scale only as $O(n^2)$. The computational complexity is determined by the number of dependency tests or computed marginals.

In addition, the computational complexity of FDA-SC is not exponential in the number of parents. FDA-SC uses marginals of at most order 3. This brings us an additional advantage related to reliability: both values $\text{Dep}(\alpha, \beta \mid \gamma)$ and $\text{Dep}(\alpha, \gamma)$ can be computed much more reliably than conditional probabilities involving many variables in the conditioning set. This fact allows us to use smaller data set as the inputs for the learning algorithm. This means less requirements for the population size.

The LPA algorithm used by FDA-SC, deals only with conditional probabilities of first order. However, the created polytree can have nodes with more than two parents. In order to keep the advantage of FDA-SC regarding its bounded complexity in the learning phase, we have to introduce a sampling operator that uses up to second order conditional probabilities. In other words, we should force FDA-SC to use up to third order distributions only.

### 4.1 A New Sampling Operator for Polytrees

The sampling method is also based on PLS. Let us assume, without loss of generality, that $\pi_{X_0} = \{X_1, X_2, \ldots, X_r\}$. Hence, once the parents are instantiated, the PLS method will sample $X_0$ with probability $P(X_0 \mid \pi_{X_0})$. We now transform the subgraph $\{X_0, X_1, X_2, \ldots, X_r\}$, connecting the parents in the following way: $X_2 - X_3 - X_4 - \ldots - X_r$, and then removing all the directions from the graph. The resulting subgraph is called junction tree, and the transformation is similar to moralization [15].

Each triplet $X_0, X_i, X_{i+1}$ is a clique of the subgraph. In junction trees, if we know the values of a clique, we can sample the variables of any clique overlapping with it, conditionally on the overlapping variables. Because, the parents variables are instantiated we use the above schema, to implement a simple simulation rejection method.

```
step 0: $x_0 \leftarrow p(x_0 \mid x_1, x_2)$
step 1: for $i = 3$ to $r$
    $z \leftarrow p(x_i \mid x_0 x_{i-1})$
    if not($z = x_i$) goto step 0
step 2: return $x_0$
```
For many practical (sparse) problems this algorithm will converge immediately. However, the simulation time may rise with the number of parents. It is possible to devise several stopping and estimation criteria, but we will not further elaborate on this.

5 The Test Functions

Now we present the set of additive decomposable functions (ADF), that will be used in our experiments.

1. The deceptive function of order $k$, $F_{\text{dec}K}$, is defined as follows. $u$ denotes the number of 1s in the string.

$$f_{\text{dec}K}\begin{cases} k-1 & \text{for } u=0 \\ \vdots \\ k-i-1 & \text{for } u=i \\ \vdots \\ k & \text{for } u=k \end{cases}$$

Then $F_{\text{dec}K}$ is separable function of subset size $k$,

$$F_{\text{dec}K} = \sum_{i=1}^{l} f_{\text{dec}K}(x_{Ki-K+1}, \ldots, x_{Ki})$$

with $n = k \times l$.

2. The function $F_{\text{IsoChain}}$, is an ADF with a chain-like structure. It is defined as follows:

$$F_{\text{IsoChain}} = \sum_{i=1}^{l-1} I_{\text{so}1}(x_{2i-1}, x_{2i}, x_{2i+1}) + I_{\text{so}2}(x_{2l-1}, x_{2l}, x_{2l+1})$$

with $n = 2 \times l + 1$. The global optimum is $(1, 1, \ldots, 1)$ with value $l(l-1)+1$. This optimum is triggered by $I_{\text{so}2}$. It is very isolated. Six strings with leading zeroes give the second best value of $l(l-1)$. These points are far away in Hamming distance from the optimum.

3. The next test function is like $I_{\text{so}Chain}$, but defined on a grid of size $n = m \times m$. The peak function $I_{\text{so}T2}$ is used at the upper left corner of the torus. Let $u$ denotes the number of 1s in a string.

$$\begin{array}{c|cccc|c} u & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline I_{\text{so}T1} & 0 & 0 & 0 & 0 & m-1 \\ I_{\text{so}T2} & 0 & 0 & 0 & 0 & m^2 \end{array}$$
\[ F_{\text{IsoTorus}} = \text{IsoT}_2(x_{1-m+n}, x_{1-m}, x_1, x_2, x_1+m) + \sum_{i=2}^n \text{IsoT}_1(x_{\text{up}}, x_{\text{left}}, x_i, x_{\text{right}}, x_{\text{down}}) \]

where \( x_{\text{up}}, \text{etc} \), etc, is defined as the appropriate neighbor, wrapping around. This function is difficult to optimize. The best and second best strings have values \( m^3 - m + 1 \) and \( m^3 - m \).

6 Numerical Results

In the experiments we compare the performance of three variants of FDA-SC:

1. using the new sampling method of section 4.1 (\( Msamp \))
2. using the PLS sampling defined by the complete polytree (\( Nsamp \))
3. using PLS sampling defined by polytrees with a bounded number of parents (\( \text{adj} = k \))

FDA-SC uses best-elitism, which means that all the best points (the selected set) are copied to the new generation. In the tables, \( \tau \) is the truncation level and \( G^* \) shows the average generation where the optima are found. \( N \) is the population size and \( n \) the number of variables. The values are computed out of 100 run.

<table>
<thead>
<tr>
<th>( Msamp )</th>
<th>( Nsamp )</th>
<th>( \text{adj} = 2 )</th>
<th>( \text{adj} = 3 )</th>
<th>( \text{adj} = 4 )</th>
<th>( N = 1000 )</th>
<th>( N = 1400 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau )</td>
<td>( G^* )</td>
<td>( % )</td>
<td>( G^* )</td>
<td>( % )</td>
<td>( G^* )</td>
<td>( % )</td>
</tr>
<tr>
<td>0.3</td>
<td>10.43</td>
<td>85</td>
<td>11.08</td>
<td>67</td>
<td>11.26</td>
<td>63</td>
</tr>
<tr>
<td>0.1</td>
<td>5.40</td>
<td>82</td>
<td>5.47</td>
<td>64</td>
<td>6.09</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 3. Numerical results for Deceptive 4, 100 runs.

Table 3 presents the results for the order 4 deceptive function with 32 variables and truncation selection values of 0.1 and 0.3. The population size was set to 800. In both cases the modified sampling (\( Msamp \)) displays better performance than the normal one. Moreover, if we constrain the numbers of neighbors of a node the results are still worse. One interesting fact is that the generation where convergence is reached is almost the same for all cases. The last columns show the results for population sizes where the normal sampling approaches the performance of the modified one. For example, for truncation 0.3, the normal sampling needed 600 more sample points.

Table 4 presents the results for the IsoChain and IsoTorus functions. For these problems we have used a truncation threshold of 0.1. The difference in the performance of the two sampling methods is still more dramatic. Under the conditions of the experiment, the IsoChain function can not be optimized with a polytree (0 success in 100 runs), unless we modify the graph to reduce the number of parents (86 success ! ).

Finally, it is worth noting, that the polytree graphs for all the functions contain nodes with more than three parents, particularly in the firsts generations. The results confirm the conjectures made in section 3.
function |  $m_{\text{sample}}$ | $n_{\text{sample}}$ | $\text{adj} = 3$
--- | --- | --- | ---
$\text{IsoTorus}$ | 64 | 1000 | 9.22 95 9 62 9 47
$\text{IsoChain}$ | 101 | 2000 | 11.2 86 - 0

Table 4. Numerical results for IsoFunctions.

7 Summary and Conclusions

The computationally complexity of genetic algorithms and UMDA is obviously bounded by $O(N^*n)$, where $N$ is the size of the population and $n$ is the size of the problem. Both algorithms use search distribution which can be specified by $O(n)$ parameters. They successfully solve many multi-modal optimization problems. This shows that for many optimization problems a simple search distribution is enough. For additively decomposed functions it is possible to compute a factorized distribution so that FDA will convergence to the optima [13]. But this might be computationally very expensive. In order to find the optimum, a search distribution with less parameters might be enough. But how can we find such a distribution? If the structure of the distribution is learned from the sampled data, then we have at least a computational complexity of $O(n^3)$.

This paper makes a contribution to better understand the tradeoffs of Factorized Distribution Algorithms. We have made a distinction between the quality of a Bayesian network from a statistical point of view, and its quality from an optimization point of view. FDA algorithms need search distributions which guide the search to the optimum, they do not need to precisely estimate empirical distributions. Our algorithm uses single connected Bayesian networks. We have shown that polytrees can be easily modified before sampling. The modified structure performs better than the original polytree, given the same sample size.

References


Abstract. Many bacteria carry gene complexes that code for a toxin-antidote pair, e.g. colicin systems. Such gene complexes can be advantageous for its host by killing competitor bacteria while the antidote protects the host. However, in order to evolve a novel and useful toxin first a proper antidote must be evolved. We present a model of bacteria that can express and evolve such allelopathic systems. Although in the model novel types must evolve from existing types we find that nevertheless in general a high diversity of toxins evolves and, as a side-effect thereof, generalized immunity mechanisms. We interpret the allelopathic systems in terms of an optimization problem: fitness cases are toxins and solutions present (potential) antidotes. As a side-effect of the evolution of allelopathic systems generalized solutions of the optimization task are evolved as well.

1 Introduction

Many bacteria, such as *Escherichia Coli* and related bacteria, carry colicin systems [7, 17]. Colicin systems are gene complexes that code for a toxic protein, i.e. a bacteriocin, and an antidote protein. The bacteriocin kills competitor bacteria, the immunity protein protects the bearer of the colicin system against the toxic protein. Different types of colicin systems exist. A type is defined as a unique combination of a toxic and an antidote pair. In general, different types of colicin systems are cross-sensitive; antidotes are specific for the corresponding toxin.

In natural circumstances colicin systems are often found in bacterium communities [4, 19] in a high diversity. Experimental data [1, 19] show that bacteria are often insensitive to many toxin types; through expression of the corresponding antidote or by more general means, for instance by lacking specific membrane receptors through which the toxin enters the cell. Many bacteria produce at least one toxin, but often bacteria are insensitive to many more toxin types than the number of toxins that they produce.

We have studied a spatial model of the evolution of colicin systems in a bacterium population in which the toxin gene and the antidote gene mutate independently [14]. In the model the number of different types of colicin easily increases, resulting in a high
diversity of colicins. The different colicin types that are present in the population occurs in two modes which we call the individual-based mode and the population-based mode, dependent on the cost per colicin system. In both modes high numbers of colicin types can evolve and be maintained at the level of the population.

The characteristics of the first mode, the individual-based mode which occurs when costs per colicin are relatively low, agree very well with the experimental data that are obtained from natural bacterium populations [1, 19]. All bacteria are insensitive to all toxins that are present in the population, but produces only a very small number of toxins. In this mode the bacteria can be said to evolve general 'immunity’ capabilities, i.e. expression of antidotes against all possible colicin types. Moreover, this general behavior evolves although every individual bacterium sees only a few types of toxin in its lifetime. We have used the term information integration to denote the evolution of general strategies under such sparse fitness evaluation regimes [12].

The population-based mode, occurring when colicins impose high costs on their host, is characterized by a heterogeneous bacterium population, in which many bacterium groups exist that carry different sets of complete colicin systems. In the population-based mode each bacterium carries much less colicin types than the bacteria in the individual-based mode. The heterogeneity of the bacterium population, however, results in an equal number of colicin types being present in the bacterium populations in the individual-based mode and in the population-based mode. In the latter mode the individual bacteria cannot evolve general immunity, due to growth rate limitations. That means that they are sensitive to many of the toxins produced by other bacteria. Rather than maximizing the number of antidotes the individual bacteria optimize their ability to kill other bacteria, i.e. maximize the number of toxins that they produce. The effect is that the different groups of bacteria maintain a standoff based on mutual killing: bacteria of each group are sensitive to toxins that are produced by bacteria of other groups, but they also express toxins to which other bacteria are sensitive.

In the model described above we used a simple genotype-phenotype mapping [14]; a colicin system was defined by two genes which were either active or inactive. New colicin systems arose through single mutations; existing colicin systems decayed, where each of the two genes could decay independently. Actually, the two genes of a colicin system are independent and thus are expected to evolve independently as well. But then, when a novel toxin evolves within a host the host should express the antidote corresponding to that toxin before actually producing it. A mechanism for this process was proposed by Riley [18] and is based on a diversification of immune functions of colicin systems. Once a colicin system has acquired an immune function that extends immunity that is required for the corresponding toxin the latter can change such that the original antidote no longer deactivates the new toxin. Some experimental evidence for this mechanism was presented by Tan & Riley [20].

In this paper we report on a study of the evolution of allelopathy using a more realistic genotype-phenotype mapping (GP-mapping) than in our previous studies. Novel toxins and antidotes must evolve on the basis of existing ones, rather than being created anew in a single mutation event. In conjunction to studying the evolution of novel

1 Here, large scale insensitivity is brought about by colicin systems with ‘defective’ toxin genes, rather than by means such as lacking certain membrane receptors.
allelopathic systems from existing ones using a more realistic GP-mapping we want to investigate if the evolution of allelopathy can be used as a search algorithm to find general solutions of optimization problems. In the individual-based mode described above the bacteria evolved general immunity repertoires. From the point of view of search algorithms we can interpret the immunity capability as a solution to the problem of protection against toxins in the environment.

The GP-mapping that we use in this model is based on a well studied optimization problem, the density classification task for cellular automata [2, 6, 9, 10, 15, 16]. The genes that code for toxins and antidotes are represented by bit strings. To determine whether an antidote gene confers immunity against a given toxin we map the antidote-toxin pair to a cellular automaton lookup table (CA) and an initial condition (IC). We iterate the CA starting with the IC for a fixed number of iterations. The final state of the CA is interpreted as a classification of the IC. Only if the classification is correct, given an a priori defined criterion, does the antidote confer immunity.

Thus, the evolutionary search algorithm in our model is as follows. Bacteria carry a ‘potential solution’ to confer immunity against fitness cases, or toxins, which it expresses itself or which are expressed by neighboring bacteria. Only if the potential solution is correct for a given fitness case is the host of the solution ‘insensitive’ to the toxin. Note that in this model there is not an explicit notion of fitness; bacteria can only ‘use’ the solution to defend themselves against toxins, and they can ‘use’ toxins to kill neighboring bacteria. New solutions are searched for (and found) before they are presented with challenging fitness cases. Search is not only blind it is also unguided.

We find nevertheless that the bacteria in the model evolve cellular automata that show general density classification behavior as a side-effect of being exposed to a diverse toxic environment. The bacteria experience a pressure to maintain immunity against the toxins that are present in their neighborhood, and at the same time explore novel antidotes and novel toxins. The exploration in an already diverse, and hazardous environment apparently is sufficient to evolve general density classifiers.

Whereas in evolutionary search algorithms the population size is generally constant in our model the population size is variable. Thus, the bacteria can die out if they do not form a viable population. In addition, the population in our model is embedded in space and individuals interact only with other individuals which are in their direct neighborhood.

2 The Model

Firstly, we will describe the model at the level of the bacteria and their interactions. After that we will describe the genotype-phenotype mapping, i.e. the density classification task, in more detail.

2.1 A Model of Bacteria and their Allelopathic Interactions

The model that we study here is closely related to the model that we studied in [14]. The bacteria and their interactions are modeled in an individual-based, discrete space, discrete time model, using synchronous updating. The bacteria live in a square grid of
50 by 50 cells, with periodic boundary conditions. Each cell contains a single bacterium or it is empty. Bacteria can colonize empty cells, they produce 'toxins', and they can die. The interactions between bacteria, and the colonization of empty cells by neighboring bacteria are defined in the Moore neighborhood, i.e. a central cell plus the eight cells that are its direct neighbors. Finally, the allelopathic systems carried by the bacteria undergo mutations with rates $\mu_{ca}$ and $\mu_{ic}$ (see below).

The probability that an empty cell is colonized is equal to the sum of the growth rates of the bacteria in its Moore neighborhood. For the simulations that we will discuss below the growth rate of bacteria is fixed at $g = 0.125$. Bacteria die with a rate $d_w = 0.1$. The death rate is increased if a bacterium is sensitive to toxins that are produced by itself or by neighboring bacteria. Per toxin for which a bacterium is sensitive its death rate is increased with $d_t = 0.3$.

The allelopathic systems that the bacteria carry are represented by a single CA, the antidote repertoire, and two ICs, the toxins. A CA can implement immunity to a few specific ICs, or toxins, but also implement a general immunity against many toxins. The latter strategy concurs with a CA that shows a high performance in the density classification task.

We found that we needed at least two ICs per bacterium in order to prevent a premature convergence of all ICs in all bacteria to the same density class. This state leads to an evolutionary dead-lock where all bacteria carry a CA that classifies ICs always as belonging to the same density class, irrespective of its density. In order to maintain diversity of ICs bacteria have to carry at least two ICs of different density classes. We fixed the number of ICs per bacterium to two in order to simplify the analysis of the results. In simulations in which the number of ICs was variable we found qualitatively the same results as we report here.

2.2 The Density Classification Task

The toxicity of ICs and the immunity given by CAs is based on the density classification task. The CAs are 1-dimensional, binary state cellular automata with a neighborhood size 3, the ICs are initial conditions of the cellular automata and are of length 149. Both CAs and ICs are represented as bit strings. The density of an IC is defined as the ratio of 1's to 0's in its bit string.

In the density classification task the CAs must classify ICs on the basis of the density of the IC. If the IC has a density less than 0.5 it belongs to class 0, otherwise it belongs to class 1. The CA is allowed to iterate for maximally 320 time steps, starting with the IC as initial condition. If the CA settles into a homogeneous state of all 0's it classifies the IC as being of class 0. If the CA settles into a homogeneous state of all 1's it classifies the IC as being of class 1. If the CA does not settle into a homogeneous state it does not classify the IC at all.

The performance of a CA is defined as the ratio of correct classifications it makes on a set of 10,000 randomly created ICs [3]. We use the performance as an objective measure of the classification behavior of CAs and to compare CAs of different populations. 'Good' cellular automata typically have performance values of approximately 0.8 (e.g. the GKL rule; 0.81). However, a complete solution, i.e. a cellular automaton with 100% classification accuracy, does not exist [8]. In previous studies of (co-)evolutionary
optimization models that used this optimization problem it appeared difficult to evolve CAs with high performance values [9, 10, 16]. Recently, however, cellular automata have been found with performance values of up to 0.86 [6].

An important property of the density classification task is that most ICs, i.e. ICs with a density around 0.5, are most difficult to classify and easily evolve from one density class to the other one (i.e. by flipping as little as a single bit). In fact, the quality of classification of a 'good' cellular automaton, like for instance the GKL rule, decreases rapidly if it is evaluated on the basis of ICs whose density approaches 0.5 [6, 10].

\[
\begin{array}{|c|c|}
\hline
\text{parameter} & \text{value} \\
\hline
\text{field size } S & 50 \times 50 \\
\text{bacterium growth rate } g & 0.125 \\
\text{bacterium death rate } d_w & 0.1 \\
\text{death rate per toxic IC } d_i & +0.3 \\
\text{CA mutation rate } \mu_{ca} & 0.1 \\
\text{IC mutation rate } \mu_{ic} & 0.2 \\
\hline
\end{array}
\]

Table 1. Parameters and their default values.

The traditional point mutation operator, i.e. flipping a bit at a random position in the string, gives a strong bias towards initial conditions with a density value of 0.5 (see also [10]). Thus, the mutation operator is biased with respect to the property of ICs on the basis of which they are to be classified. We altered the mutation operator in order to make it neutral with respect to the density value of the IC it operates on. This 'density-neutral' mutation randomly increases or decreases the density of the IC by one. When the density is increased a randomly chosen 0-bit is flipped, for a decrease in density a randomly chosen 1-bit is flipped. ICs with densities of 0.0 or 1.0 mutate only if the density increases or decreases, respectively. Only at the beginning of a simulation such ICs with extreme density values exist, therefore, this does not significantly influence the results.

This mutation operator is not neutral with respect to all possible bit strings. Many strings have density values around 0.5; the density neutral mutation operator under samples these bit strings. In the simulations, however, we see that the ICs evolve toward density values near 0.5; in the evolutionary process these ICs are preferentially sampled. We use the density-neutral point mutation in the model with a rate \( \mu_{ca} = 0.1 \) for the CAs and a rate \( \mu_{ic} = 0.2 \) for the ICs. These rates amount to probabilities per bit of approximately 0.0013 for the ICs and 0.0008 for the CAs. In table 1 we have given a table of the parameter and the values that we used in the simulations that are described in the next section.

3 Results

In this paper we foremost want to establish the possibility of evolving novel allelopathic types on the basis of existing ones. Secondly, we want to show that the evolutionary
dynamics that result in this system can lead to the evolution of general strategies. This characteristic of the evolution of allelopathic types can be interpreted, and possibly used, as a search process for general solutions to optimization problems.

We did several runs with the parameter settings given in table 1. Although we did a few additional simulations with different parameter settings we have not performed a rigorous analysis of parameter sensitivity. The results that we report here are, however, also typical for the additional simulations that we studied.

3.1 A Typical Simulation

In this model we have a variable bacterium population size. All bacteria have equal growth and death rates, thus we would expect a stable population size in the absence of allelopathic systems. Because the effectiveness of the toxins and the antidote repertoires in the bacterium population evolve independently the effective bacterium death rate changes over time. Thus, also the bacterium population size is expected to change over time. The fluctuations only indicate changes in the relative effectiveness of the toxins and antidotes, it does not indicate the absolute effectiveness of, say, the antidote repertoires.

![Population size and performance](image)

**Fig. 1.** Evolution of the performance of the best individual, plus the bacterium population dynamics.

In fig. 1 we plot the population dynamics together with the performance of the best CA in the population. The bacterium population starts out small; apparently many bacteria are sensitive to the toxins that are produced in their neighborhood. At $t \approx 700$ the bacterium population size increases sharply. At this point most bacteria carry antidote repertoires that neutralize all toxins that are present in their neighborhood. Around
At $t \approx 4000$ the bacterium population size drops again, indicating an increase in the effective death rate of the bacteria; apparently the bacteria have evolved more effective toxin repertoires. This latter situation remains stable until the simulation ends at $t = 20,000$.

The sharp increase in the population size at $t \approx 700$ is accompanied by a sharp increase in the performance of the best CA (fig. 1). The performance of the best CA at $t = 0$, i.e. $p \approx 0.5$, is typical for CAs that randomly classify IC to class 0 or to class 1, and for CAs that classify all ICs to one and the same density class. Between $t \approx 700$ and $t \approx 2000$ the performance of the best CA is approximately $p \approx 0.65$. This performance is typical for CAs that use block-expanding strategies to classify ICs [3, 5].

**Fig. 2.** Evolution the density distribution of the CAs (top panel), and of the ICs (lower panel). The length of the bit strings of the CAs is 128, the bit strings of ICs have length 149. Both ICs and CAs converge to medium density values.

In fig. 2 we plot the evolution of the distribution of the density values of the CAs (top panel) and the ICs (lower panel). At first CA have density values that are very low or very high. The behavior of CAs at this time, classifying all ICs to one density class, can be accomplished best with a CA that is encoded by a bit string consisting of all 0’s or all 1’s. The transition from low performance to intermediate performance ($t \approx 700$) coincides with a convergence of CA density values to a distribution broadly around 0.5 (i.e. 64 bits). A block-expanding strategy always settles in a homogeneous state of 1’s (0’s) unless there is a sufficiently large block of 0’s (1’s) present in the IC. This block then is expanded to over the whole state of the CA. This strategy requires both 0’s and 1’s in the CA bit string although in uneven numbers.
In the next 5000 time steps the performance of the best CA increases to values around $p \approx 0.75$. CAs with a performance in that range typically use particle-based strategies to classify the density of ICs [3, 5]. Particle-based strategies use mesoscale information processing rules on the basis of (interactions between) regular domains [3]. All of the known CAs with high performance values use particle-based strategies. The transition to particle-based CAs concurs with a strong convergence of the CA density distribution on values around 0.5 (fig. 2). In [11] it was argued that CAs with high performance values will have density values of 0.5. The performance of the best CA in the remaining 15,000 time steps fluctuates between $p \approx 0.75$ and $p \approx 0.80$. The best CA in the simulation occurs at $t = 16,700$ and has a performance fitness of $p \approx 0.79$.

![IC density distribution](image)

**Fig. 3.** Distribution of IC density values for all ICs at $t=5,000$, $t=10,000$, $t=15,000$, and $t=20,000$

In the last period when the bacteria have very general antidote repertoires; the performance of the CAs that they code for is relatively high. The bacterium population size is nevertheless significantly smaller than the carrying capacity; apparently the bacteria carry very effective toxins as well. Indeed, the ICs have density values close to 0.5 (fig. 2). Note that the density distribution of the IC-population shows two peaks, one around 0.45 and one peak around 0.55 (fig. 3). The diversity of the IC density classes is maintained rather than that it collapses to a monomorph population of ICs as in sect. 2.1. It seems that in this state a good general strategy is favorable over a specialized one.

### 4 Discussion

In the previous section we showed that novel allelopathic types can evolve also when they must evolve on the basis of already existing types. Rather then creating a complete
new allelopathic system in a single mutational event, as in earlier studies [14], here the genes that code for the toxins and antidotes evolve on the basis of already existing genes. As in the evolution of natural allelopathic systems immunity to a novel toxin should be evolved before actually evolving and producing the novel toxin. We find that this readily happens; in all simulations antidote repertoires evolve that show medium to high degrees of generality. In 8 out of 30 simulations we find CAs with performance values around 0.8, in the remaining 22 simulations we find CAs with performance values between 0.65 and 0.7.

As an evolutionary search algorithm this model seems inefficient; it requires the evolution of correct solutions prior to the use of fitness cases to evaluate the solutions. Indeed, we see that simulations run for large numbers of generations before good solutions are found (here at $t \approx 5000$). Note, however, that per time step CAs are evaluated on at most 18 ICs. This number even decreases if the local density of bacteria is less than 1. Thus, in terms of number of “fitness evaluations” this model is very efficient (see also [13]).

Also, compared to other (co-)evolutionary optimization models [21] we find CAs with reasonable high performance in a relatively large number of simulations. Also, the diversity is very high despite the low mutation rate and the absence of a diversity promoting selection mechanisms (e.g. sharing or niching). We find that the number of unique CA genotypes makes up 39% of the population, the number of unique IC genotypes makes up 70% of the population. These percentages are much higher than those reported in [15]. Actually, a large number of different good solutions is preserved in the population. For instance, in the simulation described in the previous section at $t=20,000$ 25% of all bacteria have a performance within 10% of the best CA ($p = 0.77$). In these 363 CAs we find 100 unique genotypes.

Other than the selection mechanism we also use a variable population size in our model. Elsewhere, we reported on a study of a spatial coevolutionary optimization model, using the same optimization task [15]. As one of the possible evolutionary outcomes we observed red queen dynamics. These red queen dynamics are characterized by large fluctuations in the individual fitness values, even with periods with fitness values equal to zero. In a model that includes a variable population size this can not occur; the population would simply die out. In the model described here the toxins (i.e the fitness cases, or in terms of coevolutionary optimization; the ‘parasites’) are part of a complex that includes the antidote (i.e. the solution, or ‘host’). This limits the toxin to evolve only so that the corresponding antidote can ‘solve’ it, otherwise it would commit suicide. Our results show that this limitation does not a priori restrict the evolutionary process from finding good solutions.

References


Red queen dynamics occurred when the population was globally mixed, whereas evolution of general solutions occurred when individuals remained localized in space.


Reaction-Diffusion Model of a Honeybee Colony’s Foraging Behaviour

Valery Tereshko

UMIST, Department of Electrical Engineering and Electronics
P.O. Box 88, Sackville Street, Manchester M60 1QD, UK
valery@sound.ee.umist.ac.uk,
http://images.umist.ac.uk/valery

Abstract. We have developed a model of foraging behaviour of a honeybee colony based on reaction-diffusion equations and have studied how communication in the hive determines this behaviour. The model utilizes two dominant components of colony’s foraging behaviour - recruitment to and abandonment of the located food source. The foraging mechanism is based upon competition of employed foragers advertising their nectar sources for pool of unemployed foragers, which is analogous to competition of species for limited resource. For replenished food sources, when an unemployed forager follows all wagle dances (advertisements) to sample the strongest dance encountered, a common “labour market” is formed, which leads to rapid selection of the most profitable nectar source. Otherwise, when unemployed foragers quickly acquire information only about some parts of environment, different niches where individuals are associated mainly with a particular “job sites” are formed, and correspondent foraging trails coexist for a long period. Our results elucidate the role of natural clustering of the dances in the small area of the have - it has to facilitate the information flow that is beneficial for overall process of colony’s food collection.

1 Introduction

Social insect societies can be viewed as a complex system of interacting individuals. These societies perform decision-making without symbolic representation; they exploit the physical constraints of the system as well as communication among individuals. Unlike conventional information processors, these systems are highly flexible and fault tolerant. Despite the fact that each individual follows only a few simple rules, interactions between individuals can lead to the emergence of collective behaviour, which enables the colony to make intelligent decisions in complicated and changing environment. Behaviour of insect societies has proved highly effective in a range of different situations, such as building activities [1], formation of trail networks and foraging patterns [2,3], synchronization of activities [4], sorting (displayed as the formation of regular spatial patterns in the nest) [5], and dynamical division of labour [6].

One of the main activities of social insect societies featuring intelligent decision making in complex and unpredictable environments is foraging behaviour.
A model we developed mimics, in principle, the main features of foraging for different social insect societies. We focus, however, on a honeybee colony. It is known that honeybee colonies choose between different patches of flowers, selectively exploiting the most profitable sources (determined by such variables as sugar concentration, nectar abundance, and distance from the hive [7]). A mathematical model of the foraging behaviour of a honeybee colony was developed based on coupled nonlinear ODEs [3]. The model divided bee activity into four phases, namely unloading nectar from a source, dancing for a source, feeding at a source, and following a dancer. Corresponding probability functions were developed to describe alternation between these phases. The results tallied with experimental observations.

We have developed a minimal model of a honeybee colony’s foraging which includes only two behavioural components but still exhibits collective intelligence. We focus on how communication among individuals determines the foraging dynamics. Two dominant components of a colony’s foraging behaviour we defined are recruitment to and abandonment of the located food source. The recruitment is based upon positive feedback mechanism, i.e. autocatalytic reinforcement of information about located food sources. If this information is uniformly distributed among all potential foragers in the nest, a common “labour market” is formed, which leads to rapid selection of the most profitable source. Otherwise, individuals are mainly locally informed, which leads to the formation of different “labour niches” and, hence, to long-term coexistence of corresponding foraging trails.

2 Model

Consider foraging colony members as a system of interacting elements, involving “employed” and “unemployed” foragers [7]. Employed foragers are in the process of exploiting a food source; they carry information as to the location and profitability of this source. The information is communicated through the celebratory “waggle dance” on an area near the opening of the hive, the dance floor. It is thus that potential foragers are recruited. The profitability of the nectar source is encoded in the number of waggle runs performed by the dancer. Unemployed foragers have either never foraged before, or have abandoned an unrewarding nectar source and are seeking a new food source. Unemployed foragers become employed in one of two ways. Either they search the environment for a previously undiscovered patch of flowers or they remain in the hive, where they await information about nectar sources currently being exploited, and select an employed bee to follow. These bees are termed “scouts” and “onlookers” respectively. Onlookers do not compare dances. Instead, such a bee follows only one dancer, chosen basically at random, before leaving the hive [7]. Since foragers from profitable sources dance for longer, and are also more likely to dance, the profitability of a food source is collectively communicated by proportions of dancers advertising different sources. After locating a food source, scouts may either become employed at that source, or abandon the source and rejoin unem-
ployed foragers. It is thus that we determine the two dominant components of foraging behaviour, namely the recruitment and the abandonment. As will be demonstrated, these two behavioural components govern the foraging dynamics.

Taking Wright’s idea of fitness landscape which assigns a fitness to each point in a genetic space [8], we introduce the profitability landscape which assigns a food quality value \( f \) (as judged by an insect) to each point in informational space \( r \). It is assumed for simplicity that the food sources are permanently replenished.

To formalise the foraging mechanism, we use a chemical reaction analogy. Employed and unemployed foragers are denoted by \( X \) and \( Y \) respectively. An employed forager recruits an unemployed bee to its food source at a rate proportional to that source’s quality:

\[
Y + X \xrightarrow{f} 2X
\]  

(1)

Reaction (1) illustrates the autocatalytic nature of the recruitment process. If an employed forager recruits a nestmate by means of a waggle dance to a food source at which it is employed, the recruit will in turn reinforce the trail and recruit other nestmates, and so forth.

Abandonment of food sources may also occur:

\[
X \rightarrow Y
\]  

(2)

The kinetic equations corresponding to reactions (1–2) take the form

\[
\frac{\partial x(r,t)}{\partial t} = (f(r)y(r,t) - 1)x(r,t) + D_x \frac{\partial^2 x(r,t)}{\partial r^2}
\]

\[
\frac{\partial y(r,t)}{\partial t} = -(f(r)y(r,t) - 1)x(r,t) + D_y \frac{\partial^2 y(r,t)}{\partial r^2},
\]  

(3)

where \( x \) and \( y \) are concentrations, and \( D_x \) and \( D_y \) are diffusion constants of employed and unemployed foragers respectively. For the sake of simplicity, it is assumed that characteristic rates of reactions (1–2) equal to 1.

We consider the solution in a square domain \( Q \). At its boundary \( \partial Q \), no-flux conditions are imposed: \( \frac{\partial x}{\partial k} = 0, \frac{\partial y}{\partial k} = 0 \) where \( k \) is the outward normal to \( Q \) at \( r \in \partial Q \). In accordance with the physical nature of the system, the forager concentrations, their diffusion constants, and profitability landscape values are assumed to be non-negative, i.e. \( D_x, D_y \geq 0 \), and \( x, y \) and \( f \) belong to positive orthant \( \mathbb{R}_+^3 = \{x,y,f \in \mathbb{R}^3; x,y,f \geq 0\} \). Let us suppose for simplicity that all foragers are initially distributed uniformly in space: \( x(t_0, r) = x_0, y(t_0, r) = y_0 \).

This model provides a helpful analogy for information flows from employed foragers advertising food sources to onlookers. We investigate how these flows affect the system dynamics as a whole.

---

1 The system dynamics is not changing qualitatively when the abandonment rate is inversely proportional to the quality of the food source at a given point.
3 Results

The employed foragers are assumed to diffuse very slowly, which corresponds to localization of information about every point of the explored environment space. The diffusion of unemployed foragers determines their access to this information.

In reality, most waggle dances occur on a small area, the dance floor, near the entrance to the hive and onlookers are able to obtain information about different nectar sources. To study how colony’s intelligent choice emerges through communication between employed and unemployed foragers, we compare three cases: (i) locally informed onlookers, individually possessing information about one local point of environment space without knowing of any other, (ii) globally informed onlookers possessing information on the entire environment space, and (iii) an intermediate case, when onlookers quickly acquire information about some part of the environment space and slowly become informed about others.

3.1 Theoretical analysis

According to our model, the first case involves a situation when the diffusion of onlookers in the informational space is so slow that they possess only local knowledge of the environment. In the limit of vanishing diffusions, system (3) possesses an integral of motion

$$\frac{\partial x(r,t)}{\partial t} + \frac{\partial y(r,t)}{\partial t} = 0,$$

which yields a condition of constant local concentrations

$$x(r,t) + y(r,t) = x_0 + y_0 = C_0$$

This allows elimination of the variable $y$ from system (3) which, in that case, reduces to a spatially extended logistic equation

$$\frac{\partial x(r,t)}{\partial t} = (a(r) - f(r)x(r,t))x(r,t),$$

where $a(r) = C_0 f(r) - 1$.

The profitability threshold, $1/C_0$, separates the patches of flowers which are attractive for foraging from unrewarding sources. Bees begin to concentrate only on those patches which amount of nectar exceeds the threshold:

$$f(r) > 1/C_0$$

The nature of the problem allows us to take into account a countable set of $n$ spatial modes which correspond to the local maxima of the profitability landscape. Considering only these modes, the infinite-dimensional system (8) reduces to a system of uncoupled equations describing logistic growth of employed foragers at a particular spatial point

$$\dot{X}_i(t) = (\alpha_i - f_i X_i(t))X_i(t),$$
where \( \alpha_i = C_0 f_i - 1 \) is the reproductive rate of \( i \)-th mode.

Every mode associated with a profitability value exceeding the profitability threshold converges to the attractor

\[
x_i^s = C_0 - 1/f_i, \quad i = 1, \ldots, n.
\]

(9)

When onlookers are informed at a local level only, colony’s decisions are thus also made locally. In other words, the choice is not a collective but individualistic one - trails to every profitable nectar source, i.e. source which food value satisfies condition (9), coexist, and the colony fails to select the most profitable source.

Let us consider now the opposite case when onlookers have access to all information available on the explored environment. This can be modelled by diffusional mixing of onlookers in the informational space. The faster the diffusion, the more equal the acquisition of the information by each individual in the nest. In the limit of onlooker’s full mixing (uniform distribution of the input information) system (3) reduces to the system of integro-differential equations:

\[
\frac{\partial x(r, t)}{\partial t} = (f(r)y(t) - 1)x(r, t) + D_x \frac{\partial^2 x(r, t)}{\partial r^2}
\]

\[
\frac{\partial y(t)}{\partial t} = -y(t) \frac{1}{S} \int_Q f(r)x(r, t) \, dr + \frac{1}{S} \int_Q x(r, t) \, dr
\]

(10)

where \( y(t) = (1/S) \int_Q y(r, t) \, dr \) is the spatially-averaged concentration of onlookers over domain \( Q \) with area \( S = \int_Q \, dr \).

The integral of motion\(^2\)

\[
\frac{1}{S} \int_Q \frac{\partial x(r, t)}{\partial t} \, dr + \frac{\partial y(t)}{\partial t} = 0
\]

(11)

yields the condition of constant total concentration

\[
\frac{1}{S} \int_Q x(r, t) \, dr + y(t) = x_0 + y_0 = C_0
\]

(12)

that allows us to eliminate variable \( y \) from system (12) which, in that case, reduces to a spatially extended Lotka-Volterra system with a special connection matrix

\[
\frac{\partial x(r, t)}{\partial t} = [\alpha(r) - f(r) \frac{1}{S} \int_Q x(r, t) \, dr]x(r, t) + D_x \frac{\partial^2 x(r, t)}{\partial r^2}
\]

(13)

where \( \alpha(r) = C_0 f(r) - 1 \).

In the limit of vanishing \( D_x \) if only modes corresponding to local maxima of the profitability landscape are taken into account, the infinite-dimensional

\(^2\) Integration over the space eliminates the diffusional term in the first equation of system (12) due to the boundary conditions.
system (15) reduces to the system of coupled equations for the spatial mode amplitudes

\[ \dot{x}_i(t) = (\alpha_i - f_i \sum_{i=1}^{n} x_i(t)) x_i(t) \]  

(14)

Dividing \( i \)-th and \( j \)-th equations on \( \alpha_i f_i \) and \( \alpha_j f_j \) respectively and subtracting one equation from the another, one obtains

\[ \frac{1}{f_i} \dot{x}_i(t) - \frac{1}{f_j} \dot{x}_j(t) = \frac{1}{f_j} - \frac{1}{f_i} \]  

(15)

The integration of equation (17) results in

\[ \frac{x_i(t)}{x_j^\eta(t)} = \frac{x_i(0)}{x_j^\eta(0)} \exp\{ (\eta - 1)t \} \]  

(16)

where \( \eta = f_i / f_j \).

Expression (18) provides an analytical proof of selection in the system. If the \( m \)-th mode is the fittest, i.e. corresponding to the maximum of the food landscape, then \( \eta = f_m / f_j > 1 \) for \( \forall j \neq m, j = 1, \ldots, n \). Hence, it immediately follows that when \( t \to \infty, x_m / x_j^\eta \to \infty \) for \( \forall j \neq m, j = 1, \ldots, n \). However, a condition of constant total concentration (14) and positive definiteness of variables prevents unlimited growth of modes. This means that the amplitudes of all modes excluding the fittest should tend to zero as time increases.

If the food value of at least one mode exceeds the replication threshold, then the trivial equilibrium

\[ x_i^s = 0, i = 1, \ldots, n \]  

(17)

loses stability and system (16) converges to a non-trivial attractor

\[ x_m^s = C_0 - 1/f_m, x_i^s = 0, i = 1, \ldots, n; i \neq m \]  

(18)

where \( f_m > f_i \), which corresponds to selection of the most profitable food source.

A honeybee colony where onlookers can acquire global information about advertised sources exhibits collective intelligence, i.e. it is able to select the most profitable food source in the explored environment.

### 3.2 Numerical analysis

Let us consider the outcome when the information about an explored environment is not uniformly distributed among the onlookers. This case can be modelled by intermediate values of onlooker’s diffusion in the informational space. To model this situation, consider a profitability landscape with three separate niches (Fig. 1). The diffusional length \( l_{d(k)} \) of onlookers in the \( k \)-th niche can be evaluated as [9]

\[ l_{d(k)} = \sqrt{D_y \tau_{d(k)}} \]  

(19)

where \( \tau_{d(k)} \) is the characteristic time of diffusion in the \( k \)-th niche.
Everywhere outside the niches, the reproductive rate is kept equal to the profitability threshold, so that neither attract nor repel foragers. The characteristic diffusion times $\tau_{d(k)}$ can be approximated then as

$$\tau_{d(k)} \sim \frac{1}{\alpha_m(k)} = \left(\frac{C_0 f_{m(k)}}{C_0 f_{m(k)} - 1}\right)^{-1}$$

(20)

where $\alpha_m(k)$ and $f_{m(k)}$ are, respectively, the maximal reproductive rate and the maximal profitability value in the $k$-th niche.

Fig. 1. Food landscape.

Fig. 2. Concentration of employed foragers at the following parameter values: $x_0 = 0.01$, $y_0 = 1$, $D_x = 0.001$, $D_y = 1$, and (a) $t = 102$, (b) $t = 150$, (c) $t = 300$, (d) $t = 367$. 
If $D_y = 1$, the length of the information spreading among onlookers is comparable to the niche size. This means that there are three informational clusters of onlookers. Each cluster is well-informed about its "own" nectar patch, and there is slow diffusional exchange of information between the clusters. To observe the dynamics, we performed computer simulations. We use the explicit method of numerical integration of PDEs when the space and time are divided into discrete uniform sub-intervals, and derivatives are replaced by their finite-difference approximations. The numerical integrations are performed on a 2D lattice with the space step $\Delta r = 0.25$. The time steps are chosen to guarantee the stability and convergence of the explicit method. In the first and second niches, there are two peaks of value 2.25 and 2.6, and 1.9 and 2.25 respectively, and in the third niche, there is one peak of value 1.9 (Fig. 1). Throughout, the initial concentration of employed foragers $x_0 = 0.01$, and their diffusion $D_x = 0.001$ respectively. Initially, the concentration of employed foragers begins to grow proportionally to the profitability distribution. At $t \approx 100$ then, foragers abandon the less profitable nectar source in the second niche and switch over completely to the more profitable neighbouring source. For this reason, the concentration of foragers at this source becomes even higher than at the most profitable source in the first niche. Concentration of bees at the third niches is markedly reduced (Fig. 2a). At $t \approx 150$, foragers abandon the third niche (Fig. 2b). At $t \approx 300$, all foragers in the first niche switch over to the more profitable nectar source. The concentration of bees at the second niche becomes very low (Fig. 2c). At $t \approx 370$, all bees select the most profitable food source only (Fig. 2d).

The faster the information diffuses, the more rapidly every onlooker acquires all information available on the "market". As a result, at the given profitability landscape, the same scenario to the above develops but in a shorter timescale. Fig. 3 illustrates selection of the most profitable nectar source at two higher constants of $D_y$. One can see that the colony needs less time to select the "best" source and to abandon all others when spreading the information among onlookers is more rapid.
4 Conclusions

We have found that two dominant behavioral components of a honeybee colony govern its foraging dynamics, recruitment (when an unemployed bee follows one of the recruiters and becomes employed at a food source) and abandonment (when a forager returns to the pool of unemployed bees). When unemployed foragers are recruited, they then become recruiters themselves for a particular source – the information is able to reproduce itself. Its reproduction rate is proportional to the quality of the food source. This mechanism is merely a positive feedback, or autocatalytic reinforcement of useful information [10]. The dissipation of information occurs when foragers abandon unrewarding food sources. This ensures the mechanism of changing in the system and prevents its sticking in local optima. The carriers of information, employed foragers, exploit an available limited “resource”, the pool of unemployed bees. The natural physical limitation of the hive, i.e. the condition for mass conservation, restricts the number of information carriers. The replication of information is thus naturally limited. All these factors lead to the competition of information and, as a result, only the most useful information, trail to the most profitable food source, survives in the system.

We have analyzed how communication between employed and unemployed foragers determine the colony’s behaviour. When the onlookers acquire information about the entire environment, a common “labour niche” is formed, which leads to a rapid selection of the most profitable nectar source. Otherwise, different niches where individuals are associated mainly with a particular “job sites” are formed, and correspondent foraging trails coexist for a long period. The model elucidates thus the role of the dance floor as an information-sharing center and the dependence of dynamics on the information-mapping patterns.

From the complex system theory point of view, the model shows how complex behaviour emerges from the interactions of individuals possessing a very simple behavioural repertoire.

References

A Religion-Based Spatial Model for Evolutionary Algorithms

René Thomsen, Peter Rickers and Thiemo Krink

EvAlife Group, Dept. of Computer science, University of Aarhus, Denmark
{thomsen,rickers,krink}@daimi.au.dk, http://www.evalife.dk

Abstract. Traditionally, the population in Evolutionary Algorithms (EA) is modelled as a simple set of individuals. In recent years, several models have been proposed, which demonstrated that a structured population can improve the performance of EAs. We suggest that religions should be considered as an interesting new source of inspiration in this context, because of their important role in the organisation of human societies. Essential concepts of religions are commandments and religious customs, which influence the behaviour of the individuals. Typically, religious rules include the commitments to reproduce, to believe in no other religions, and to convert non-believers. In this paper, we used these concepts and the struggle between religions as an inspiration for a novel Religion-Based EA (RBEA) with a structured population. In the RBEA, individuals inhabit a two dimensional world in which they can move around and interact with each other. Additional ideas are the religion membership of the individuals, which restricts mating between individuals of different religions and the exchange of individuals between religions by conversion. The result of this design is a simple model with spatial diffusion of genes and self-organised subpopulations with varying population sizes. Our experiments on six numerical benchmark problems showed that the performance of the RBEA was clearly better than the performance of a standard EA and a Diffusion EA.

1 Introduction

The traditional model of the population in Evolutionary Algorithms (EA) is a simple non-ordered set of individuals. Regarding mate selection, any individual can be recombined with any other individual of the population. The main consequence of this design is that the gene-flow inside the population is much higher compared to a real world situation, which often leads to premature genetic convergence. In recent years, several studies tackled this problem by the design of models with a structured population, such as the Cellular GA [9], the island model [1, C6.3], tagging [1, C6.2], multinational evolutionary algorithms [8], patchwork models [4], [6], and terrain-based GAs [2]. Most of these models are based on analogies to real world phenomena. In this paper we took inspiration from religions, regarding their emergent effects on populations.

Religious rules typically include the commitments to reproduce, to believe in no other religions, and to convert non-believers, which provide the means
for a propagation of the religion. Further, meetings and rituals, such as divine
service, christening, and confirmation in the Christian religion, help to reinforce
these religious rules. New members are acquired by reproduction and religious
education or by conversion. Not surprisingly, there is strong competition among
different religions. This competition often leads to aggression and warfare such as
the epic conflict between Christianity and the Islam in Europe and the clash be­
tween Muslims and Hindus on the Indian sub-continent. Apart from the interest
in human culture and religion, religious concepts and the history of their impact
on human civilisations can serve as a great source of inspiration for population
models. This motivated us to design the multi-agent Jihad System\(^1\) [7], which
models the interactions between artificial religions and individuals within reli­
gions. In this paper, we introduce a hybrid system consisting of a very simplified
version of our original Jihad System and an Evolutionary Algorithm, which we
call the RBEA (Religion-Based EA\(^2\)). This new approach partly resembles ideas
found in the Cellular GA (mating between individuals of neighboured cells),
patchwork models (individuals as mobile multi-agents), and tagging / island
models (semi-separation of subpopulations; here: religions). The two main inspi­
rations from real world religions in our RBEA model are (i) the competition for
new individuals by conversion and (ii) the restriction of matings to individuals
that belong to the same religion.

The paper is organised as follows. The next section introduces the key fea­
tures of the RBEA. In section 3 we present some experiments, in which we
compare the performance of our approach with a standard EA and a Diffusion
EA (also known as the Cellular GA). The Diffusion EA is a spatial model, where
the individuals are placed in a two dimensional grid lattice. Mating and selection
is restricted to immediate neighbours [1, C6.4]. In the final section, we discuss
our results and outline some future extensions and applications.

2 The RBEA model

The RBEA model is a hybrid evolutionary algorithm which consists of three
components: a world, religions, and a population of individuals.

The world is represented as a two dimensional grid. The structure of the grid
is a 9-cell neighbourhood\(^3\), which means that each cell in the grid is surrounded
by 8 neighbours. The edges of the grid are wrapped, so that the grid acts as a
torus, thereby avoiding artifacts created by the borders. The size of the world
is determined by the world dimension parameter (e.g. a value of 50 indicates a
world represented as a 50×50 grid). Furthermore, a cell can contain either one
or no individual.

The religions define the subpopulations in which the individuals are grouped
i.e. each individual belongs to a religion. The number of religions specifies the
initial number of religions in the world. Initially, each religion has the same

\(^1\) We use the name Jihad as a pseudonym for the struggle between religions.
\(^2\) Formerly known as the Jihad EA.
\(^3\) Also called a Moore Neighbourhood.
number of individuals as believers. Further, the number of believers in a reli­
gion cannot decrease below the minimal believers threshold. Finally, the convert threshold influences the success of a conversion action - described below.

The population size defines the initial number of individuals in the world, which is constant. Each individual consist of the attributes religion, location, and genome.

The religion attribute defines to which religion the individual belongs and can only be changed if the individual is converted to another religion. The individual’s position in the multi-agent world is determined by its location attribute, whereas the genome defines its position in the search space.

procedure RBEA
begin
 initialise
 evaluate
 while (not termination-condition) do
 begin
 for each individual i do
 begin
 random walk(i)
 convert(i)
 mate(i)
 end
 mutate
 evaluate
 end
end

Fig. 1. The structure of the RBEA model.

The RBEA model works as follows (see fig. 1): First, all individuals are initialised and evaluated according to the objective function. Afterwards, the following process will be executed as long as the termination condition (e.g. current generation > max generation) is not fulfilled: Each individual performs the actions random walk, convert, and mate.

The random walk action moves the individual to a neighbour cell, provided that it is vacant.

Further, individuals can change the religion of other believers by using the convert action, which works as follows (fig. 2): If individual a locates another individual b, which belongs to a different religion and the amount of believers of individual b is above minimal believers then individual a tries to convert b. The conversion succeeds if two conditions are fulfilled: (i) individual a has a relative fitness greater than b and (ii) a random number (between 0.0 and 1.0) times the relative fitness is greater than the convert threshold. The relative fitness of
an individual is defined as a value between 0 and 1, where 0 corresponds to the worst and 1 to the best fitness recorded in the current generation.

**procedure** Convert(individual)
**begin**
scan neighbourhood for individuals from other religions
if other individual exist **then**
if number of believers of other individual’s religion > minimal believers **then**
if relative fitness > other individual’s relative fitness **then**
if (random number * relative fitness > convert threshold) **then**
change other individual’s religion to this individual’s religion
**end**

**Fig. 2.** The structure of the convert action.

The execution of the *mate* action creates an offspring with an individual from the neighbourhood. Usually, the mate choice is restricted such that only individuals from the same religion can mate. However, if an individual belongs to a small religion with a size equal to the *minimal believers* threshold then it can try to mate with individuals of other religions in its neighbourhood. If the offspring is fitter than one or both of its parents then it will substitute the parent with the lowest fitness. The structure of the mating action is shown in figure 3.

**procedure** Mate(individual)
**begin**
if number of believers of the individual’s religion > minimal believers **then**
pick individual of the same religion from the neighbourhood
create offspring with the same religion as the mating individuals.
if fitness of offspring is better than fitness of worse parent **then**
replace worse parent with offspring
else (i.e. number of believers is equal to minimal believers)
pick individual of any religion from the neighbourhood
create offspring with the same religion as the initiator of the mating.
if fitness of offspring is better than current individual’s fitness **then**
replace current individual with offspring
**end**

**Fig. 3.** The structure of the mate action.

The *mate* action is executed after the *random walk* and *convert* action to ensure that the offspring cannot replace its parent before the parent gets the opportunity to convert other individuals. Finally, some of the individuals are mutated and their fitness scores are re-evaluated.
3 Experiments

3.1 Experimental Setup and Data Sampling

We used the following parameters in the RB-, standard-, and Diffusion EA:

- population size = 400
- mutation operator : Gaussian mutation with variable variance\(^4\) \(\sigma^2(t) = \frac{1}{1+t}\)
- mutation probability\(^5\) = 0.75
- crossover operator : arithmetic crossover
- crossover probability\(^6\) = 0.90
- genome representation : floating point vectors

Further, in all algorithms we used elitism. In case of the RBEA elitism was applied within the religions.

In the RBEA, we used the following additional parameter settings:

- world dimension = 25
- number of religions = 4
- conversion threshold = 0.98
- minimal believers = 40

During the initialisation of the RBEA, all individuals were randomly distributed in the grid world. In the standard EA we used tournament selection with a tournament size of two. Finally, in the Diffusion EA we used a 20x20 grid with a wrap-around topology which corresponds to 400 individuals. Each cell had 9-cell neighbourhood like in the RBEA. In order to compare our model with the standard EA and the Diffusion EA we calculated the average fitnesses and standard error of the best individuals in 30 runs.

3.2 Test Functions

In our experiments we used the following six test functions:

Schaffer F6: \(f(x, y) = 0.5 + \frac{\sin^2(\sqrt{x^2 + y^2}) - 0.5}{(1+0.001(x^2+y^2))^2}\) where \(-100 \leq x, y \leq 100\)

Rastrigin F1 (20d): \(f(\bar{x}) = 200 + \sum_{i=1}^{20} x_i^2 - 10 \cdot \cos(2\pi x_i)\) where \(-5.12 \leq x_i \leq 5.12\)

Michalewicz's function: \(f(x) = \sum_{i=1}^{10} \sin(x_i) \cdot \sin(\frac{20(i-1)x_i}{\pi})\) where \(0 \leq x_i \leq \pi\)

Griewank's function: \(f(\bar{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{10} (x_i - 100)^2 - \prod_{i=1}^{10} \cos(\frac{x_i - 100}{\sqrt{i}}) + 1\) where \(-600 \leq x_i \leq 600\)

De Jong F2: \(f(x, y) = 100(x^2 - y)^2 + (1 - x)^2\) where \(-2.048 \leq x, y \leq 2.048\)

De Jong F4: \(f(\bar{x}) = \sum_{i=1}^{30} x_i^4\) where \(-1.28 \leq x_i \leq 1.28\)

\(^4\) \(t\) is the current generation.

\(^5\) Note that the applied mutation is mainly determined by the variance of the Gaussian mutation operator (the genome is real encoded). The mutation probability of 0.75 reduces the effect of the mutation operator.

\(^6\) We did not use \(p_c\) because the crossover operation was part of the mate action.
Except for the Michalewicz function, all test functions are minimisation tasks. The Michalewicz test function has its global optimum at 9.66; all other functions have their global optimum at 0.

3.3 Results

Figure 4 shows an example of the World at three different time steps.

![Fig. 4. Screenshots of different world states. Gray-tones correspond to different religions.](image)

In all experiments, we observed characteristic differences in the performance of the standard EA, Diffusion EA and RBEA (see table 1).

<table>
<thead>
<tr>
<th>Test function</th>
<th>Std. EA Mean ± SE</th>
<th>Diffusion EA Mean ± SE</th>
<th>RBEA Mean ± SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schaffer F6</td>
<td>6.134E-3±1.214E-4</td>
<td>1.447E-4±4.904E-6</td>
<td>0.0±0.0</td>
</tr>
<tr>
<td>Rastrigin F1 (20D)</td>
<td>3.631±2.797E-2</td>
<td>0.7783±1.694E-2</td>
<td>3.758E-2±1.274E-3</td>
</tr>
<tr>
<td>Griewank (10D)</td>
<td>1.887±5.794E-2</td>
<td>5.279E-2±2.221E-3</td>
<td>1.423E-3±4.824E-5</td>
</tr>
<tr>
<td>De Jong F2</td>
<td>6.492E-3±1.435E-4</td>
<td>1.214E-7±4.112E-9</td>
<td>2.344E-5±2.917E-7</td>
</tr>
<tr>
<td>De Jong F4</td>
<td>4.739E-3±1.076E-4</td>
<td>5.683E-6±1.889E-7</td>
<td>4.569E-7±1.423E-8</td>
</tr>
</tbody>
</table>

Except for the De Jong F2-function, the RBEA found much better solutions after 800 generations with a higher convergence speed than the Diffusion EA. The performance of the standard EA was always much worse compared to the other two algorithms. Figures 5(a) to 5(f) show the average fitness curves of the best individuals in the standard EA, Diffusion EA and RBEA for the six test functions, which illustrate these characteristics. The RBEA turned out to
be 3% faster (10 seconds after 30 runs on a Pentium II-466MHz PC) than the Diffusion EA and 49% slower than the standard EA in terms of computation time. Regarding the population dynamics, we observed that religions frequently exchanged individuals by *convert* actions. Mating among individuals of different religions turned out to keep the population dynamics lively.

Fig. 5. Average best fitness (30 runs).
Interestingly, religions that temporarily had very few members were able to regain several individuals and sometimes took the lead in the number of believers. Figure 6 shows an example of the population dynamics during the execution of the RBEA with the De Jong F2 test function.

![Figure 6. Religion sizes during a run of the De Jong F2 function.](image)

### 4 Discussion

The results of our experiments with the RBEA show that a population structure model based on religious concepts can clearly improve the performance of evolutionary algorithms. Except for the De Jong F2 function, our approach yielded much better results regarding the mean fitness of the best individual and the convergence speed than the Diffusion EA. In case of the Schaffer F6 test function, our RBEA found the exact global optimum in all 30 test runs compared to a much worse performance of the standard EA and the Diffusion EA.

Why did our RBEA perform so well compared to the other two algorithms? First, the genetic information spread slowly due to the spatial topology of the population model, which restricted matings to neighboured individuals like in the Diffusion EA. Second, the division of the population into religious subpopulations like in island models lead to additional separation. However, in contrast with island models, our model provides flexible subpopulation sizes, a self-adaptive control of the migration, and a spatial neighbourhood between subpopulations. Another important aspect was that religions could regain individuals, when their number of members was equal to the number of minimal believers. This shifting balance of control between religions improved the performance significantly compared to experiments with an initial model without this feature.

So far, we only tested our RBEA with one arbitrary selection of parameter values. Further systematic analysis, such as an investigation into the effects of the world dimension in relation to the population size or the role of the number of religions will probably improve the algorithm even further. In addition, it would be interesting to analyse the performance characteristics of the RBEA
search process more systematically regarding the known landscapes of the six benchmark test problems.

Moreover, there are several possible extensions to this approach. One extension could be a combination with a mass-extinction model [3], [5] or a more effective mutation operator [5]. The effect of these extensions would be an additional exploration of the search space, which could improve the performance results for certain problems, such as the De Jong F2 function. Another idea would be to interpret the spatial dimensions of the arena as the parameter space of two EA variables such as overall mutation rate and number of crossover points like in TBGAs [2] and terrain-based Patchwork models [6].

Regarding future applications, we would expect that the RBEA will turn out to be valuable in context of dynamic optimisation problems due to its ability to reorganise its population structure by self-adaptation.

5 Acknowledgements

The authors would like to thank Zbigniew Michalewicz and Rasmus K. Ursem for useful comments and suggestions.

References

5. Krink, T., Rickers, P., and Thomsen, R. Applying self-organised criticality to evolutionary algorithms. In PPSN VI - Sixth International Conference on Parallel Problem Solving from Nature (in subm.).

Bayesian Evolutionary Optimization Using Helmholtz Machines

Byoung-Tak Zhang and Soo-Yong Shin

Artificial Intelligence Lab. (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
{btzhang, syshin}@scai.snu.ac.kr
http://scai.snu.ac.kr/

Abstract. Recently, several evolutionary algorithms have been proposed that build and use an explicit distribution model of the population to perform optimization. One of the main issues in this class of algorithms is how to estimate the distribution of selected samples. In this paper, we present a Bayesian evolutionary algorithm (BEA) that learns the sample distribution by a probabilistic graphical model known as Helmholtz machines. Due to the generative nature and availability of the wake-sleep learning algorithm, the Helmholtz machines provide an effective tool for modeling and sampling from the distribution of selected individuals. The proposed method has been applied to a suite of GA-deceptive functions. Experimental results show that the BEA with the Helmholtz machine outperforms the simple genetic algorithm.

1 Introduction

Evolutionary algorithms are typically used to solve function optimization problems, i.e. to find the point $x^*$ that maximizes the objective function:

$$x^* = \arg \max_x \{f(x)\}. \quad (1)$$

Conventional evolutionary algorithms solve this problem by iteratively generating populations $X^t$ of search points $x^t$ until $x^*$ is found or the best solution $x^t_{best}$ at generation $t$ is acceptable [3]. This class of algorithms does not use an explicit model of the sample population; They just generate new points based on old points. Therefore, it is difficult to capture the structure of the objective function.

An alternative way is to model the population explicitly using a probability density function. Since many optimization problems have underlying structure in their search space, using this structure can help the search for the optimal solution. Recently, a number of methods have been proposed that explicitly model the population of good solutions and use the constructed model to guide further search [2, 5, 9, 11]. These methods are generally known as the estimation
of distribution algorithms or EDAs [8]. They use global information contained in the population, instead of using local information through crossover or mutation of individuals. From the population, statistics of the hidden structure are derived and used when generating new individuals.

One of the main issues in distribution-based optimization is how to build and sample from the distribution of the population. Several methods have been proposed, including the methods based on dependency chains [5], dependency trees [2], factorization [9], neural trees [13], Bayesian networks [8,12], and genetic programs [14].

In this paper, we present a method that estimates the sample distribution using a graphical learning model known as Helmholtz machines. The method is implemented as a Bayesian evolutionary algorithm (BEA), a probabilistic model of evolutionary computation [13]. The Helmholtz machine is a multi-layer network [4] and can find the hidden structure in a data set. Even if the structure has complex relationship, the Helmholtz machine can model the data dependency by a hierarchical network. The wake-sleep algorithm [6] provides a learning mechanism for capturing the underlying structure of the data. In addition, since the Helmholtz machine is a generative model, generation of new samples from the model is efficient. Our experimental evidence supports that Helmholtz machines are effective for estimation and simulation of the population distribution.

The paper is organized as follows. In Section 2, previous work is briefly reviewed. Section 3 presents the Bayesian evolutionary algorithm using the Helmholtz machine. We also describe the architecture and learning algorithm of the Helmholtz machine. Section 4 reports the experimental results. Conclusions are drawn in Section 5.

2 Optimization by Distribution Estimation

A simplest way for distribution estimation is to consider each variable in a problem independently and generate new solutions by only preserving the proportions of the values of all variables independently of the remaining solutions. Population-based incremental learning (PBIL) uses a single probability vector to replace the population [1]. The components of the vector are regarded independently of each other, so PBIL only takes into account first-order statistics. Mühlenbein and Paß [7] present a univariate marginal distribution algorithm (UMDA) that estimates the distribution using univariate marginal frequencies in the set of selected parents, and resamples the new points. UMDA shows good performance on linear problems. Both PBIL and UMDA are, however, not appropriate for learning higher-order dependency.

To capture more complex dependency, structures that can express higher-order statistics are necessary. De Bonet et al. [5] present a population-based algorithm using second-order statistics, called MIMIC (mutual information maximizing input clustering). It uses a chain structure to express conditional probabilities. Baluja and Davies [2] propose to use dependency trees to learn second-order
probability distributions and use these statistics to generate new solutions. Pelikan and Mühlenbein [11] suggest the bivariate marginal distribution algorithm (BMDA) as an extension of the UMDA. BMDA uses the pairwise dependencies in order to improve algorithms that use simple univariate marginal distribution. MIMIC, dependency trees, and BMDA can cover pairwise interactions.

Mühlenbein and Mahnig [9] present the factorized distribution algorithm (FDA). Here, the distribution is decomposed into various factors or conditional probabilities. Distribution factorization is obtained by analyzing the problem structure. FDA is able to cover interactions of higher-order and combine important partial solutions effectively. Even though it works very well on additively decomposable problems, FDA requires the prior information about the problem in the form of a problem decomposition and its factorization. Pelikan et al.[12] describe the Bayesian optimization algorithm (BOA) that uses techniques from the field of modeling data by Bayesian networks in order to estimate the joint distribution of promising solutions. BOA is also capable of expressing higher-order interactions. In principle, BOA does not require any predefined structure of a problem. But, if no information on the structure is given, BOA also has some difficulties since the construction of a Bayesian network is not an easy task without any prior information.

Zhang [13] presents Bayesian evolutionary algorithms (BEAs) where optimization is formulated as a probabilistic process of finding an individual with the maximum a posteriori probability (MAP). In previous work, tree-structured neural networks [15] and Lisp-like symbolic programs [14] were used to model the distribution of the data points. In the following section, we present another implementation of the BEA that uses a Helmholtz machine for the estimation of the density of search points and for generating new search points from the estimated density.

3 The Bayesian Evolutionary Algorithm Using Helmholtz Machines

3.1 The Bayesian Evolutionary Algorithm for Optimization

The Bayesian evolutionary algorithm is a probabilistic model of evolutionary computation that is based on the Bayesian inductive principle [13]. Initially, a population $X^0$ of $M$ individuals are generated from a prior distribution $P_0(x)$. Then, the fitness values of the individuals are observed and its likelihood $P(X^t)\theta$ is computed, where $\theta$ is the parameter vector for the probability model. Combining the prior and likelihood, we can compute the posterior probability $P(\theta|X^t)$ of individuals, using the Bayes rule:

$$P(\theta|X^t) = \frac{P(X^t|\theta)P(\theta)}{P(X^t)}.$$  \hspace{1cm} (2)

Since $P(X^t)$ does not depend on the parameter vector $\theta$, maximization of Eqn. (2) is equivalent to maximizing the numerator, i.e.

$$P(\theta|X^t) \propto P(X^t|\theta)P(\theta).$$  \hspace{1cm} (3)
Offspring are then sampled from the posterior distribution and selected into the next generation.

Note that, under the uniform prior for $\theta$, maximization of Eqn. (3) is reduced to the problem of finding the maximum likelihood estimate $\theta^*$:

$$\theta^* = \arg \max_\theta P(\theta | X^t) = \arg \max_\theta P(X^t | \theta). \quad (4)$$

In this paper, we make use of this assumption and present a Bayesian evolutionary algorithm that performs optimization using a Helmholtz machine to estimate $P(X^t | \theta^*)$. The algorithm is summarized in Figure 1.

1. (Initialize) $X^0 \leftarrow$ generate $M$ search points $x_i^0$ from the prior distribution $P_0(x)$. Set generation count $t \leftarrow 0$.
2. (P-step) Estimate the parameter $\theta^*$ of a Helmholtz machine that maximizes $P(X^t | \theta)$.
3. (V-step) Generate $L$ variations $X' = \{x'_1, ..., x'_L\}$ by sampling from the posterior predictive distribution $P_{t+1}(x) = P(x | X^t)$ using $\theta^*$ of the Helmholtz machine.
4. (S-step) Select $M$ points from $X'$ and $X^t$ into $X^{t+1} = \{x'_1, ..., x'_M\}$ based on their fitness values $f(x)$.
5. (Loop) Set $t \leftarrow t + 1$ and go to Step 2.

Fig. 1. Outline of the Bayesian evolutionary algorithm using the Helmholtz machine for density estimation.

In essence, the BEA consists of three steps: probability estimation (P), variation (V), and selection (S) steps. In the P-step, the density of the current population $X^t$ is estimated, in this case, by a Helmholtz machine. The same Helmholtz machine is used throughout the generations. In the V-step, the learned Helmholtz machine is used to generate offspring population $X'$ of $L$ data points. More details on learning and simulating from the Helmholtz machine are described in the next subsection. In the S-step, $M$ best individuals are chosen into the next population $X^{t+1}$ from the union of $X^t$ and $X'$ from the experiments, we use $L = 10M$. This is similar to the $(\mu + \lambda)$ evolution strategy [3] with $\mu = M$, $\lambda = 10M$.

Note the similarity between the general structure of the BEA and the conceptual EDA [8]. The original BEA [15] is more general than this; The one above is a EDA-like variant of it. More general BEAs calculate the maximum a posteriori probability rather than the maximum likelihood and the sample size increases as generation goes on.

### 3.2 Distribution Estimation by the Helmholtz Machine

The Helmholtz machine is a connectionist system with multiple layers of neuron-like binary stochastic processing units connected hierarchically by two sets of weights, recognition weights and generative weights [4]. Bottom-up connections
R, shown as dashed lines in Fig. 2, implement the recognition model. This model is to infer a probability distribution over the underlying causes y (latent variables) of the input vector x:

$$P(y|x, R).$$

(5)

Top-down connections G, shown as solid lines in Fig. 2, implement the generative model. This second model is to reconstruct an approximation to the original input vector x

$$P(x|y, G)$$

(6)

from the underlying representation y captured by the hidden layer of the network. This enables to operate in a self-supervised manner. Both the recognition and generative models operate in a strictly feedforward fashion, with no feedback.

Hinton et al. [6] describe a stochastic algorithm, called the wake-sleep algorithm, to calculate the recognition and generative weights of the Helmholtz machine. There are two phases in the algorithm: a wake phase and a sleep phase. In the “wake” phase, the units are driven bottom-up using the recognition weights, producing a representation of the input vector in the hidden layer. Therefore, the representation $y_c$ produced in the hidden layer of the network provides a representation of the input vector $x_c$.

$$y_c = h(x_c, R).$$

(7)

Although the nodes are driven by the recognition weights, only the generative weights are actually learned during the wake phase using locally available information and the simple delta rule [10]:

$$G' = G + \eta(x_c - Gy_c)y_c,$$

(8)
where \( G \) is the generative weight vector, \( x_c \) is the \( c \)-th sample, \( y_c \) is the value of the latent variables, and \( \eta \) is the learning rate. In effect, this phase of the learning process makes generative weights be adapted to increase the probability that they would reconstruct the correct activity vector in the layer below.

In the "sleep" phase of the algorithm, the recognition weights \( R \) are turned off. And all of the units in the network is driven using the generative weights, starting at the hidden layer and working down to the input units. Because the nodes are stochastic and the values of the hidden units, \( y \), are randomly chosen, repeating this process would typically gives rise to many different "fantasy" vectors \( x \) on the input layer:

\[
x_k = g(y_k, G).
\]

These fantasies supply an unbiased sample of the network's generative model of the data. Having produced a fantasy, the recognition weights are adjusted by the simple delta rule [10]:

\[
R' = R + \eta (y_k - Rx_k)x_k,
\]

where \( R \) is the recognition weight vector, \( y_k \) is the \( k \)-th latent vector, and \( x_k \) is the \( f \)-th fantasy vector. The "sleep" phase uses only locally available information without reference to any observation. This is why offspring in the Bayesian evolutionary algorithm can be efficiently sampled from the distribution.

In effect, the Helmholtz machine estimates the distribution of the data points \( X' \), i.e. find the parameters \( \theta^* = (R^*, G^*) \) that maximize the likelihood \( P(X') \). After the distribution is learned, the samples from this distribution can be generated by randomly setting the latent variables and then propagating the values down to the input layer, just as the process in the sleep mode of the wake-sleep algorithm. This process is equivalent to sampling \( L \) offspring from the posterior predictive distribution since the following holds:

\[
P_{t+1}(x) \equiv P(x|X') = \int_{y} \int_{\theta} P(x|y, \theta)P(y, \theta|X')d\theta dy
\]

\[
\approx \int_{y} P(x|y, \theta^*)P(y, \theta^*|X')dy
\]

\[
\approx \sum_{k=1}^{L} P(x|y_k, \theta^*),
\]

where \( \theta^* = (R^*, G^*) \) is the maximum likelihood estimator for data \( X' \), and \( P(x|y_k, \theta^*) \) is the generative model for the latent vectors \( y_k \) which are independently sampled from the uniform distribution.

4 Experimental Results

Experiments have been performed on three benchmark problems from the literature [11]. They are the one-max function, quadratic function, and 3-deceptive function as defined below.
- One-max function:

\[ f_{\text{onemax}}(x) = \sum_{i=0}^{n-1} x_i, \]  

where \( x_i \) is the value on the \( i \)th position in bit string \( x \). The one-max function is a simple linear function that is just the sum of all bits in a string.

- Quadratic function:

\[ f_{\text{quadratic}}(x, \pi) = \sum_{i=0}^{\frac{n}{2}-1} f_2(x_{\pi(2i)}, x_{\pi(2i+1)}), \]

where \( \pi \) is defined as

\[ \pi_k(i) = \left\lfloor \frac{n(i \mod k) + 1}{k} \right\rfloor. \]

For this problem, the permutation of order 2, \( \pi_2 \), was used and \( f_2 \) is defined as

\[ f_2(u, v) = 0.9 - 0.9(u + v) + 1.9uv. \]

With both arguments equal to 1 we get \( f_2(1, 1) = 1 \). Therefore, the optimum is clearly in the string with all 1's.

- 3-deceptive function:

\[ f_{\text{3decep}}(x, \pi) = \sum_{i=0}^{\frac{3}{2}-1} f_3(x_{\pi(3i)} + x_{\pi(3i+1)} + x_{\pi(3i+2)}), \]

where \( \pi_3 \) is used and \( f_3 \) is defined as

\[ f_3(u) = \begin{cases} 0.9 & \text{if } u = 0, \\ 0.8 & \text{if } u = 1, \\ 0 & \text{if } u = 2, \\ 1 & \text{otherwise}. \end{cases} \]

The performance of the Helmholtz machine was compared with that of the simple genetic algorithm (sGA). The sGA we use is the usual implementation that is based on one cut-point crossover, one point mutation, and roulette-wheel selection. The parameters for sGA were: maximum generation = \( 10^6 \), population size = \( 10^3 \), crossover rate = 0.5, and mutation rate = 0.01. The parameters of the BEA with the Helmholtz machine were: maximum generation = \( 10^3 \), population size = \( 10^3 \), learning rate = 0.001, and the number of learning iterations = \( 10^3 \). To select the next population, upper 10% truncation selection was used. For objective comparison, the parameter values for both methods were set as similar as possible. We use a 2-layer Helmholtz machine for the Beysian evolutionary algorithm, and only one latent variable was used for solving both the
Table 1. Results for the one-max function.

<table>
<thead>
<tr>
<th>Prob. Size</th>
<th>Succ %</th>
<th>#iterations</th>
<th>CPU time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BEA</td>
<td>sGA</td>
<td>BEA</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>100</td>
<td>5.0</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>27.8</td>
</tr>
<tr>
<td>150</td>
<td>100</td>
<td>100</td>
<td>54.2</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>100</td>
<td>70.0</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>100</td>
<td>96.6</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>100</td>
<td>139.6</td>
</tr>
</tbody>
</table>

Table 2. Results for the quadratic function.

<table>
<thead>
<tr>
<th>Prob. Size</th>
<th>Succ %</th>
<th>#iterations</th>
<th>CPU time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BEA</td>
<td>sGA</td>
<td>BEA</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>100</td>
<td>20.1</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>48.0</td>
</tr>
<tr>
<td>150</td>
<td>100</td>
<td>0</td>
<td>58.6</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>0</td>
<td>85.1</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>0</td>
<td>118.8</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>0</td>
<td>134.0</td>
</tr>
</tbody>
</table>

Table 3. Results for the 3-deceptive function.

<table>
<thead>
<tr>
<th>Prob. Size</th>
<th>Succ %</th>
<th>#iterations</th>
<th>CPU time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BEA</td>
<td>sGA</td>
<td>BEA</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>30</td>
<td>3.4</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0</td>
<td>6.3</td>
</tr>
<tr>
<td>45</td>
<td>100</td>
<td>0</td>
<td>7.8</td>
</tr>
<tr>
<td>60</td>
<td>100</td>
<td>0</td>
<td>10.5</td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>0</td>
<td>13.1</td>
</tr>
<tr>
<td>120</td>
<td>100</td>
<td>0</td>
<td>15.7</td>
</tr>
</tbody>
</table>

Table 4. Results for various pop size on the one-max function with problem size 100.

<table>
<thead>
<tr>
<th>Prob. Size</th>
<th>Succ %</th>
<th>#iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BEA</td>
<td>sGA</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
one-max and quadratic functions. While more than one latent variable was used for 3-deceptive function. A constructive algorithm was devised to solve the for 3-deceptive function. It starts from one latent variable, and increases the number of latent variables during learning.

Tables 1-4 summarize the results. The results shown are average values over 10 runs. The CPU time was measured on Intel Pentium II - 350 Mhz PC with Windows 2000. The algorithm is considered as converged if an optimal solution is found. The entry of the tables marked with '-' means that the algorithm did not find the optimal solution. The results show that the Bayesian evolutionary algorithms with Helmholtz machines outperform the simple genetic algorithm both in the success rate and the number of iterations. BEAs find the optimal solutions all the time while the simple GA finds solutions for easy problems such as one-max and small size instances of the quadratic and 3-deceptive function. It can be observed that from Table 4 that the BEA with Helmholtz machine can solve the problem using a very small data set. Even though the population size is 50, BEA can find the optimal solution. The improved algorithms, BEA with constructive Helmholtz machines could solve the 3-deceptive function very efficiently, significantly outperforming the simple genetic algorithm.

The number of iterations for the sGA denotes the number of generations, while that for the BEA with the Helmholtz machine denotes the number of sampling the population, excluding the distribution estimation (learning) time. Therefore, it is hard to infer the CPU time from the number of iterations only. For comparing the real evaluation time, we measured the CPU time for BEA with Helmholtz machines and the simple genetic algorithm. Even if BEAs took more CPU time than sGAs, BEAs could solve larger size problems. It is also interesting that the number of evaluations for sGAs grows exponentially while that for BEAs grows almost linearly.

5 Conclusions

We presented a distribution estimation algorithm that is based on the Helmholtz machine. Our empirical results show that the Bayesian evolutionary optimization algorithms using Helmholtz machines outperform the simple genetic algorithms in several conditions. The superiority of the probabilistic algorithms tend to grow linearly as the problem complexity and size increase and solve the using very small data sets. Future work includes the analysis of the effect of the number of latent variables and the number of hidden layers in the Helmholtz machine for more effective estimation of population distribution.

Acknowledgments

This research was supported in part by the Korea Science and Engineering Foundation (KOSEF) under Grant 981-0920-107-2, by the Korea Ministry of Science and Technology through KISTEP under Grant BR-2-1-G-06, and by BK21 program.
References


Multiobjective Optimisation
The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization

David W. Corne, Joshua D. Knowles, Martin J. Oates

School of Computer Science, Cybernetics and Electronic Engineering
University of Reading, Reading RG6 6AY, UK
D.W.Corne, J.D.Knowles, M.J.Oates@reading.ac.uk

Abstract. We introduce a new multiobjective evolutionary algorithm called PESA (the Pareto Envelope-based Selection Algorithm), in which selection and diversity maintenance are controlled via a simple hyper-grid based scheme. PESA’s selection method is relatively unusual in comparison with current well known multiobjective evolutionary algorithms, which tend to use counts based on the degree to which solutions dominate others in the population. The diversity maintenance method is similar to that used by certain other methods. The main attraction of PESA is the integration of selection and diversity maintenance, whereby essentially the same technique is used for both tasks. The resulting algorithm is simple to describe, with full pseudocode provided here and real code available from the authors. We compare PESA with two recent strong-performing MOEAs on some multiobjective test problems recently proposed by Deb. We find that PESA emerges as the best method overall on these problems.

1 Introduction

Following seminal work on multiobjective evolutionary algorithms (MOEAs) such as the Niched Pareto Genetic Algorithm (Horn et al., 1994; Horn and Nafpliotis, 1994), and the Non-Dominated Sorting method (Srinivas and Deb, 1994), more recent work on MOEAs has resulted in improved techniques which provide fast and effective approximations to the Pareto frontier for a variety of benchmark problems. These new methods include SPEA (Strength Pareto Evolutionary Algorithm – Zitzler and Thiele, 1999), and PAES (Pareto Archived Evolution Strategy – Knowles and Corne, 2000). Both PAES and SPEA have been shown to outperform sophisticated versions of NPGA and NDS on a variety of benchmark problems, while various other modern MOEAs exist which have been shown to perform well on particular applications (eg: Fonseca and Fleming, 1995; Parks and Miller, 1998), but have not yet been systematically compared against other modern MOEAs on a common set of test problems.

Somewhat removed from the MOEA research community, researchers in multiple criteria decision making (MCDM) and operations research communities have also worked on multiobjective optimization over the years, and produced
a variety of local-search based multiobjective techniques. These include, for example, Czyzak and Jaszkiewicz (1998), Gandibleux et al. (1996), and Hansen (1996; 1997). Cross comparison of techniques between these communities and the MOEA community has not yet been done to any significant extent, although it seems clear from the results reported in Zitzler and Thiele (1999) and Knowles and Corne (2000) that the MOEA community would be best represented by either SPEA or PAES in such a comparative study.

Here we introduce a new multiobjective evolutionary algorithm called PESA (the Pareto Envelope-based Selection Algorithm), which incorporates ideas from each of SPEA and PAES, and which performs quite well in comparison to both of these new methods. In fact, PESA seems to be the overall best on the suite of test functions used in this paper.

Like SPEA, PESA uses a small ‘internal population’ and a (usually) larger ‘external population’. The external population is actually the archive which stores the current approximation to the Pareto front, and the internal population are new candidate solutions vying for incorporation into the archive. Like PAES, PESA implicitly maintains a hyper-grid division of phenotype space which allows it to keep track of the degree of crowding in different regions of the archive. However, unlike both PAES and SPEA, selection in PESA is based on this crowding measure (in PAES, selection is trivial – there is just a single current solution, as in local search, which is the parent for a mutation; in SPEA, selection is based on a sophisticated and ingenious ‘strength’ measure). However, like both PAES and SPEA, replacement (deciding what must leave the archive if it becomes over-full) is also based on a crowding measure.

Each of SPEA, PAES and PESA are compared on six test functions from Deb (1998), and results are analysed using a sophisticated statistical comparison method based on ideas from Fonseca and Fleming (1995a). Both SPEA and PESA have already been found to outperform NDS and NPGA on these functions (Zitzler et al, 1999; Knowles and Corne, 1999). We find that each algorithm turns out to be the winner on at least one of the test functions, but PESA is the overall best performer.

The remainder of the paper is organized as follows. In Section 2 we describe the PESA algorithm, and in Section 3 we describe the test functions. Experimental design is discussed in Section 4 and Results given and discussed in Section 5. Section 6 indicates our conclusions and notes for further work.

2 The PESA Algorithm

Apart from standard parameters such as crossover and mutation rates, PESA has two parameters concerning population size, and one parameter concerning the hyper-grid crowding strategy. A high-level description of the PESA algorithm is as follows, in which the two population based parameters are $P_I$ (the size of the internal population, IP) and $P_E$ (the maximum size of the archive, or ‘external population’, EP).
1. Generate and evaluate each of an initial ‘internal’ population (IP) of $P_I$ chromosomes, and initialise the ‘external’ population (EP) to the empty set.

2. Incorporate the non-dominated members of IP into EP.

3. If a termination criterion has been reached, then stop, returning the set of chromosomes in EP as the result. Otherwise, delete the current contents of IP, and repeat the following until $P_I$ new candidate solutions have been generated:

   -- With probability $p_C$, select two parents from EP, produce a single child via crossover, and mutate the child. With probability $(1 - p_C)$, select one parent and mutate it to produce a child.

4. return to Step 2.

In the ‘archive incorporation’ step (step 2), the current set of new candidate solutions (IP) are incorporated into the archive one by one. A candidate may enter the archive if it is non-dominated within IP, and if is not dominated by any current member of the archive. Once a candidate has entered the archive, members of the archive which it dominated (if any) will be removed. If the addition of a candidate renders the archive over-full (its size temporarily becomes $P_E + 1$), then a current member of EP is removed. The choice of which member is removed will be detailed later.

The selection of a parent in PESA, within step 3, is based on the degree of crowding in different regions of the archive. This is simply illustrated by Figure 1. In the figure, a number of points are shown in the phenotype space of a two-objective minimization problem. The circles are non-dominated points, and hence might currently be in the PESA archive. The squares are dominated by members of the archive, but they may be points in the current internal population.

![Figure 1. PESA's crowding strategy](image-url)
The crowding strategy in PESA works by forming an implicit hyper-grid which divides (normalised) phenotype space into hyper-boxes. In Figure 1, this is illustrated by the thick horizontal and vertical lines; the problem is two-dimensional and hence these hyper-boxes are simply squares. Each chromosome in the archive is associated with a particular hyper-box in phenotype space, and has an attribute called a 'squeeze factor', which is simply the total number of other chromosomes in the archive which inhabit the same box. For example, the squeeze factor of chromosome $A$ in Figure 1 is 2, and the squeeze factor of chromosome $B$ is 1. The squeeze factor is used for selective fitness. For example, when PESA uses binary tournament selection, two chromosomes are taken at random from the archive, and the one with the lowest squeeze factor is chosen (breaking ties randomly), hence orienting search effort towards areas of the emerging Pareto frontier which currently have little representation in the population.

The squeeze factor is also used for archive update. As indicated above, the incorporation of an internal population member into the archive may lead to the archive size temporarily exceeding the maximum size $E_P$. One solution must therefore be removed from the archive. The choice is made by first finding the maximal squeeze factor in the population, and removing an arbitrary chromosome which has this squeeze factor.

We can now more easily distinguish between the selection and archive-update strategies of PAES, SPEA, and PESAg again with reference to Figure 1. As we have seen, PESA uses the squeeze factor for both of these tasks. PAES uses a hyper-grid strategy, but only for archive update. In the selection step of PAES, since PAES is actually a hillclimbing algorithm, selection is only between two candidates: the current solution, and a mutant. If the archive is full, squeeze factor is indeed used in PAES, and if the mutant has a lower squeeze factor than the current then it will become the new current solution. To some extent PAES is therefore similar to PESA in employing squeeze factor in both selection and archive update, however the fact that PAES is a local search method and PESA is a population based technique render them fundamentally different algorithms.

SPEA, on the other hand, uses a novel selection strategy in which a 'strength' is associated with each member of the archive, based on the number of chromosomes in the internal population which it dominates. Each member of the internal population is also given a strength value, based on summing the strengths of the archive members which dominate it. This is illustrated by the dashed lines in Figure 1. Chromosome $A$ clearly has a higher strength value than chromosome $B$, since it dominates more members of the internal population than $B$. In SPEA, this means that $B$ will have a higher selective fitness than $A$ (fitness for selection is based on minimal strength value).

Archive update in SPEA is also quite different to the hyper-grid strategy. In SPEA, a simple agglomerative clustering algorithm is used based on phenotypic distance, which prunes an archive of $k > E_P$ chromosomes into an archive of $E_P$ chromosomes. It first produces $E_P$ clusters of chromosomes from the over-full archive. The chromosomes in each cluster which are nearest to its centre then become the $E_P$ chromosomes in the pruned archive.
3 The Test Problems T1–T6

Deb (1998) gives a procedure for designing tunable test functions for multiobjective optimisation, involving a range of characteristics which may or may not be present to varying degrees in the Pareto surface. In particular, Deb argues that key aspects of a multiobjective search landscape which would cause difficulty for an optimizer are, among others, discontinuities in the Pareto front, non-uniform distribution of solutions along the Pareto front, and deception. The test function design scheme produces test functions which vary in these aspects.

Six test functions, $T_1$-$T_6$, designed using Deb’s scheme, were used in a comparison of the performance of eight different MOEAs by Zitzler et al (1999). $T_1$ has a convex Pareto front but no particular difficulty characteristics; $T_2$ has a non-convex Pareto front; $T_3$ has many discontinuities in the Pareto front; $T_4$ is highly multimodal and has $2^{10}$ Pareto fronts; $T_5$ is a deceptive problem, and $T_6$ has a non-uniformly distributed search space with solutions non-uniformly distributed along the Pareto front.

Each is a two-objective problem defined on $m$ parameters, in which both objectives are to be minimized. In five of the problems the parameters $x_i$ were coded as a binary string decoded such that $x_i \in [0, 1]$. The remaining function ($T_5$) also employed a binary chromosome but this time unitation was used to evaluate each of the parameters. The experiments in this paper employ identical functions to those presented in Zitzler et al (1999) and are coded onto chromosomes using identical numbers of bits to represent each parameter.

The results of Zitzler et al (1999) indicated that on three of the test functions, $T_1$, $T_3$, and $T_6$, SPEA (Zitzler and Thiele, 1999) generates solution sets which consistently dominate all of the other algorithms tested. On test function $T_4$ SPEA is clearly superior to all other algorithms, although it no longer consistently beats two of the algorithms, namely NSGA (Srinivas and Deb, 1994), and SOEA (a single objective EA run 100 times with a different randomly chosen linear combination of the objectives). On $T_5$, SPEA and NSGA perform very similarly, with SOEA performing slightly better according to the measures used by Zitzler et al (1999).

In summary, Zitzler’s study indicated that SPEA seemed to be the best algorithm overall (of the eight tested) on Deb’s test functions. In this study, we hence use SPEA as our main comparative algorithm, but also use PAES. We therefore compare the performance of SPEA against two rival techniques, one of which has been found best on the Deb functions when compared with a wide range of other multiobjective approaches.

4 Experimental Design

4.1 Experiments

Our experiments sought to determine the relative quality of PESA, SPEA and PAES on the Deb test functions. Guided by real-world considerations, we were also very interested in speed. That is, we were interested in the development
of the approximations to the Pareto front over time. In some applications (for example, design) the optimisation method must develop as good as possible an approximation to the Pareto tradeoff surface, but without any real-time or near real-time processing constraints. In other applications, for example optimal multiobjective control, good approximations to the Pareto front must be produced very quickly.

To compare the algorithms along these lines we therefore performed three sets of experiments, to different time limits, of 1,000, 5,000, 20,000 evaluations. In each set of experiments, 20 trial runs of PESA, PAES, and SPEA were performed on each of the six test functions. Parameters were set as described in Table 1.

<table>
<thead>
<tr>
<th>Parameter Settings</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Crossover rate</strong></td>
<td>0.7 in PESA and SPEA; not used in PAES</td>
</tr>
<tr>
<td><strong>Crossover method</strong></td>
<td>uniform crossover in PESA and SPEA; not used in PAES</td>
</tr>
<tr>
<td><strong>Mutation rate</strong></td>
<td>bit-flip mutation set to $1/L$ where $L$ is chromosome length</td>
</tr>
<tr>
<td><strong>Populations</strong></td>
<td>archive 100 in all algorithms, IP size 10 in PESA and SPEA</td>
</tr>
<tr>
<td><strong>Chromosome lengths</strong></td>
<td>900 in $T_1$, $T_2$, and $T_3$, 300 in $T_4$ and $T_5$, 80 in $T_6$</td>
</tr>
<tr>
<td><strong>Hyper-grid size</strong></td>
<td>32x32 grid in PESA and PAES, not used in SPEA</td>
</tr>
</tbody>
</table>

Table 1. Parameter settings

In the next section we summarise the statistical comparison method used to analyse the results within a set of experiments.

4.2 Statistics

Given the results of 20 or more trial runs for each algorithm, we compare the performance of two or more multiobjective optimisers using a method proposed originally by Fonseca and Fleming (1995a) which we have implemented with certain extensions. When comparing two multiobjective algorithms (A and B), this method essentially returns two numbers: the percentage of the Pareto frontier on which algorithm A conclusively beats algorithm B (based on a Mann-Whitney U test at the 95% confidence level), and the percentage of the Pareto frontier on which algorithm B conclusively beat algorithm A.

Typically, for example, two very good MOEAs with similar performance on a problem might yield a result like $[3.7, 6.1]$, indicating that each algorithm was definitely better than the other in small regions of the space, but they performed similarly well on the majority of the Pareto frontier. A clear indication that one algorithm is superior to another, however, is given by a comparison result such as $[58.3, 2.2]$, or $[100, 0.0]$.

In a comparison of $k > 2$ algorithms, the comparison code performs pairwise statistical comparisons, as before, for each of the $k(k-1)/2$ distinct pairs of algorithms. The results then show, for each algorithm, on what percentage of the space we can be statistically confident that it was unbeaten by any of the other $k - 1$ algorithms, and on what percentage of the space it beat all $k - 1$
algorithms. For example, in Table 2, we can see that, on problem $T_1$, PESA was conclusively better than both PAES and SPEA on more than half of the Pareto Tradeoff surface, and was only bettered (in this case by PAES) on 28% of this frontier. SPEA performed particularly poorly in this case, being conclusively outperformed by the other methods on all but about 1% of the Pareto surface.

5 Results and Discussion

Table 2 summarises all results for the set of experiments in which each trial run was allowed just 1,000 fitness evaluations. The best performing algorithm for each problem has its table entries highlighted in bold; when there is little difference between the best two (or all three), both sets of entries are highlighted in bold.

<table>
<thead>
<tr>
<th>Problem (beats all)</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$T_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAES</td>
<td>44.7 (28.0)</td>
<td>0.0 (0.0)</td>
<td>0.0 (0.0)</td>
<td>24.6 (0.0)</td>
<td>0.9 (0.0)</td>
<td>16.3 (0.0)</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.9 (0.0)</td>
<td>1.8 (0.0)</td>
<td>98.9 (0.0)</td>
<td>18.2 (0.0)</td>
<td>100 (0.0)</td>
<td>100 (0.0)</td>
</tr>
<tr>
<td>PESA</td>
<td>72.0 (54.4)</td>
<td>100 (98.2)</td>
<td>100 (1.1)</td>
<td>100 (57.2)</td>
<td>63.2 (0.0)</td>
<td>100 (0.0)</td>
</tr>
</tbody>
</table>

Table 2. Comparison of PAES, SPEA and PESA at 1,000 evaluations

As Table 2 shows, PESA was clearly the best method on three of the functions, and joint best with SPEA on a further two. On the one remaining function it achieved the second-best performance. SPEA is clearly best on just one function, and joint best with PESA on two. PAES is the worst performer here, being clearly worst on three of the test functions, and second or joint second best on the remaining three.

In Table 3, we can see the results when trials were set to run for 5,000 evaluations. PESA is now clearly best on two of the six test functions, and joint best with SPEA on a further two. It is second best on the remaining two test functions. Given the greater time limit available, PAES starts to improve its comparative performance, now being clearly best on one function and joint best on two. SPEA seems to lose out to PAES somewhat with the increased time limit, now being clearly best on one function, but worst or joint worst on the remaining five.

Table 4 indicates the results when trials were given a full 20,000 evaluations each. PESA is now best or joint best on five of the six test functions. In the other case it is second best, beating SPEA, although rather a poor second to PAES. PAES is clearly best in one case and joint best in another, while SPEA is joint-best in two cases, second best in two cases, and worst in the remaining two cases.

All of these results are summarised in Table 5, in which we show the rank for each algorithm at each evaluation time limit, on each problem. For example, the
entry ‘1/3/3’ for SPEA under $T_3$ indicates that for the shorter time limit it was the best algorithm (indicated by the ‘1’), but then became third-best (i.e., worst) at the moderate time limit of 5,000 evaluations (indicated by the first ‘3’), and also at the longest time limit of 20,000 evaluations (indicated by the second ‘3’).

<table>
<thead>
<tr>
<th>Problem</th>
<th>PAES</th>
<th>SPEA</th>
<th>PESA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>2/1/1</td>
<td>99.6 (98.7)</td>
<td>0.0 (0.0)</td>
</tr>
<tr>
<td>$T_2$</td>
<td>0.5 (0.0)</td>
<td>41.4 (2.4)</td>
<td>97.6 (58.6)</td>
</tr>
<tr>
<td>$T_3$</td>
<td>47.9 (9.1)</td>
<td>37.4 (0.0)</td>
<td>90.9 (18.3)</td>
</tr>
<tr>
<td>$T_4$</td>
<td>99.9 (0.0)</td>
<td>100 (0.0)</td>
<td>100 (2.3)</td>
</tr>
<tr>
<td>$T_5$</td>
<td>0.0 (0.0)</td>
<td>100 (2.3)</td>
<td>97.7 (0.0)</td>
</tr>
<tr>
<td>$T_6$</td>
<td>11.2 (8.5)</td>
<td>0.1 (0.0)</td>
<td>97.5 (88.7)</td>
</tr>
</tbody>
</table>

Table 4. Comparison of PAES, SPEA and PESA at 20,000 evaluations

With reference to Table 5, PAES clearly seems to be a slow starter, with poor comparative performance at short time limits, but quite able to compete with other methods at longer time limits. To some extent this would indicate that PAES is best not used for (near) real-time applications. However, the time complexity of PAES algorithm is much more favourable than population-based multiobjective optimisers (Knowles and Corne, 2000), so this conclusion may not be fair. The conclusion stands, however, in the case of applications where fitness evaluation is the bottleneck.

There is no clear pattern to the performance of SPEA as we increase the time limit, except for the fact that its rank generally worsens from the fast to the moderate time limit as PAES ‘catches up’ with it. PESA, on the other hand, seems to successfully hold its own at all time limits. Reference to table 5 seems to suggest it is generally the best of the three algorithms compared here on these test functions.
It is instructive to now consider performance in terms of the problem characteristics. $T_\infty$ is not a particularly difficult problem (lacking multimodality, deception, and so forth), and this was the only problem on which PESA was not best or joint best at the longest time limit. It seems reasonable to conclude that this is because the hillclimbing strategy of PAES (the clear winner on this problem) is particularly suitable here, the problem has none of the characteristics that the sophisticated aspects of PESA or SPEA (such as use of crossover, and selection from a population) are designed to address. PAES therefore makes aggressive and fruitful use of its time on this problem, while PESA and SPEA essentially waste much of their search effort. This intuition seems to be confirmed by the fact that one of the problems on which PAES performed particularly badly was $T_5$, which is the deceptive problem in the suite.

6 Conclusion

We have described the Pareto Envelope-based Selection Algorithm, and compared its performance with two recent strong-performing multiobjective optimisers on a suite of test functions devised by Deb (1998). Comparative performance was measured using a sophisticated statistical comparison technique, and performance was compared in respect of three separate time limits, reflecting the varying needs for solution speed in real-world applications. We found that PESA generally outperforms both SPEA and PAES on these functions. It was never the worst of the three, and tended to perform best or joint best (with SPEA), whether solutions were needed quickly, moderately quickly, or without stringent time constraints.

PAES and SPEA are both modern multiobjective optimisation methods which have previously been found to outperform a wide range of classical methods on a wide range of problems. The relative performance of PESA demonstrated here therefore suggests that PESA may be well qualified to join these two methods in the current set of ‘best performers’ in the multiobjective evolutionary algorithm community. However, results on a limited set of test functions must always be regarded as tentative, and hence much further work is needed on further problems to better assess the value of PESA.

Acknowledgments

The authors are grateful to British Telecommunications Plc for financial support of the second and third authors.

References


A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan

Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
{deb,samira,apratap,mary}@iitk.ac.in
http://www.iitk.ac.in/kangal

Abstract. Multi-objective evolutionary algorithms which use non-dominated sorting and sharing have been mainly criticized for their (i) $O(MN^3)$ computational complexity (where $M$ is the number of objectives and $N$ is the population size), (ii) non-elitism approach, and (iii) the need for specifying a sharing parameter. In this paper, we suggest a non-dominated sorting based multi-objective evolutionary algorithm (we called it the Non-dominated Sorting GA-II or NSGA-II) which alleviates all the above three difficulties. Specifically, a fast non-dominated sorting approach with $O(MN^2)$ computational complexity is presented. Second, a selection operator is presented which creates a mating pool by combining the parent and child populations and selecting the best (with respect to fitness and spread) $N$ solutions. Simulation results on five difficult test problems show that the proposed NSGA-II, in most problems, is able to find much better spread of solutions and better convergence near the true Pareto-optimal front compared to PAES and SPEA—two other elitist multi-objective EAs which pay special attention towards creating a diverse Pareto-optimal front. Because of NSGA-II's low computational requirements, elitist approach, and parameter-less sharing approach, NSGA-II should find increasing applications in the years to come.

1 Introduction

Over the past decade, a number of multi-objective evolutionary algorithms (MOEAs) have been suggested [8, 3, 5, 11]. The primary reason for this is their ability to find multiple Pareto-optimal solutions in one single run. Since the principal reason why a problem has a multi-objective formulation is because it is not possible to have a single solution which simultaneously optimizes all objectives, an algorithm that gives a large number of alternative solutions lying on or near the Pareto-optimal front is of great practical value.

The Non-dominated Sorting Genetic Algorithm (NSGA) proposed in Srinivas and Deb [8] was one of the first such evolutionary algorithms. Over the years, the main criticism of the NSGA approach have been as follows:

High computational complexity of non-dominated sorting: The non-dominated sorting algorithm in use until now is $O(MN^3)$ which in case of large population sizes is very expensive, especially since the population needs to be sorted in every generation.
Lack of elitism: Recent results [10,7] show clearly that elitism can speed up the performance of the GA significantly, also it helps to prevent the loss of good solutions once they have been found.

Need for specifying the sharing parameter $\sigma_{share}$: Traditional mechanisms of insuring diversity in a population so as to get a wide variety of equivalent solutions have relied heavily on the concept of sharing. The main problem with sharing is that it requires the specification of a sharing parameter ($\sigma_{share}$). Though there has been some work on dynamic sizing of the sharing parameter [4], a parameterless diversity preservation mechanism is desirable.

In this paper, we address all of these issues and propose a much improved version of NSGA which we call NSGA-II. From the simulation results on a number of difficult test problems, we find that NSGA-II is, in general, better than PAES and SPEA—two other elitist multi-objective evolutionary algorithm—in terms of converging near the Pareto-optimal front and maintaining diversity among obtained solutions. These results encourage the application of NSGA-II to more complex and real-world multi-objective optimization problems.

2 Elitist Multi-Objective Evolutionary Algorithms

In the study of Zitzler, Deb, and Theile [10], it was clearly shown that elitism helps in achieving better convergence in MOEAs. Among the existing elitist MOEAs, Zitzler and Thiele’s [11] strength Pareto EA (SPEA), Knowles and Corne’s Pareto-archived evolution strategy (PAES) [6], and Rudolph’s [7] elitist GA are well known.

Zitzler and Thiele [11] suggested an elitist multi-criterion EA with the concept of non-domination in their strength Pareto EA (SPEA). They suggested maintaining an external population at every generation storing all non-dominated solutions discovered so far beginning from the initial population. This external population participates in genetic operations. At each generation, a combined population with the external and the current population is first constructed. All non-dominated solutions in the combined population are assigned a fitness based on the number of solutions they dominate and dominated solutions are assigned fitness worse than the worst fitness of any non-dominated solution. This assignment of fitness makes sure that the search is directed towards the non-dominated solutions. A deterministic clustering technique is used to ensure diversity among non-dominated solutions. Although the implementation suggested in [11] is $O(M N^2)$, with proper book-keeping the complexity of SPEA can be reduced to $O(M N^2)$.

Knowles and Corne [6] suggested a simple MOEA using an evolution strategy (ES). In their Pareto-archived ES (PAES) with one parent and one child, the child is compared with respect to the parent. If the child dominates the parent, the child is accepted as the next parent and the iteration continues. On the other hand, if the parent dominates the child, the child is discarded and a new mutated solution (a new child) is found. However, if the child and the parent do not dominate each other, the choice between the child and the parent is made by comparing them with an archive of best solutions found so far. The child is compared with the archive to check if it dominates any member of the archive. If yes, the child is accepted as the new parent and the dominated solution is
eliminated from the archive. If the child does not dominate any member of the archive, both parent and child are checked for their nearness with the solutions of the archive. If the child resides in a least crowded region in the parameter space among the members of the archive, it is accepted as a parent and a copy of added to the archive. Authors have calculated the worst case complexity of PAES for \( N \) evaluations as \( O(aMN) \), where \( a \) is the archive length. Since the archive size is usually chosen proportional to the population size \( N \), the overall complexity of the algorithm is \( O(MN^2) \).

Rudolph [7] suggested, but did not simulate, a simple elitist multi-objective EA based on a systematic comparison of individuals from parent and offspring populations. The non-dominated solutions of the offspring population are compared with that of parent solutions to form an overall non-dominated set of solutions, which becomes the parent population of the next iteration. If the size of this set is not greater than the desired population size, other individuals from the offspring population are included. With this strategy, he has been able to prove the convergence of this algorithm to the Pareto-optimal front. Although this is an important achievement in its own right, the algorithm lacks motivation for the second task of maintaining diversity of Pareto-optimal solutions.

### 3 Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II)

The non-dominated sorting GA (NSGA) proposed by Srinivas and Deb in 1994 has been subjected to a number of criticism, as mentioned earlier. In this section, we suggest NSGA-II, which alleviate all these difficulties. We begin by presenting a number of different modules that form part of NSGA-II.

#### 3.1 A fast non-dominated sorting approach

In order to sort a population of size \( N \) according to the level of non-domination, each solution must be compared with every other solution in the population to find if it is dominated. This requires \( O(MN) \) comparisons for each solution, where \( M \) is the number of objectives. When this process is continued to find the members of the first non-dominated class for all population members, the total complexity is \( O(MN^2) \). At this stage, all individuals in the first non-dominated front are found. In order to find the individuals in the next front, the solutions of the first front are temporarily discounted and the above procedure is performed again. The procedure is repeated to find the subsequent fronts. As can be seen the worst case (when there exists only one solution in each front) complexity of this algorithm without any book-keeping is \( O(MN^3) \). In the following we describe a fast non-dominated sorting approach which will require at most \( O(MN^2) \) computations.

This approach is similar in principle to above approach, except that a better book-keeping strategy is performed to make it a faster algorithm. In this approach, every solution from the population is checked with a partially filled population for domination. To start with, the first solution from the population is kept in a set \( P' \). Thereafter, each solution \( p \) (the second solution onwards) is compared with all members of the set \( P' \) one by one. If the solution \( p \) dominates any member \( q \) of \( P' \), then solution \( q \)
is removed from $P'$. This way non-members of the non-dominated from get deleted from $P'$. Otherwise, if solution $p$ is dominated by any member of $P'$, the solution $p$ is ignored. If solution $p$ is not dominated by any member of $P'$, it is entered in $P'$. This is how the set $P'$ grows with non-dominated solutions. When all solutions of the population is checked, the remaining members of $P'$ constitute the non-dominated set.

```
fast-nondominated-sort(P)
P' = {1} include first member in P'
for each $p \in P \land p \notin P'$ take one solution at a time
P' = P' \cup \{p\} include p in P' temporarily
for each $q \in P' \land q \neq p$
if $p < q$, then $P' = P'\{q\}$ if p dominates a member of $P'$, delete it
else if $q < p$, then $P' = P'\{p\}$ if p is dominated by other members of $P'$, do not include p in P'
```

To find other fronts, the members of $P'$ will be discounted and the above procedure is repeated.

Here, we observe that the second element of the population is compared with only one solution $P'$, the third solution with at most two solutions of $P'$, and so on. This requires a maximum of $O(N^2)$ domination checks. Since each domination check requires $M$ function value comparisons, the maximum complexity of this approach is also $O(MN^2)$.

### 3.2 Density estimation

To get an estimate of the density of solutions surrounding a particular point in the population we take the average distance of the two points on either side of this point along each of the objectives. This quantity $i_{\text{distance}}$ serves as an estimate of the size of the largest cuboid enclosing the point $i$ without including any other point in the population (we call this the crowding distance). In Figure 1, the crowding distance of the $i$-th solution in its front (marked with solid circles) is the average side-length of the cuboid (shown with a dashed box). The following algorithm is used to calculate the crowding distance of each point in the set $I$:

```
crowding-distance-assignment(I)
l = |I| number of solutions in I
for each $i$, set $I[i]_{\text{distance}} = 0$ initialize distance
for each objective $m$
$I = \text{sort}(I, m)$ sort using each objective value
$I[1]_{\text{distance}} = I[l]_{\text{distance}} = \infty$ so that boundary points are always selected
for $i = 2$ to $(l - 1)$ for all other points
$I[i]_{\text{distance}} = I[i]_{\text{distance}} + (I[i + 1].m - I[i - 1].m)$
```

Here $I[i].m$ refers to the $m$-th objective function value of the $i$-th individual in the set $I$. The complexity of this procedure is governed by the sorting algorithm. In the worst case (when all solutions are in one front), the sorting requires $O(mN \log N)$ computations.
3.3 Crowded comparison operator

The crowded comparison operator \( (<_n) \) guides the selection process at the various stages of the algorithm towards a uniformly spread-out Pareto-optimal front. Let us assume that every individual \( i \) in the population has two attributes.

1. Non-domination rank \( (i_{\text{rank}}) \)
2. Local crowding distance \( (i_{\text{distance}}) \)

We now define a partial order \( <_n \) as:

\[
i <_n j \quad \text{if} \quad (i_{\text{rank}} < j_{\text{rank}}) \quad \text{or} \quad ((i_{\text{rank}} = j_{\text{rank}}) \quad \text{and} \quad (i_{\text{distance}} > j_{\text{distance}}))
\]

That is, between two solutions with differing non-domination ranks we prefer the point with the lower rank. Otherwise, if both the points belong to the same front then we prefer the point which is located in a region with lesser number of points (the size of the cuboid inclosing it is larger).

3.4 The main loop

Initially, a random parent population \( P_0 \) is created. The population is sorted based on the non-domination. Each solution is assigned a fitness equal to its non-domination level (1 is the best level). Thus, minimization of fitness is assumed. Binary tournament selection, recombination, and mutation operators are used to create a child population \( Q_0 \) of size \( N \). From the first generation onward, the procedure is different. The elitism procedure for \( t \geq 1 \) and for a particular generation is shown in the following:

\[
R_t = P_t \cup Q_t \quad \text{combine parent and children population}
\]
\[
\mathcal{F} = \text{fast-nondominated-sort}(R_t) \quad \mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots), \text{all non-dominated fronts of } R_t
\]
\[
P_{t+1} = \emptyset \quad \text{till the parent population is filled}
\]
\[
crowding-distance-assignment(\mathcal{F}_i) \quad \text{calculate crowding distance in } \mathcal{F}_i
\]
\[
P_{t+1} = P_{t+1} \cup \mathcal{F}_i \quad \text{include } i\text{-th non-dominated front in the parent pop}
\]
\[
\text{Sort}(P_{t+1}, <_n) \quad \text{sort in descending order using } <_n
\]
\[ P_{t+1} = P_{t+1}[0 : N] \]
\[ Q_{t+1} = \text{make-new-pop}(P_{t+1}) \]
\[ t = t + 1 \]

choose the first \( N \) elements of \( P_{t+1} \)
use selection, crossover and mutation to create
a new population \( Q_{t+1} \)

First, a combined population \( R_t = P_t \cup Q_t \) is formed. The population \( R_t \) will be of size \( 2N \). Then, the population \( R_t \) is sorted according to non-domination. The new parent population \( P_{t+1} \) is formed by adding solutions from the first front till the size exceeds \( N \). Thereafter, the solutions of the last accepted front are sorted according to \( <_n \) and a total of \( N \) solutions are picked. This is how we construct the population \( P_{t+1} \). This population of size \( N \) is now used for selection, crossover and mutation to create a new population \( Q_{t+1} \) of size \( N \). It is important to note that we use a binary tournament selection operator but the selection criterion is now based on the niched comparison operator \( <_n \).

Let us now look at the complexity of one iteration of the entire algorithm. The basic operations being performed and the worst case complexities associated with are as follows:

1. Non-dominated sort is \( O(MN^2) \),
2. Crowding distance assignment is \( O(MN \log N) \), and
3. Sort on \( <_n \) is \( O(2N \log(2N)) \).

As can be seen, the overall complexity of the above algorithm is \( O(MN^2) \).

The diversity among non-dominated solutions is introduced by using the crowding comparison procedure which is used in the tournament selection and during the population reduction phase. Since solutions compete with their crowding distance (a measure of density of solutions in the neighborhood), no extra niching parameter (such as \( C_{\text{share}} \) needed in the NSGA) is required here. Although the crowding distance is calculated in the objective function space, it can also be implemented in the parameter space, if so desired [1].

4 Results

We compare NSGA-II with PAES on five test problems (minimization of both objectives) [9, 10]:

\[ \text{MOP2:} \begin{cases} f_1(x) = 1 - \exp \left( -\sum_{i=1}^{3} \left( x_i - \frac{1}{\sqrt{3}} \right)^2 \right) -4 \leq x_1, x_2, x_3 \leq 4 \\ f_2(x) = 1 - \exp \left( -\sum_{i=1}^{3} \left( x_i + \frac{1}{\sqrt{3}} \right)^2 \right) \end{cases} \]  
\[ \text{MOP3:} \begin{cases} f_1(x) = \left[ 1 + (A_1 - B_1)^2 + (A_2 - B_2)^2 \right] \\ f_2(x) = \left[ (x + 3)^2 + (y + 1)^2 \right] \end{cases} \]  

where
\[ A_1 = 0.5 \sin x - 2 \cos y + 2 - 1.5 \cos y \]
\[ A_2 = 1.5 \sin x + 2 \cos y - 2 - 0.5 \cos y \]
\[ B_1 = 0.5 \sin x - 2 \cos y + 2 - 1.5 \cos y \]
\[ B_2 = 1.5 \sin x - 2 \cos y + 2 - 0.5 \cos y \]
MOP4: \[
\begin{align*}
  f_1(x) &= \sum_{i=1}^{n-1} \left( -10 \exp \left( -0.2 \sqrt{x_i^2 + x_{i+1}^2} \right) \right) -5 \leq x_1, x_2, x_3 \leq 5 \\
  f_2(x) &= \sum_{i=1}^{n} \left( |x_i|^{0.8} + 5 \sin(x_i^3) \right)
\end{align*}
\]

TC4: \[
\begin{align*}
  f_1(x) &= x_1 \quad 0 \leq x_1 \leq 1 \\
  f_2(x) &= g \left( 1 - \sqrt{\frac{x_1}{g}} \right) -5 \leq x_2, \ldots, x_{10} \leq 5 \\
  \text{where} \quad g(x) &= 91 + \sum_{i=2}^{10} (x_i^2 - 10 \cos(4\pi x_i))
\end{align*}
\]

TC6: \[
\begin{align*}
  f_1(x) &= 1 - \exp(-4x_1) \sin^6(6\pi x_1) \quad 0 \leq x_i \leq 1 \quad i = 1, \ldots, 10 \\
  f_2(x) &= g \left( 1 - (f_1/g)^2 \right) \\
  \text{where} \quad g(x) &= 1 + 9 \left( \sum_{i=2}^{10} x_i/g \right)^{0.25}
\end{align*}
\]

Since the diversity among optimized solutions is an important matter in multi-objective optimization, we devise two measures—one based on the consecutive distances among the solutions of the best non-dominated front in the final population and the other based on the average distance of solutions from the known global Pareto-optimal front. The obtained set of the first non-dominated solutions are compared with a uniform distribution and the deviation is computed as follows:

\[
\Delta = \frac{1}{|F_1|} \sum_{i=1}^{|F_1|} |d_i - \bar{d}|
\]

In order to ensure that this calculation takes into account the spread of solutions in the entire region of the true front, we include the boundary solutions\(^1\) in the non-dominated front \(F_1\). For discrete Pareto-optimal fronts, we calculate a weighted average of the above metric for each of the discrete regions. In the above equation, \(d_i\) is the Euclidean distance between two consecutive solutions in the first non-dominated front of the final population in the objective function space. The parameter \(\bar{d}\) is the average of these distances.

The second metric \(\bar{T}\) measures the convergence property of an algorithm. From each solution in the non-dominated front, its perpendicular distance to the global Pareto-optimal front is calculated by approximating the Pareto-optimal front as a combination of 500 piece-wise linear segments. The average of these perpendicular distances is measured.

For all test problems and with NSGA-II, we use a population of size 100, a crossover probability of 0.8, a mutation probability of \(1/n\) (where \(n\) is the number of variables). We run NSGA-II for 250 generations. The variables are treated as real numbers and the simulated binary crossover (SBX-20) \(^2\) and the real-parameter mutation operator (with distribution index of 500) are used. For the (1+1)-PAES, we have used an archive size of 100 and depth of 4 \(^6\). For SPEA, we have used \(N = 80\) and an external population of size 20. A crossover probability of 0.8 is used. A mutation probability of

---

\(^1\) Boundary solutions are not considered for TC4 and TC6. This is because in these problems the none of the algorithms has converged to the global Pareto-optimal front.
0.01 is used\(^2\) are different. In order to make the comparisons fair, we have used 25,000 iterations in PAES, so that total number of function evaluations in NSGA-II, PAES, and SPEA are the same.

Table 1 shows the deviation from an ideal (uniform) spread (\(\Delta\)) and its variance in 10 independent runs. We show two columns for each test problem. The first column presents the \(\Delta\) value of 10 runs and the second column shows its variance. It is clear from the table that in most test problems NSGA-II has found much smaller \(\Delta\), meaning that NSGA-II is able to find a distribution of solutions closer to a uniform distribution along the non-dominated front. The variance columns suggest that the obtained \(\Delta\) values are consistent in all 10 runs. Table 2 shows the average of convergence measure \(\Upsilon\)

\[\text{Table 1. Comparison of mean and variance of deviation measure } \Delta \text{ obtained using NSGA-II, PAES, and SPEA}\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MOP2</th>
<th>MOP3</th>
<th>MOP4</th>
<th>TC4</th>
<th>TC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II</td>
<td>0.361</td>
<td>0.145</td>
<td>0.87</td>
<td>0.383</td>
<td>0.365</td>
</tr>
<tr>
<td>PAES</td>
<td>1.609</td>
<td>1.341</td>
<td>0.733</td>
<td>1.563</td>
<td>1.195</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.740</td>
<td>0.880</td>
<td>0.167</td>
<td>0.170</td>
<td>0.061</td>
</tr>
</tbody>
</table>

and its standard deviation of 10 runs. It is evident that NSGA-II can come closer to the actual Pareto-optimal front in all problems compared to other two algorithms (except in TC4, where PAES is the best).

\[\text{Table 2. Comparison of distance } \Upsilon \text{ from the true Pareto-optimal front and its standard deviation obtained using NSGA-II, PAES and SPEA}\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MOP2</th>
<th>MOP3</th>
<th>MOP4</th>
<th>TC4</th>
<th>TC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II</td>
<td>0.0019</td>
<td>0.0151</td>
<td>0.0250</td>
<td>4.5128</td>
<td>0.0611</td>
</tr>
<tr>
<td>PAES</td>
<td>0.1704</td>
<td>0.00002</td>
<td>0.0378</td>
<td>4.55386</td>
<td>0.2211</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.1257</td>
<td>0.00004</td>
<td>0.00009</td>
<td>0.00056</td>
<td>0.00045</td>
</tr>
</tbody>
</table>

In order to have a better understanding of how these algorithms are able to spread solutions over the non-dominated front, we present the entire non-dominated front found by NSGA-II and PAES in two of the above five test problems. Figures 2 and 3 show that NSGA-II is able to find a much better distribution than PAES on MOP4. In TC4, converging to the global Pareto-optimal front is a difficult task. PAES’s grid assignment scheme does well in coming closer to the global Pareto-optimal front. With NSGA-II,

---

\(^2\) Mutation probability used in PAES and SPEA is different from NSGA-II, since in NSGA-II a real-parameter mutation operator is used and in PAES and SPEA a bit-wise mutation operator is used.
we find a front with $g = 3.5$ in one out of five different runs. Figure 4 shows the non-dominated solutions obtained using NSGA-II, PAES, and SPEA for TC6. It is clear that the NSGA-II is able to better distribute its population along the obtained front than PAES.

5 Conclusions

In this paper, we have proposed a computationally fast elitist multi-objective evolutionary algorithm based on non-dominated sorting approach. On five difficult test problems
borrowed from the literature, it has been found that the proposed NSGA-II outperforms PAES and SPEA—two other popular multi-objective EAs with the explicit goals of preserving spread on the non-dominated front. With the properties of a fast non-dominated sorting procedure, an elitist strategy, and a parameterless approach, NSGA-II should find increasing attention and applications in the near future.

Acknowledgements

Authors acknowledge the support provided by All India Council for Technical Education, India during the course of this study.

References

Mechanical Component Design for Multiple Objectives
Using Elitist Non-dominated Sorting GA

Kalyanmoy Deb, Amrit Pratap, and Subrajyoti Moitra
Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
{deb, apratap, subhra}@iitk.ac.in
http://www.iitk.ac.in/kangal

Abstract. In this paper, we apply an elitist multi-objective genetic algorithm for solving mechanical component design problems with multiple objectives. Although there exists a number of classical techniques, evolutionary algorithms (EAs) have an edge over the classical methods in that they can find multiple Pareto-optimal solutions in one single simulation run. Recently, we proposed a much improved version of the originally proposed non-dominated sorting GA (we call NSGA-II) in that it is computationally faster, uses an elitist strategy, and it does not require fixing any niching parameter. In this paper, we use NSGA-II to handle constraints by using two implementations. On four mechanical component design problems borrowed from the literature, we show that the NSGA-II can find a much wider spread of solutions than classical methods and the NSGA. The results are encouraging and suggests immediate application of the proposed method to other more complex engineering design problems.

1 Introduction

Most engineering design problems involve multiple and often conflicting objectives. In principle, the presence of conflicting objectives results in a number of optimal solutions, commonly known as Pareto-optimal solutions. Since no single Pareto-optimal solution can be said to be better than another without further considerations, it is desired to find as many such Pareto-optimal solutions as possible. For the last decade or so, a number of multi-objective evolutionary algorithms (MOEAs) have been suggested, mainly because of their ability to find multiple Pareto-optimal solutions in one single simulation run. The non-dominated sorting genetic algorithm (NSGA) was one such algorithm suggested by Srinivas and Deb in the year 1994 [10]. NSGA and a few other algorithms were mainly generational in approach and did not use any elitism. Realizing the need of elitism for faster convergence, researchers have recently introduced elitism in the paradigm of MOEAs [12,6]. Recently, we have proposed a fast elitist NSGA, we called NSGA-II [4], to alleviate three major difficulties of NSGA: (i) large computational effort for non-dominated sorting, (ii) no preservation of elites, and (iii) need to fix a niche parameter. NSGA-II has also possess the lowest possible computational complexity achievable with any non-dominated sorting approach. NSGA-II has been shown to outperform PAES and SPEA—two other elitist EA which are also
explicitly designed for maintaining spread among non-dominated solutions—in terms of spread of trade-off solutions on a number of difficult test problems.

In this paper, we suggest two ways to handle constraints in NSGA-II and investigate the efficiency of constrained NSGA-IIs in finding diverse Pareto-optimal front in a number of engineering design problems. In a two-bar truss design problem, the NSGA-II is compared with a classical $\epsilon$-constraint method. In a gear train design, a spring design, and a welded beam design problem, NSGA-II is compared with the original NSGA and with the best reported single-objective optimizers.

2 Non-dominated Sorting GA (NSGA)

Srinivas and Deb [10] proposed NSGA in 1994 for multi-objective optimization. NSGA is different from a single-objective GA in the way the fitness is assigned to individuals. In order to assign fitness, first the population is sorted according to their non-domination level. Thereafter, all solutions of the better non-dominated fronts are assigned better fitness. In order to preserve diversity among solutions of any front, a niche-preservation technique (sharing function method) is used to degrade raw fitness of solutions which are crowded by other solutions of the same front. On a number of test problems [10] and on a number of engineering design problems [7, 11], NSGA is reported to find a number of non-dominated solutions.

However, NSGAs have been criticized for the following three reasons: (i) the non-dominated sorting approach is $O(N^3)$, where $N$ is the population size, (ii) no elitism approach is used, and (iii) a sharing parameter $\sigma_{\text{share}}$ needs to be fixed. Recently, we have developed an improved NSGA (we called NSGA-II), which eliminates all these difficulties.

3 The Elitist Non-dominated Sorting GA: NSGA-II

The details of NSGA-II algorithm appears in another paper in this volume [4]. Initially, a random parent population $P_0$ is created. The population is sorted based on the non-domination. A special book-keeping procedure is used in order to reduce the computational complexity down to $O(N^2)$. Each solution is assigned a fitness equal to its non-domination level. Binary tournament selection, recombination, and mutation operators are used to create a child population $Q_0$ of size $N$. Thereafter, we use the following algorithm in every generation. First, a combined population $R_t = P_t \cup Q_t$ is formed. This allows parent solutions to be compared with the child population, thereby ensuring elitism. The population $R_t$ is of size $2N$. Then, the population $R_t$ is sorted according to non-domination. The new parent population $P_{t+1}$ is formed by adding solutions from the first front and continuing to other fronts successively till the size exceeds $N$. Thereafter, the solutions of the last accepted front are sorted according to a crowded comparison criterion and a total of $N$ points are picked. Since the diversity among the solutions is important, we define a partial order relation $\prec_n$ as follows:

Definition 1 Solution $i$ is better than solution $j$ in relation $\prec_n$ if ($i_{\text{rank}} < j_{\text{rank}}$) or ($i_{\text{rank}} = j_{\text{rank}}$ and $i_{\text{distance}} > j_{\text{distance}}$).
That is, between two solutions with differing non-dominance ranks we prefer the point with the lower rank. Otherwise, if both the points belong to the same front then we prefer the point which is located in a region with lesser number of points (or with larger crowded distance). This way solutions from less dense regions in the search space are given importance in deciding which solutions to choose from $R_t$. This constructs the population $P_{t+1}$. This population of size $N$ is now used for selection, crossover and mutation to create a new population $Q_{t+1}$ of size $N$. We use a binary tournament selection operator but the selection criterion is now based on the crowded comparison operator $<_n$. The above procedure is continued for a specified number of generations.

It is clear from the above description that NSGA-II uses (i) a faster non-dominated sorting approach, (ii) an elitist strategy, and (ii) no niching parameter.

### 3.1 Constraint handling

We suggest a couple of methods for handling constraints using NSGA-II. In the penalty function approach, total constraint violation is multiplied by a penalty parameter and then added to the each objective function. The penalty parameter is different for different objectives.

The constrained domination approach is more elegant and goes well with NSGA-II. We simply change the definition of domination between two solutions as follows:

**Definition 2** A solution $i$ is said to constrained-dominate a solution $j$, if any of the following conditions is true:

1. Solution $i$ is feasible and solution $j$ is not.
2. Solutions $i$ and $j$ are both infeasible, but solution $i$ has a smaller constraint violation.
3. Solutions $i$ and $j$ are feasible and solution $i$ dominates solution $j$.

This way, feasible solutions constrained-dominate any infeasible solution and two infeasible solutions are compared based on their constraint violations only. However, when two feasible solutions are compared, they are checked based on their domination level.

### 4 Mechanical Component Design Problems

In the following, we discuss four mechanical component design problems which we have studied. For all problems, have kept the GA parameters same: Population size of 100, crossover probability of 1.0, mutation probability of $1/n$ (where $n$ is the number of variables). The SBX operator [3] with a spread factor of 10 and the real-parameter mutation operator [2] with a spread factor of 500 are used. In all simulations, we run for a maximum of 100 generations. All constraints are normalized and a sum of all constraint violations is added to all objective functions. All problems are solved with the penalty function approach. For space restrictions, the constrained domination approach is applied only to the spring and weld design problems.

#### 4.1 Two-bar truss design

This problem was originally studied using the $\varepsilon$-constraint method [8].
The truss (Figure 1) has to carry a certain load without elastic failure. Thus, in addition to the objective of designing the truss for minimum volume, there are additional objectives of minimizing stresses in each of the two members AC and BC. We construct the following two-objective optimization problem for three variables $y$ (vertical distance between B and C in m), $x_1$, and $x_2$ (cross-sectional areas of AC and BC, respectively, in m$^2$):

Minimize $f_1(x) = x_1 \sqrt{16 + y^2} + x_2 \sqrt{1 + y^2}$

Minimize $f_2(x) = \max(\sigma_{AC}, \sigma_{BC})$

subject to $\max(\sigma_{AC}, \sigma_{BC}) \leq 1(10^5)$

$1 \leq y \leq 3$ and $x \geq 0$

(1)

The stresses are calculated as follows:

$$\sigma_{AC} = \frac{20\sqrt{16 + y^2}}{yx_1}$$

$$\sigma_{BC} = \frac{80\sqrt{1 + y^2}}{yx_2}$$

The original study reported only five solutions with the following spread: (0.004445 m$^3$, 89983 kPa) and (0.004833 m$^3$, 83268 kPa). In order to restrict solutions with stress in the above range, we have added an additional constraint of maximum stress being smaller than $1(10^5)$. A penalty parameter of $10^3$ is used to handle this constraint. We apply the proposed method with $0 \leq x_i \leq 0.01$ m$^2$ for $i = 1, 2$. Figure 2 shows the optimized front found using the proposed method. The solutions are spread in the following range: (0.00407 m$^3$, 99755 kPa) and (0.05304 m$^3$, 8439 kPa), which indicates the power of NSGA-II compared to the $\epsilon$-constraint method. The $\epsilon$-constraint method could not find wide variety of solutions in terms of the second objective. If minimization of stress is important, NSGA-II finds a solution with stress as low as 8439 kPa, whereas the $\epsilon$-constraint method has found a solution with minimum stress of 83268 kPa, an order of magnitude higher than that found in NSGA-II. What is also important that all these solutions have been found in just one simulation run of NSGA-II. NSGA-II solutions are better than NSGA solutions, both in terms of closeness to the optimum front and in their spread.

4.2 Gear train design

A compound gear train is to be designed to achieve a specific gear ratio between the driver and driven shafts. The objective of the gear train design is to find the number of teeth in each of the four gears so as to minimize (i) the error between the obtained gear ratio and a required gear ratio of 1/6.931 [5] and (ii) the maximum size of any of the four gears. Since the number of teeth must be integers, all four variables are strictly integers. By denoting the variable vector $x = (x_1, x_2, x_3, x_4) = (T_d, T_b, T_a, T_f)$, we
write the two-objective optimization problem:

\[
\begin{align*}
\text{Minimize } f_1(x) &= \left[ \frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4} \right]^2 \\
\text{Minimize } f_2(x) &= \max(x_1, x_2, x_3, x_4) \\
\text{Subject to } 12 \leq x_1, x_2, x_3, x_4 \leq 60, \\
\text{all } x_i \text{'s are integers.}
\end{align*}
\]

A discrete version of SBX operator is used to make sure that only integer children solutions are created from two integer parents. Since only integer values are allowed, spread factors equal to 2 and 10 for SBX and mutation, respectively, are used. Rigid bounds are used for each variable in order to handle the constraints. Figure 3 shows the obtained optimized solutions. The solutions obtained by the single objective GAs (GeneAS-I and GeneAS-II) [2], by the augmented Lagrangian (AL), and the branch-and-bound (BB) methods for the error minimization are also shown. The figure shows that although the proposed method could not find the best single-objective solutions (GeneAS-I and II), the solution E is close to them and is better than all other single-objective optimizers in terms of the second objective. Both GeneAS solutions are very sensitive to the variables and using a multi-objective optimization algorithms, it may be difficult to find the individual optimum solutions in this problem. Nevertheless, the plot shows the spread in solutions obtained by the proposed method. The solutions marked as ‘E’ and ‘D’ are shown in the following table:

<table>
<thead>
<tr>
<th>Solution</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>Error</th>
<th>Max. Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>12</td>
<td>12</td>
<td>27</td>
<td>37</td>
<td>1.83(10^{-8})</td>
<td>37</td>
</tr>
<tr>
<td>D</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>5.01(10^{-1})</td>
<td>13</td>
</tr>
</tbody>
</table>

Fig. 2. Optimized solutions obtained using the NSGA-II and NSGA for the two-bar truss problem. Five solutions found in [8] are also shown for comparison.
The table shows that a wide variety of optimal solutions have been obtained. NSGA solutions are not as good as NSGA-II solutions.

4.3 Spring design

A helical compression spring needs to be designed for minimum volume and for minimum stress. Three variables are identified: The number of spring coils $N$, the wire diameter $d$, and the mean coil diameter $D$. Of these variables, $N$ is an integer variable, $d$ is a discrete variable having 42 non-equispaced values as given in [5], and $D$ is a real-parameter variable. Denoting the variable vector $x = (x_1, x_2, x_3) = (N, d, D)$, we write the two-objective optimization problem:

Minimize $f_1(x) = 0.25\pi^2 x_2^2 x_3 (x_1 + 2)$
Minimize $f_2(x) = \frac{8Kp_{\text{max}}x_2}{\pi x_3^2}$

Subject to $g_1(x) = l_{\text{max}} - \frac{p_{\text{max}}}{k} - 1.05(x_1 + 2)x_2 \geq 0,$
$g_2(x) = x_2 - d_{\text{min}} \geq 0,$
$g_3(x) = D_{\text{max}} - (x_2 + x_3) \geq 0,$
$g_4(x) = C - 3 \geq 0,$
$g_5(x) = \delta_{pm} - \delta_p \geq 0,$
$g_6(x) = \frac{p_{\text{max}}}{k} - \delta_w \geq 0,$
$g_7(x) = S - \frac{8Kp_{\text{max}}x_2}{\pi x_3^2} \geq 0,$
$g_8(x) = V_{\text{max}} - 0.25\pi^2 x_2^2 x_3 (x_1 + 2) \geq 0$
$x_1$ is integer, $x_2$ is discrete, $x_3$ is continuous.
The parameters used above are as follows:

\[
\begin{align*}
K &= \frac{4C-1}{4C-4} + \frac{0.615x_2}{x_3} \\
P &= 300\text{lb} \\
D_{\text{max}} &= 3\text{in} \\
\delta_p &= \frac{P}{k} \\
P_{\text{max}} &= 1000\text{lb} \\
\delta_w &= 1.25\text{in} \\
\ell_{\text{max}} &= 14\text{in} \\
\delta_{pm} &= 6\text{in} \\
S &= 189\text{ kpsi} \\
C &= D/d
\end{align*}
\]

We add the last two constraints to restrict the stress to be within allowable strength and the volume to be within a pre-specified volume of \(V_{\text{max}} = 30\text{ in}^3\). Discrete version of SBX is used to handle the first two variables and the continuous version of SBX is used to handle the third variable. A penalty parameter of \(10^3\) is used for each normalized constraint.

Figure 4 shows the non-dominated front obtained using the proposed algorithm with penalty function approach and the best solutions obtained using two single-objective optimizers—GeneAS [2] and the branch-and-bound (BB) method [5]. The proposed method is able to find solutions close to these single-objective (volume) optimum and, most importantly, is able to maintain a wide spread of different solutions. The extreme solutions are presented in the following table:

<table>
<thead>
<tr>
<th>Solution</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(f_1)</th>
<th>(f_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. volume</td>
<td>5</td>
<td>0.307</td>
<td>1.619</td>
<td>2.690</td>
<td>1,87,053</td>
</tr>
<tr>
<td>Min. stress</td>
<td>24</td>
<td>0.500</td>
<td>1.865</td>
<td>24.189</td>
<td>61,949</td>
</tr>
</tbody>
</table>

Fig. 4. Optimized solutions obtained using NSGA-II with penalty function approach and NSGA for the spring design problem

Fig. 5. Optimized solutions obtained using NSGA-II with constrained dominated approach

Once again, NSGA-II solutions are better than NSGA solutions. Figure 5 shows the obtained Pareto-optimal solutions using NSGA-II with the constrained domination approach. Although the range of solutions obtained is not as wide as with the penalty function approach, the distribution of solutions is certainly better.
4.4 Welded beam design

A beam needs to be welded on another beam and must carry a certain load $F$ (Figure 6). The overhang portion of the beam has a length of 14 inch and $F = 6,000$ lb force is applied at the end of the beam. The objective of the design is to minimize the cost of fabrication and minimize the end deflection. In the following, we formulate the two-objective optimization problem:

$$
\begin{align*}
\text{Minimize } f_1(x) &= 1.10471h^2\ell + 0.04811tb(14.0 + \ell), \\
\text{Minimize } f_2(x) &= \delta(x), \\
\text{Subject to } g_1(x) &= 13,600 - \tau(x) \geq 0, \\
g_2(x) &= 30,000 - \sigma(x) \geq 0, \\
g_3(x) &= b - h \geq 0, \\
g_4(x) &= P_c(x) - 6,000 \geq 0.
\end{align*}
$$

The deflection term $\delta(x)$ is given as follows:

$$
\delta(x) = \frac{2.1952}{\ell^3b}.
$$

The first constraint makes sure that the shear stress developed at the support location of the beam is smaller than the allowable shear strength of the material (13,600 psi). The second constraint makes sure that normal stress developed at the support location of the beam is smaller than the allowable yield strength of the material (30,000 psi). The third constraint makes sure that thickness of the beam is not smaller than the weld thickness from a practical standpoint. The fourth constraint makes sure that the allowable buckling load (along $t$ direction) of the beam is more than the applied load $F$. The stress and buckling terms are given as follows [9]:

$$
\tau(x) = \sqrt{(\tau')^2 + (\tau'')^2 + (\ell\tau'\tau'')/\sqrt{0.25(\ell^2 + (h + t)^2)}},
\quad \tau' = \frac{6,000}{\sqrt{2h\ell}},
$$

Fig. 6. The welded beam design problem. Minimizations of cost and end deflection are two objectives.
\[
\tau'' = \frac{6,000(14 + 0.5\ell) \sqrt{0.25(\ell^2 + (h + t)^2)}}{2 \{0.707h\ell(\ell^2/12 + 0.25(h + t)^2)\}},
\]

\[
\sigma(x) = \frac{504,000}{t^2b},
\]

\[
P_c(x) = 64,746.022(1 - 0.0282346t)^6.
\]

The variables are initialized in the following range: \(0.125 < h, b < 5.0\) and \(0.1 < \ell, t < 10.0\). Penalty parameters of 100 and 0.1 are used for the first and second objective functions, respectively.

Figure 7 shows the non-dominated solutions obtained using NSGA and NSGA-II with penalty function approach. It is clear that NSGA-II is able to find a wider distribution of solutions than NSGA. NSGA-II found the best cost solution with a cost of 2.79 units, which is close to the best solution (with a cost of 2.38 units) found using a single-objective GA [1]. Figure 8 shows that NSGA-II with constrained domination approach

![Fig. 7. Non-dominated solutions obtained using NSGA-II with penalty function approach and NSGA for the welded beam design problem](image1)

![Fig. 8. Non-dominated solutions obtained using NSGA-II with constrained domination approach](image2)

is able to find a better distribution of solutions than the penalty function approach.

5 Conclusions

In this paper, we have used a modified version of the non-dominated sorting GA (or NSGA) for finding multiple Pareto-optimal solutions in a number of engineering design problems. NSGA-II is different from its predecessor NSGA in a number of ways: (i) it uses a computationally faster non-dominated sorting approach, (ii) it uses an elitist strategy, thereby not allowing good solutions to be deleted by genetic operators, and (iii) it eliminates the need of any niching parameter. The results on four engineering
design problems show that a wide spread of solutions have been obtained. In a two-
member truss design problem, NSGA-II has found many trade-off solutions compared
to only 5 solutions reported in the literature using the e-constraint method. In all prob-
lems, NSGA-II finds a front better and wider than that found by NSGA. The results
of this study are encouraging. The study offers a computationally fast, an elitist, and a
parameter-less multi-objective optimizer which should ease the way to solving complex
engineering multi-objective optimization problems.

Acknowledgements
Authors acknowledge the support provided by All India Council for Technical Educa-
tion, India during the course of this study.

References
Methods in Applied Mechanics and Engineering*.
design based on genetic adaptive search. *Transactions of the ASME: Journal of Mechanical
Design*, 120(2), 162–164.
*Complex Systems*, 9 115–148.
ing genetic algorithm for multi-objective optimization: NSGA-II. *Proceedings of the Sixth
Parallel Problem Solving from Nature Conference*. (Also Technical Report No. 200001. Kan-
pur: Kanpur Genetic Algorithms Laboratory, Indian Institute of Technology, Kanpur, India).
for mixed integer discrete continuous optimization and its applications to mechanical design.
gorithm for multiobjective optimisation. *Proceedings of the 1999 Congress on Evolutionary
industrial Nylon 6 semibatch reactor using genetic algorithms. *Journal of Applied Polymer
\(\epsilon\)-inequality constraint method for multiple objectives decision making. *ASME Journal of
Pareto-optimal broad band microwave absorbers. *IEEE Transactions on Electromagnetic
Compatibility*, 38(4).
12. Zitzler, E. (1999). Evolutionary algorithms for multiobjective optimization: Methods and
applications. Doctoral thesis ETH NO. 13398, Zurich: Swiss Federal Institute of Technology
(ETH), Aachen, Germany: Shaker Verlag.
On the Assessment of Multiobjective Approaches to the Adaptive Distributed Database Management Problem

Joshua D. Knowles, David W. Corne and Martin J. Oates

School of Computer Science, Cybernetics and Electronic Engineering
University of Reading, Reading RG6 6AY, UK.
Tel: +44 (0) 118 931 8983; Fax: +44 (0) 118 975 1774
Email: j.d.knowles,d.w.corne@reading.ac.uk

Abstract. In this paper we assess the performance of three modern multiobjective evolutionary algorithms on a real-world optimization problem related to the management of distributed databases. The algorithms assessed are the Strength Pareto Evolutionary Algorithm (SPEA), the Pareto Archived Evolution Strategy (PAES), and M-PAES, which is a Memetic Algorithm based variant of PAES. The performance of these algorithms is compared using two distinct and sophisticated multiobjective-performance comparison techniques, and extensions to these comparison techniques are proposed. The information provided by the different performance assessment techniques is compared, and we find that, to some extent, the ranking of algorithm performance alters according to the comparison metric; however, it is possible to understand these differences in terms of the complex nature of multiobjective comparisons.

1 Introduction

In real-world applications, obtaining the complete set of Pareto optimal solutions for a multiobjective problem may be impossible to attain, and instead we seek a ‘good’ approximation to this set, which may in fact contain no true Pareto optima. Measuring the quality of approximations such as these is a problematic area but a variety of methods have been put forward (e.g. [1, 3, 5, 8–12]), and most have been put to use in establishing differences in algorithm performance. However, further investigation of these methods is much needed because, as yet, no single method can determine and present in a concise form, unequivocal information about both the statistical and geometric properties of an algorithm’s approximations to the Pareto set.

In this paper, a real-world telecommunications problem is considered under two slightly different multiobjective formulations. In both cases, we wish to make no a priori judgments about the preferences of the telecommunications company with regard to how they might view the QoS delivered to customers as a function of the objectives considered. Three modern, and competitively-matched multiobjective optimization algorithms are to be tested on a suite of
different versions of the problem representing a number of different scenarios. Thus, the aim is to judge which algorithm obtains the ‘best’ sets of solutions on this suite of problems, using only the weak Pareto assumption. Rather than just viewing the results from the single perspective of just one statistical method of judging performance, we instead consider and extend several methods, and observe the differences and similarities in the information they provide.

The three algorithms we consider are all modern solutions to the problem of Pareto optimization. All three of them are elitist approaches, and they each maintain a list of (a limited number of) the nondominated solutions that they find. The first algorithm, the Strength Pareto Evolutionary Algorithm (SPEA) of Zitzler and Thiele [11,12], has been widely tested. The second algorithm is (1+1)-PAES, originally put forward by us as a simple baseline algorithm [2]. Its performance has also been tested elsewhere on several problems [3]. Third, is an extension of the basic (1+1)-PAES algorithm; a memetic approach called M-PAES. Its performance was shown to be competitive with SPEA on a suite of multiobjective 0/1 knapsack problems [4].

The remainder of this paper is organised as follows. In the next section, the Adaptive Distributed Database Management Problem (ADDMP) is defined in general, and then the particular choice of scenarios tackled here is explained. Section 3 provides details of the experimental methods used, including algorithm parameter settings, and the performance assessment techniques employed. In this section, extensions to previous methods of assessing performance are included. The results of the experiments are presented and discussed in Section 4. Finally, in the concluding section, a summary of the findings, and ideas for future work are given.

2 The Adaptive Distributed Database Management Problem

The Adaptive Distributed Database Management problem (ADDMP) is a problem faced by distributed database service providers (DDSPs), such as video-on-demand, genome databanks, and so forth. Oates and Corne [7] gives a detailed description, and C source code for the evaluation function can be found via the first author’s website\(^1\). Here, we provide basic details of the ADDMP, aimed at conveying an understanding of its multiobjective nature.

A DDSP needs to regularly ensure that database users (clients) are receiving adequate quality of service (QoS). Indeed, clients’ subscription to the database may involve guarantees from the DDSP of distinct levels of QoS, perhaps varying with subscription cost. A key factor in QoS is the delay (or response time) experienced by a client for a typical database query. In maximizing QoS, the DDSP aims to minimize the delay for each client. However, since copies of all or parts of the database exist on several perhaps globally distributed servers, this minimization must occur in the context of load balancing. That is, we may

\(^1\) http://www.reading.ac.uk/~ssr97jdk
be able to minimize the delays experienced by certain clients by routing their queries to the fastest server which contains the required data; however, the extra load on this server will degrade the delays. So, the optimal solution will involve a careful balancing of clients across servers.

The ADDMP is hence the problem of finding the best client/server connection configuration, given a particular scenario which specifies details of the underlying communications network, server speeds, and access rates for each client. What counts as ‘best’ depends on many things, but a single-objective QoS measure will typically involve combining the worst client delay with the mean or median delays. However, such QoS measures are growing increasingly inadequate as distributed database service provision becomes more widespread and complex as regards the range of service guarantees on offer. For example, consider two potential solutions to a 5-client ADDMP in which the vectors of client delays (in milliseconds) are, respectively: Solution 1 (155, 130, 140, 140, 140), Solution 2 (350, 80, 90, 90, 90).

In a single-objective approach, which of solution 1 or 2 is preferred depends very much on the relative weightings given to the worst and mean (or median) components. It is hence complex, and perhaps impossible, to derive ‘correct’ relative weightings for these components, especially considering the widely different kinds of ADDMP scenarios which exist.

A multiobjective approach therefore seems more sensible and flexible. Client 1, for example, may have paid for a QoS guarantee which indicates that their delay will always be below 200 ms. Client 2, on the other hand, may have been given a guarantee that their delay would be always within 20% of the median delay level at any snapshot in time. With varied sets of factors like this, the task of an optimizer addressing an ADDMP would be to quickly produce a good and diverse spread of solution configurations, leaving it to a later decision process to then choose from these on the basis of the various QoS guarantees in operation for the clients currently using the service.

The problem we address in this paper is therefore that of quickly providing a good set of diverse ADDMP configurations, from which a second decision-making process can then choose the best according to prevailing QoS issues.

ADDMP Variants

ADDMP instances can occur in great variety. The numbers of clients and servers can range typically between 2 and 20, and the number of clients between 10 and several thousand. Access patterns can vary equally dramatically. E.g., access to share price and similar financial databases may be very frequent with constantly changing global activity, and hence re-optimization of client/server access configurations may need to occur every few minutes. In other scenarios, involving a small number of clients, re-optimization may only need to occur every few hours.

A key part of the datafile defining an ADDMP instance is an array of client access rates. Over time, we can expect these to vary, leading to different ADDMP instances requiring re-optimization to redistribute the load according to current usage.
In this paper we look at scenarios involving 10, 20, and 40 client/servers, and in each case we consider 5 separate problems which reflect possible changes in access patterns over time. Thereby, we are comparing the quality of SPEA, PAES, and M-PAES on the ADDMP over a wide but representative range of potential instances. We are interested particularly in ADDMPs which need constant, and hence fast, re-optimization. I.e., results must arrive quickly. Hence, in increasing order of the problem sizes, the maximum allowed number of evaluations is 500, 2000 and 5000.

We consider both 2-objective and 3-objective versions of each problem. In the 2-objective version, the objectives are the worst delay figure and the median delay figure. In the 3-objective version, the objectives are the 90% delay figure (i.e. 90% of clients will have a better delay figure than this), the 80% delay figure, and the median figure.

3 Parameter control and performance assessment

The problems of comparing and assessing the performance of multiobjective optimizers fall broadly into two categories; controlling the parameters of the various algorithms, and actually measuring the performance. Space restrictions preclude proper discussion relating how parameters were controlled, but the ranges of values used are shown in Fig. 1.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>pc</th>
<th>crossover type</th>
<th>pm</th>
<th>mutate type</th>
<th>internal pop.</th>
<th>external pop.</th>
<th>l_fails</th>
<th>L_opt</th>
<th>cr_trials</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-PAES</td>
<td>N/A</td>
<td>1/L</td>
<td>flip</td>
<td>5–20</td>
<td>80–95</td>
<td>1–20</td>
<td>2–100</td>
<td>N/A</td>
<td>5</td>
<td>5 / 3</td>
</tr>
<tr>
<td>SPEA</td>
<td>0.8</td>
<td>uniform</td>
<td>1/L</td>
<td>flip</td>
<td>5–20</td>
<td>80–95</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1+1-PAES</td>
<td>N/A</td>
<td>N/A</td>
<td>1/L</td>
<td>flip</td>
<td>1</td>
<td>99</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Fig. 1. Parameter settings for the three algorithms. Bold face indicates a fixed value in all experiments. The ranges of values used for the free parameters is shown. These were investigated on an ad hoc basis to provide 'best' performance. The two values shown for l, the number of bisection levels in the adaptive grid algorithm [4], refer to the values for the two and three objective problems, respectively. Please refer to [4] for a description of the parameters used in M-PAES.

The quality of a set of nondominated vectors can be assessed in several distinctly different ways, leading to many possible metrics. Our preferences (at least in this study) are towards methods that do not require knowledge of the true Pareto front, such as Generational Distance or Error Ratio [9], because these are often not available. Nor do we consider cardinal measures, e.g. the Coverage metric [11], and again Error Ratio, to be very satisfactory because they can give extremely misleading information (see [3]). Instead, metrics based on measuring the position of the discovered attainment surface (the boundary between dominated and nondominated regions) [1], or the size of the dominated region itself [5,11], seem preferable because they conform more closely to the ideals of Pareto optimization. The specific metrics we use are described next.
Fig. 2. A plot of the objective space of a 2-objective minimization problem. There are two attainment surfaces defined by two sets, A and B, of solution vectors. The combined dominated region $S(A \cup B)$ can be calculated easily. By subtracting the dominated region of A, $S(A)$, from $S(A \cup B)$, the combined size of the regions dominated by B but not A (e.g. dark shaded region), $S_B \setminus A$, can be calculated. Similarly, the combined size of regions dominated by A but not B (e.g. light shaded region), $S_A \setminus B$, can be calculated by subtracting $S(B)$ from $S(A \cup B)$.

In [1], a means of combining the information from several runs of an algorithm was put forward. To date, this has been our favoured means of performance assessment, and we have extended the approach to cater for comparison of multiple algorithms. As a whole, we refer to the method as attainment surface sampling. In brief, the method works by taking samples of the attainment surfaces defined by the discovered solution sets from independent runs of an algorithm or algorithms. The samples allow us to calculate and plot the best, median and worst attainment surfaces from a collection of runs. This information is not easy to analyse, however, except when there are only two objectives, so that surface plots are easy to interpret. Fortunately, statistical comparison of algorithms is also strongly catered for by these techniques; because each sampling of the attainment surfaces provides a univariate distribution, standard non-parametric statistics can be applied.

We use two statistical methods for comparing algorithms, one for comparing a pair of algorithms only, and another for comparing $n$ algorithms at once. When comparing algorithms we must have a collection of solution sets from independent runs of each algorithm. If we adopt the phrase ‘A beats B’ at a particular sampled region of the Pareto front if in that region the distribution of attainment surfaces from multiple runs of algorithm A is better than that from multiple runs of algorithm B at a statistically significant level, according to some non-parametric test, then we can define our two measures: In the case of comparing just two algorithms denoted by A and B, the statistics returned
are a pair of numbers, \((a, b)\), where \(a\) indicates the percentage of the front on which \(A\) beats \(B\), and where \(b\) indicates the percentage of the front on which \(B\) beats \(A\). When comparing \(n > 2\) algorithms, \(n(n - 1)\) pair-wise comparisons are made as above, but with the sampling of the surfaces always carried out on the full extent of the combined discovered Pareto front. Two statistics can then be defined for each algorithm. The algorithm’s \textit{unbeaten} statistic is the percentage of the front on which no other algorithm beats it, and it’s \textit{beats all} statistic is the percentage of the front on which it beats all of the other algorithms. For the univariate statistical tests that the above metrics depend upon, we use the Mann-Whitney \(U\) test [6] at the 95% confidence level. A fuller explanation of our implementation and criticisms of these techniques are given in [3].

Our other measures are based on the size of the dominated region, \(S(C_\alpha)\), defined as the union of regions dominated by each individual solution vector in the set, \(C_\alpha\), of nondominated vectors found during optimization run \(\alpha\), within some bounding rectangular polytope, as in [5]. However, we do not size the bounding polytope based on the location of an ideal point, but just ensure that it contains all of the nondominated vectors. (Note: Both methods of normalization induce rather arbitrary scalings of the objectives that are hard to fully justify in a multiobjective context.) We propose using the size of the dominated region in three new ways, with the aim of combining the information contained in several runs of an algorithm, or to provide more useful comparative information. In the following, we also consider a collection \(C\) of \(n\) sets \(C_\alpha\), \(\alpha \in \{1, \ldots, n\}\), of the points found in one optimization run of an algorithm.

Our first measure is of the total discovered region, found over a set of \(n\) runs:

\[
S_T = S\left(\bigcup_{\alpha=1}^{n} C_\alpha\right)
\]

The second measure combines the \(n\) runs by finding the size of the median attainment surface:

\[
S_m = S(MS(C))
\]

where \(MS(C)\) is the median attainment surface of the collection of solution sets, \(C\), obtained using attainment surface sampling. Finally, we propose a measure for directly comparing two collections \(A\) and \(B\) of approximations, each obtained from \(n\) independent runs. The measure is based on the notion of the coverage difference of two sets [10] defined by:

\[
S_{A \setminus B} = S(A \cup B) - S(B)
\]

The meaning of the coverage difference is illustrated in Figure 2. This measure is useful when comparing two algorithms. Taking the two complements together, the degree to which one set of points covers the other can be inferred. For example, if the pair of complements \(S_{A \setminus B}\) and \(S_{B \setminus A}\) have the values 0 and 0.05 respectively then one can say that the points in \(B\) completely cover the points in \(A\), and also that the points in \(A\) dominate 95% of the region dominated by \(B\). Now, for a collection of runs from two algorithms, \(A\) and \(B\), we propose calculating the median value of the coverage differences.
4 Results

In the following, all results are based on thirty independent runs of each algorithm on each problem instance. The first set of results (Fig. 3) was obtained using attainment surface sampling and our $n$-algorithm comparison metric. These

<table>
<thead>
<tr>
<th>ADDMP instance</th>
<th>statistic</th>
<th>2-objective</th>
<th>3-objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>M-PAES</td>
<td>SPEA</td>
</tr>
<tr>
<td>10-1 unbeaten</td>
<td>93.8</td>
<td>83.1</td>
<td>99.2</td>
</tr>
<tr>
<td>10-2 unbeaten</td>
<td>100</td>
<td>99.8</td>
<td>94.9</td>
</tr>
<tr>
<td>10-3 unbeaten</td>
<td>100</td>
<td>98.7</td>
<td>95.2</td>
</tr>
<tr>
<td>10-4 unbeaten</td>
<td>100</td>
<td>98.9</td>
<td>100</td>
</tr>
<tr>
<td>10-5 unbeaten</td>
<td>99.8</td>
<td>49.7</td>
<td>100</td>
</tr>
<tr>
<td>20-1 unbeaten</td>
<td>100</td>
<td>48.7</td>
<td>100</td>
</tr>
<tr>
<td>20-2 unbeaten</td>
<td>65.0</td>
<td>67.2</td>
<td>100</td>
</tr>
<tr>
<td>20-3 unbeaten</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20-4 unbeaten</td>
<td>52.1</td>
<td>0.5</td>
<td>100</td>
</tr>
<tr>
<td>20-5 unbeaten</td>
<td>49.9</td>
<td>50.0</td>
<td>100</td>
</tr>
<tr>
<td>40-1 unbeaten</td>
<td>92.0</td>
<td>15.6</td>
<td>99.9</td>
</tr>
<tr>
<td>40-2 unbeaten</td>
<td>69.3</td>
<td>7.9</td>
<td>93.4</td>
</tr>
<tr>
<td>40-3 unbeaten</td>
<td>4.6</td>
<td>0</td>
<td>30.7</td>
</tr>
<tr>
<td>40-4 unbeaten</td>
<td>68.7</td>
<td>10.3</td>
<td>93.8</td>
</tr>
<tr>
<td>40-5 unbeaten</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 3. The *unbeaten* and *beats all* statistics for the combined space inhabited by the solutions found. Two forms of the problem were investigated: The 2-objective case, where the median response time and the worst response time are minimized. And the 3-objective case, where the median response time, the response time bettered by 80% of requests, and the response time bettered by 90% of requests are minimized. The different problems are labelled by the number of nodes and the number of the scenario. E.g. The third scenario of the twenty node problem is labelled 20-3.

results, taken on their own, seem to indicate that (1+1)-PAES is consistently difficult to beat, whereas SPEA and M-PAES are closely matched but not as
consistently good as PAES. Results obtained from our other metrics, a key subset of which are presented below, are not always in agreement, however.

<table>
<thead>
<tr>
<th>ADDMP instance</th>
<th>2-objective</th>
<th>3-objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M-PAES</td>
<td>SPEA</td>
</tr>
<tr>
<td>10-3</td>
<td>0.33803</td>
<td>0.33776</td>
</tr>
<tr>
<td>20-3</td>
<td>0.33580</td>
<td>0.34616</td>
</tr>
<tr>
<td>40-3</td>
<td>0.27758</td>
<td>0.30131</td>
</tr>
</tbody>
</table>

Fig. 4. The values of the total dominated region, \( S_{r} \), for three problem instances.

<table>
<thead>
<tr>
<th>ADDMP instance</th>
<th>2-objective</th>
<th>3-objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M-PAES</td>
<td>SPEA</td>
</tr>
<tr>
<td>10-3</td>
<td>0.1551</td>
<td>0.1590</td>
</tr>
<tr>
<td>40-1</td>
<td>0.342969</td>
<td>0.345659</td>
</tr>
</tbody>
</table>

Fig. 5. The values of the Median Attainment Region, \( S_{\mu} \), for two problem instances.

<table>
<thead>
<tr>
<th>ADDMP instance</th>
<th>obj-acceptives</th>
<th>( A = ) M-PAES 1+1-PAES</th>
<th>( B = ) SPEA 1+1-PAES</th>
<th>( A = ) SPEA M-PAES</th>
<th>( B = ) M-PAES 1+1-PAES</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc10-1</td>
<td>2</td>
<td>0.0327</td>
<td>0.0124</td>
<td>0.0161</td>
<td>0.0016</td>
</tr>
<tr>
<td>sc10-1</td>
<td>3</td>
<td>0.0678</td>
<td>0.0</td>
<td>0.0161</td>
<td>0.0016</td>
</tr>
<tr>
<td>sc20-4</td>
<td>2</td>
<td>0.0187</td>
<td>0.0161</td>
<td>0.0286</td>
<td>0.0153</td>
</tr>
<tr>
<td>sc20-4</td>
<td>3</td>
<td>0.0026</td>
<td>0.0</td>
<td>0.0239</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Fig. 6. The median values of the coverage differences of alternate pairs of algorithms on two problem instances.

The results of calculating \( S_{r} \) for three of the problems are shown in Fig. 4. First, notice that the values for the three algorithms are very close and that a large number of figures are significant. Nonetheless, the results here still provide extra information about the distribution of solutions found by the algorithms over multiple runs. On the first two of the 3-objective cases, two algorithms generate exactly the same total dominated region. In all probability this must indicate that the total set of non-dominated solutions found in each case is exactly the same. Interestingly, the total dominated region measure favours SPEA over M-PAES, although (1+1)-PAES is superior overall. Using our unbeaten and beats all statistics, M-PAES is ranked ahead of SPEA, with (1+1)-PAES in first place. This disagreement must indicate that SPEA tends to generate different solutions on different runs more often than M-PAES. Still, the total dominated
region preserves the position of (1+1)-PAES as the most consistent algorithm, it winning on three of the six measurements, and being beaten only once.

In the next set of results presented (Fig. 5) the median attainment surface is first calculated for each algorithm. The size of the region dominated is then measured. Once again, using this measure alone could lead to different conclusions than if using it in conjunction with other measures. For example, on the two-objective version of problem 10-3, the rank order reported by this measure $S_\mu$, the $S_r$ measure, and our unbeaten statistic are all different. Clearly the algorithms perform at very similar levels on this problem, but which is best? On the three objective version of the same problem, M-PAES and (1+1)-PAES are very similar with regard to $S_\mu$, whereas M-PAES is a poor third when considering the whole distribution of attainment surfaces, using the Mann-Whitney $U$ test.

Finally, the coverage differences, $S_{A\setminus B}$ and $S_{B\setminus A}$ were calculated for pairs of the algorithms, for each of the thirty runs. The median values of these differences are presented in Fig. 6. These results exhibit a high degree of agreement with the equivalent results in Fig. 3. According to the former, the attainment surface generated by (1+1)-PAES completely dominates the attainment surface found by the other two algorithms on at least 50% of the runs, on the three-objective versions of the two problems.

5 Conclusion

In this paper we have compared the performance of three multiobjective algorithms with respect to a suite of real-world problems related to the management of distributed databases. Several test metrics were employed to measure and compare algorithm performance over collections of solution sets found from several (30) runs. Three extensions to a metric based on the size of the dominated space were presented and used. The results show that with such closely-matched algorithms, it is a very difficult matter to select the one that performs best. In fact, from the results presented, (1+1)-PAES seems to be the best performer all-round. However, results given by the different metrics indicated that the rank order of algorithms was certainly not independent of the test metric. This shows that it is most important to use a number of different metrics, when comparing algorithms. Then, where a rank order is inconsistent across different metrics, the extra information provided can help to understand exactly in what way one algorithm's approximations differ from another's. Actually, more detailed analysis of the full results reveals that SPEA was most consistent at providing an even distribution of solutions along the front, whereas (1+1)-PAES more often found very strong compromise solutions.

The metrics we have proposed could also be extended further. For example, testing the coverage differences over a collection of runs, could be carried out by using a statistical test such as Kolmogorov-Smirnov, rather than the median. In addition, we are currently developing a method based on the $S$ measure, that can calculate and present information about the individual regions where one attainment surface dominates another.
Acknowledgments

The authors thank the anonymous reviewers for their helpful comments. The first author would also like to express his gratitude to BT Labs plc. for sponsorship of his PhD.

References


A Hierarchical Genetic Algorithm
Using Multiple Models for Optimization

Mourad Sefrioui and Jacques Périaux

Abstract. This article presents both the theoretical basis and some experimental results on Hierarchical Genetic Algorithms (HGAs). HGAs are explained in details, along with the advantages conferred by their multi-layered hierarchical topology. This topology is an excellent compromise to the classical exploration/exploitation dilemma. Another feature is the introduction of multiple models for optimization problems, within the frame of an HGA. We show that with such an architecture, it is possible to use a mix of simple models that are very fast and more complex models (with slower solvers), and still achieve the same quality as that obtained with only complex models. The different concepts presented in this paper are then illustrated via experiments on a Computational Fluid Dynamics problem, namely a nozzle reconstruction. The overall results are that a Hierarchical Genetic Algorithm using multiple models can achieve the same quality of results as that of a classic GA using only a complex model, but up to three times faster.

1 Introduction

This work attempts to find a remedy to two major concerns of any GA based optimization: the dilemma between exploration and exploitation, and the computational cost associated to complex models. Section 2 presents a possible answer to both problems, by introducing the concepts behind Hierarchical GAs and the use of multiple models. HGAs offer a good alternative to the exploration/exploitation quandary by using a hierarchical topology that makes it possible to have different layers performing different tasks. As for the other dilemma, HGAs with multiple models propose a way out, by using different models depending on the position in the hierarchy.

This approach is then compared to a similar approach, iiGA, and we will try to outline the common points and differences of the two methods.

The proposed strategies are then tested in section 3 by solving an optimization problem coming from Computational Fluid Dynamics.

2 Hierarchical Genetic Algorithms

2.1 Parallel Genetic Algorithms

Parallel Genetic Algorithms (PGAs) are a particular instance of GAs, based on the notion of sub-population [1–3]. Instead of a single large population, PGAs
use a network of interconnected sub-populations, thus defining a paradigm called the *Island model* [4, 5]. Figure 1.a shows such a model, with the greyed area corresponding to a node (or sub-population) and its neighborhood.

The idea is that each of these sub-populations will evolve *independently* for a given period of time (or epoch). After each epoch, a period of migration and information exchange takes place, before the isolation resumes. Figure 1.b shows that this sort of approach can lead different sub-populations to explore different regions of the search space during the isolation stage. Yet, the most promising solutions are shared by the whole set of sub-populations, since those solutions are sent to the neighbors.

![Fig. 1. Parallel GAs](image)

2.2 Hierarchical Topology

HGAs are a particular approach we developed, based on PGAs [6]. They use a hierarchical topology for the layout of the sub-populations. In the following, we use a particular architecture, with a binary tree spawning 3 levels (fig. 2).

![Fig. 2. Hierarchical topology](image)
The main advantage lies in the interaction of the 3 layers: the solutions go up and down the layers, and the best ones keep going up till they get refined.

Besides this circulation of solutions up and down, each node is handled by a different GA code, and has specific parameters that can be tuned. The nodes of each layer may have a different purpose, defined by their associated GA:

1. Top layer: refining solutions, by tuning the GA in a way that makes mutation take very small steps (since it is a real-coded GA using a non-uniform mutation, the mutation span \(^1\) can be set to be very small).
2. Intermediate layer: compromise between exploitation and exploration.
3. Bottom layer: full-on exploration. That means that the GA can make big leaps in the search space via a large mutation span.

A few questions arise when it comes to migration:

1. Which node exchanges with which node
2. How many solutions should be sent from one node to the other
3. Which solutions should be sent

The choice we made for question 1 is the hierarchical topology of figure 2. As for question 2, we chose to send \(\frac{1}{3}\) of the population, which means that each sub-population gets to replace the worst third of its population by new arrivals. For question 3, we adopted a strategy consisting of sending up the best solutions and sending down a random sampling of solutions.

**Characteristics of the GA** In the following, each sub-population is of size 10, and is evolved by a real-coded GA. Elitism and tournament selection are used, with a tournament size of 3. We used a double point crossover with \(P_c = 0.95\). As for mutation, we used a non-uniform mutation scheme [8] to determine the span of the mutation. For the mutation rate, we applied **distance-dependent mutation** as a way to avoid premature convergence [9]. Distance-dependent mutation is a technique we developed to maintain diversity in small populations. Instead of a fixed mutation rate, each offspring has its mutation rate computed after each mating.

### 2.3 Multiple Models

The fact that each sub-population is run by an independent GA is instrumental in the multiple models approach. It means that each GA may have a different fitness function. And from these premises, it is easy to go a step further and associate each GA with a different model for a given optimization problem. To take the full benefit of the hierarchical structure, we developed a strategy where the top layer uses a very precise model – meaning a time-consuming solver. But at the same time, the sub-populations of the bottom layer need not yield a very

\(^1\) the mutation span is the size of the mutation, not the mutation rate
precise result, as their main goal is to explore the search space. That means that they can make do with simple models and much fast solvers.

Figure 3 shows how that can be done. All the bottom layer nodes use Model 3 to compute the fitness function. Model 3 should be a relatively coarse model, and the associated solver should be as fast as possible. Of course, these solutions are an approximation, but the hierarchical topology makes it work, because these solutions are sent up to the intermediate layer where they are re-evaluated using Model 2, a reasonably precise model. This re-evaluation is essential, as it gives a more accurate idea of the actual quality of the solution. This process is then applied to the intermediate layer, and solutions are sent up to the top layer where they are re-evaluated with Model 1, a precise model that gives an accurate value for the fitness function.

The following algorithm details a HGA with multiple models:

\begin{verbatim}
for all sub-populations do {Initialization}
    Randomly generate solutions
    Layer 1 uses Model 1 to evaluate fitness
    Layer 2 uses Model 2 to evaluate fitness
    Layer 3 uses Model 3 to evaluate fitness
end for

repeat
    for \( i = 1 \) to \( Epoch \) do {Isolation phase}
        for all sub-populations do
            Evolve 1 generation
        end for
    end for

    Start migration {Migration phase}
    Layer 1 : gets best solutions from Layer 2
              reevaluates them using Model 1.
    Layer 2 : gets random solutions from Layer 1
              gets best solutions from Layer 3
\end{verbatim}
reevaluates them using Model 2.
Layer 3: gets random solutions from Layer 2
reevaluates them using Model 3.

until Stopping criterion is met

2.4 iiGA: Injection Island Genetic Algorithms

In parallel to our work [10,6], several applications have been developed independently with a somewhat similar technique called iiGA, for Injection Island GA [11,12]. We recently found out about this method, and the overall structure is quite close conceptually. However, there are a few differences:

- iiGA relies on a binary coded GA, whereas HGA is based on a real-coded GA. This difference is quite significant, since HGA does not change the parameterization from one level to another (the whole parameterization vector is passed between subpopulations), thus maintaining the same precision regardless of the level. On the other hand, iiGA needs to change the representation for a solution when it has to be refined (for instance by increasing the length of the chromosome for higher precision).
- In iiGA, solutions can only go up and be further refined. In HGA, we have seen that this process is actually symmetrical, and that solutions are sent either way. This is actually much closer to what happens in computational analysis, for instance in multi-grid methods.
- iiGA uses large sub-populations (up to 300 individuals), whereas the different techniques we evolved for real-coded GAs allow us to use sub-populations of size 20 or less and yet avoid premature convergence.
- In iiGA, the exploration effort is mainly enforced by tuning up the crossover rate and using large populations. In HGA, the exploration is enforced by enlarging the span of the mutation.

In the future, it could be interesting to find some benchmark problem that would make it possible to compare the performances of the two approaches.

3 Application: Nozzle reconstruction

3.1 Problem definition

Fig. 4. Converging-diverging target nozzle
The strategy we just presented is exploited in the field of Computational Fluid Dynamics (CFD) shape optimization. The optimization tasks consists in reconstructing the shape of a converging-diverging nozzle for a transonic flow involving a shock. The principle of the reconstruction is quite simple: we first take a target nozzle and generate the corresponding pressure distribution with a CFD code. Then, we start from a random nozzle and try to match its pressure distribution to the target distribution. When both distributions perfectly match, we consider that the reconstruction has been achieved.

To be a bit more explicit, for a given target nozzle (fig. 4), the corresponding Mach number distribution is computed using a quasi-steady one-dimensional approximation for the flow. The equations are solved by a time marching technique using a CUSP scheme with an iterative solver [13, 14].

Figure 5 shows the different Mach number distributions obtained for that nozzle, for different grid sizes. The first distribution is generated using 100 points along the nozzle, and is the most precise model. The second distribution corresponds to a grid size of 50, whereas the third one is for a grid size of 25.

Modifying the size of the grid changes the quality of the Mach number distribution (and subsequently the position of the shock). At the same time, it also greatly changes the CPU time needed by the CFD solver to converge. The following table shows the CPU times per iteration, depending on the resolution (res) of the CFD solver and the size of the grid:

<table>
<thead>
<tr>
<th>res</th>
<th>Grid size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>100</td>
<td>1.75 s</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>50</td>
<td>2.47 s</td>
</tr>
</tbody>
</table>

With a resolution of $10^{-2}$ for the solver, the CPU time is over 15 times larger for a grid size of 100 compared to a grid size of 25.
3.2 Single Population GA

The first optimization we present is done by a real-coded GA, with a single population of size 20. The shape of the nozzle is defined by a Bézier curve of order 5. The curve is defined by 5 control points $P^1, \ldots, P^5$. The x coordinates of the $P^i$ are all distributed evenly in $[-0.33, 1]$. The variables of the optimization are the y coordinates of the middle control points, $P^2_y, \ldots, P^4_y$. For the fitness, we use 100 points along the nozzle (i.e. Model 1). The fitness function is: $f = \sum_{i=1}^{100} (M_i - M_i^{\text{target}})^2$ where $M_i$ is the candidate nozzle Mach number distribution and $M_i^{\text{target}}$ the target Mach number distribution for point $i$.

![Fig. 6. Convergence](image)

(a) GA Single Model  (b) Hierarchical GA Single Model

Figure 6.a shows the evolution of the best solution, on a logarithmic scale, for a required precision $Pres = 10^{-5}$.

<table>
<thead>
<tr>
<th>$Pres$</th>
<th>Evaluations</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-3}$</td>
<td>1080</td>
<td>2595 s (43mn)</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>2580</td>
<td>6460 s (1h47)</td>
</tr>
</tbody>
</table>

**Table 1. Single Population GA**

Table 1 shows the performance of single GA for 2 different stopping criteria. *Evaluations* corresponds to the average number of evaluations over 50 runs. *CPU time* denotes the average CPU time before convergence over 50 runs. In this table, as in the next ones, the CPU time is based on a single Pentium 366 processor. *Pres* shows the value of the stopping criteria, by defining the required precision for the fitness function before convergence.

3.3 Hierarchical GA, Single Model

Table 2 shows the performance of a HGA using a single model (sub-population size = 10, $Epoch = 20$). The values of *Evaluations* and *CPU time* are averaged
over 50 runs. The HGA uses Model 1, i.e. the most precise model with a grid of size 100. All sub-populations of all layers use this model. It is important to note that for the sake of comparison, only one processor is used even for the hierarchical topology and the parallelism is actually simulated by running sequentially each sub-population.

<table>
<thead>
<tr>
<th>Pres</th>
<th>Evaluations</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-3})</td>
<td>1485</td>
<td>3309 s (55mn)</td>
</tr>
<tr>
<td>(10^{-5})</td>
<td>2300</td>
<td>5200 s (1h28)</td>
</tr>
</tbody>
</table>

**Table 2. Hierarchical GA, Single Model**

Figure 6.b shows the evolution of the best solutions for all sub-populations with HGA, for \(\text{Pres} = 10^{-5}\).

Figure 7.a shows that after convergence, the Mach number distribution of the optimized solutions is indeed identical to the target Mach number distribution. Figure 7.b shows that the nozzle corresponding to the optimized Mach number distribution fits perfectly the target nozzle.

### 3.4 Hierarchical GA, Multiple Models

Finally, a HGA with multiple models is applied to the problem. It uses Model 1 for the top layer: the grid has a size of 100 and the target is the one generated with 100 points. The intermediate layer uses Model 2 and a grid size of 50, while the fitness is computed with the target generated for 50 points (fig. 5). The bottom layer uses Model 3, with a grid size of 25 points and a target generated with 25 points.

<table>
<thead>
<tr>
<th>Pres</th>
<th>Evaluations</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-3})</td>
<td>1128</td>
<td>936 s (15mn)</td>
</tr>
<tr>
<td>(10^{-5})</td>
<td>1700</td>
<td>1850 s (31mn)</td>
</tr>
</tbody>
</table>

**Table 3. Hierarchical GA, Multiple Model**

Table 3 shows the performance of this algorithm (sub-population size = 10, \(\text{Epoch} = 20\)). The values of \textit{Evaluations} and \textit{CPU time} are averaged over 50 runs.
By comparing Table 2 and Table 3, we can assess the benefits of multiple models. For both criteria, HGA with multiple models largely outperforms HGA with a single model: it is respectively 3.5 times faster for the first stopping criterion and 2.8 times faster for the second stopping criterion.

Figure 8.a shows the evolution of the best solutions for the different sub-populations of the HGA with Multiple Models, for $Pres = 10^{-5}$.

![Figure 8](image)

(a) Convergence  
(b) Mach Distribution

**Fig. 8. Hierarchical GA Multiple Model**

Figure 8.b shows that the converged solution of the top layer is identical to the target Mach number distribution (computed with a grid size of 100).

4 Conclusion

<table>
<thead>
<tr>
<th>Pres</th>
<th>Single GA</th>
<th>HGA</th>
<th>HGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-3}$</td>
<td>43mn</td>
<td>55mn</td>
<td>15mn</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>1h47</td>
<td>1h28</td>
<td>31mn</td>
</tr>
</tbody>
</table>

This table sums up the performances for traditional GAs, HGAs and HGAs with multiple models. It focuses on the CPU time needed by each of these 3 approaches. It is clear that HGA with multiple models is the best answer to the dilemmas that were considered in the introduction. It is on average 3 times faster that either of the other approaches.

So the most important conclusion of this paper is that GAs handle well approximate models within a hierarchical topology. It shows that using models of different complexity can significantly speed-up an optimization process.

**Acknowledgments** We would like to thank Dr K. Srinivas, from the Dept of Aeronautical Engineering, University of Sydney, for providing the codes for the nozzle optimization problem.
References


EA Software
Take It EASEA

Pierre Collet, Evelyne Lutton, Marc Schoenauer, and Jean Louchet

Abstract. Evolutionary algorithms are not straightforward to implement and the lack of any specialised language forces users to reinvent the wheel every time they want to write a new program. Over the last years, evolutionary libraries have appeared, trying to reduce the amount of work involved in writing such algorithms from scratch, by offering standard engines, strategies and tools. Unfortunately, most of these libraries are quite complex to use, and imply a deep knowledge of object programming and C++. To further reduce the amount of work needed to implement a new algorithm, without however throwing down the drain all the man-years already spent in the development of such libraries, we have designed EASEA (acronym for EA sy Speci fication of Evolutionary Algorithms): a new high-level language dedicated to the specification of evolutionary algorithms. EASEA compiles .ez files into source files in a target language, containing function calls to a chosen existing library. The resulting source file is in turn compiled and linked with the library to produce an executable file implementing the evolutionary algorithm specified in the original .ez file.

EASEA v0.4 is available at: http://www-rocq.inria.fr/EVO-Lab/.

1 Introduction

Not so long ago, evolutionary algorithms were considered as mere curiosities set up by mad computer scientists. No respectable researcher would ever have considered using such algorithms to do anything serious. Things have changed however over the years and many end-users (chemists, physicists, mathematicians, ...) have ended up selling their scientific souls to DARWIN. Unfortunately taking this decision is not the hardest part of their ordeal: the evolutionary algorithm they have been dreaming of remains to be written and many of them are only occasional programmers, used to procedural languages such as PASCAL or FORTRAN. This is very understandable as they may not be state of the art computer scientists after all.

One way to speed up the process is to use one of the many existing evolutionary libraries. All is for the best as they offer very powerful tools provided ...
one is fluent enough with constructors, copy-constructors, destructors and such niceties involved by relatively low-level object languages.

The next hurdle is then to learn how to use the library, to understand the intricate data structures and to memorise the necessary several hundred object types, functions and variables and the way they are inter-related. This can be quite time consuming when all major evolutionary libraries are written in C++ or JAVA and make full use of object programming.

All in all, many physicists, chemists, mathematicians and other scientists who otherwise would be capable of writing relatively simple functions in C, FORTRAN or LISP are denied experimentation of evolutionary algorithms due to the sheer complexity of their implementation.

The aim of EASEA (EA$y$ Specification of Evolutionary Algorithms) is to hide this complexity behind a high-level language, allowing scientists to concentrate on evolutionary algorithms, rather than on their implementation.

2 Previous work

Some research teams have already felt the need for a specific evolutionary language. They have however chosen a theoretic viewpoint, trying to enrich the evolutionary paradigm with new concepts or features not yet implemented [7, 8, 10, 11].

We have chosen a radically different approach, trying to be as pragmatic as possible. Our goal was to start with the realisation of a minimal working prototype, able to implement almost any problem. We count on feedback from end-users to guide the evolution of EASEA.

3 Presentation of EASEA

3.1 Introduction

Several important ideas lie behind the EASEA language and compiler:

- EASEA must be general enough to be able to write virtually any evolutionary algorithm.
- Conceptually speaking, a language such as EASEA need not be tied to a specific evolutionary library. Hence, EASEA must be able to operate different evolutionary libraries.
- EASEA should aim to hide away all programming mechanisms not explicitly needed to describe the evolutionary algorithm.
- EASEA source files must be simple enough to be written automatically using a graphic user interface.
3.2 Mode of operation

The specifications of EASEA show that an EASEA compiler should be able to produce source code using different evolutionary libraries.

Two libraries have been chosen to start with: GAlib—a widely used C++ genetic library [5]— and EO (Evolutionary Objects [3]) developed at the University of Granada (Spain) within the EVONET [2] framework.

In parallel, EASEA will be used in the European DREAM project (Distributed Resource Evolutionary Algorithm Machine) [6], where the output is JAVA source code.

![Diagram of EASEA mode of operation]

**Fig. 1.** EASEA mode of operation.

EASEA–EO is still under development while EASEA–GAlib is already operational. EASEA–DREAM should be operational by November 2000.

The EASEA–GAlib compiler uses for input an ascii file with a .ez suffix. Its output is a GAlib C++ source file including calls to GAlib functions and objects. The resulting C++ file must then be compiled by a C++ compiler and linked with the GAlib library (cf. figure 1). The produced executable implements the evolutionary algorithm described in the original EASEA source file.

3.3 Graphic User Interface

The EASEA language is in fact an intermediary language designed to decouple a graphic user interface from evolutionary libraries.

The GUI (under development) allows the user to graphically specify his evolutionary algorithm. Once this is done, the GUI creates the corresponding EASEA source file, which in turn automatically compiled into an object source file using the specified library. The object source file (containing an interface with the GUI) is compiled to produce an executable file which is run by the GUI, which is then used to control the running evolutionary algorithm and to display partial and final results.
4 EASEA compiler

4.1 Description

EASEA is written in C++, using Lex and Yacc (in fact ALex and AYacc [4]). The EASEA compiler is somewhat unusual in the sense that it produces source code in another language rather than microprocessor instructions. As EASEA syntax is rather simple, most serious errors come in fact from user-functions which are compiled by a host C++ compiler. The nice consequences are that such errors are trapped by the very elaborate host compiler syntax analyser and that semantic errors (bugs) are as elaborately dealt with by the host compiler symbolic debugger. The not so nice consequence is that the human end-user must somehow debug the C++ code produced by EASEA, which requires highly readable source code.

The main difficulty resides in the fact that humans usually find compiler-produced source code quite difficult to read.

A further factor requiring readable generated C++ is that we think EASEA can also be used as a primer: EASEA creates a C++ source file which can be a starting point for an evolutionary algorithm that will be refined afterwards.

Our main concern has then been to improve presentation and to have EASEA-generated C++ look as human as possible.

This is mainly achieved through:

1. a man-made template file —galib.tpl for the GAlib version. As one can infer by its name, the GAlib template file contains the framework of a generic GAlib evolutionary algorithm, ready to be filled up with user-specific information found in the EASEA .ez source file.

2. very carefully typeset code, whenever EASEA generates code to fill up the blanks: indentation is respected, meaningful variable names are used and comments are generated from scratch to explain what the created code is supposed to do.

The compiler contains two main parts: one responsible for the genome analysis and the other responsible for code production.

4.2 Genome declaration analysis

EASEA genome declarations look very much like C or C++ structure declarations. char, int, double, bool are accepted as basic types. Modifiers are accepted, allowing arrays and pointers to be declared. Finally, it is possible to declare new types (classes in fact, as EASEA is fully object-oriented).

Let us imagine, as a demonstrative example, a genome representing a polygon:

```cpp
Side{ int Coord[2];
    Side *pNext; }
Genome{ Side *pList;
        int NbSides; }
```

This genome is made of a pointer towards a linked list of sides and of `NbSides`, an integer containing the number of elements in the linked list.
A side is made of an array of two coordinates and a pointer towards the next side.

All variables are stored in a symbol table, along with their type and size (arrays). New user types are stored along with the elements they contain.

4.3 Generation of complete C++ classes

The template file is in fact an empty shell, containing the source code for a generic GALib evolutionary algorithm. The EASEA compiler copies lines from the template file towards the object .cpp file until it comes across a compiler directive telling it to insert information which is to be found in the user-supplied .ez file.

The user-defined types and functions are then inserted in the output .cpp file, as well as the genome declaration:

- New types are inserted as new C++ classes, with all methods necessary to obtain fully fledged C++ classes (constructor, destructor, copy-constructor, operator=, operator==, operator!=, operator<<, operator>>).

Here is for instance the operator= member function, transparently created by the EASEA compiler for class Side:

```cpp
Side operator=(Side &EASEA_Var) { // Operator=
    if (pNext) delete pNext;
    EASEA_Var.pNext ? pNext=new Side(*(EASEA_Var.pNext)):pNext=NULL;
    for(int EASEA_Ndx=0; EASEA_Ndx<2; EASEA_Ndx++)
        Coord[EASEA_Ndx]=EASEA_Var.Coord[EASEA_Ndx];
    return *this;
}
```

- The Coord array has been automatically expanded and each of its elements are assigned individually.
- If the R-value pNext pointer is not null (EASEA_Var.pNext), a new Side object is created, and its copy-constructor is invoked with the object pointed to by the R-value pNext pointer. This new Side object is assigned to the L-value pNext pointer.

As an intelligent copy-constructor has also been created automatically for class Size, this results in a recursive copy of the linked list.

- The genome is derived from the GALib genome class. As for new user classes, all generic member functions are created transparently (constructor, destructor, copy-constructor, operator=), as well as all member functions required by GALib (clone, copy, equal, read, write, and Comparator).

The remaining member functions required by GALib (Initializer, Mutator, Evaluator and Crossover) are, (in v0.4), user-specific. This means that EASEA will look for their C++ code in the .ez file.

4.4 Remaining independent of target libraries

Accepting different target libraries means that EASEA cannot force the programmer to use the GALib-specific GARandomDouble function. This would mean
that an EASEA .ez source file would not recompile if another library were to be used.

The solution is for EASEA to provide (as in any new language), its own specific predefined functions and keywords. Whenever such functions or keywords appear in the .ez source file, the EASEA compiler translates them to their GAlib (or EO or DREAM) equivalent. As such, the EASEA random function call will be translated in its GAlib equivalent: GARandomDouble.

5 Performance

The concern about performance surfaces whenever a piece of code is generated by a compiler. First of all, as far as syntax is concerned, EASEA-produced C++ files are not that different from what human-produced code would have looked like ... after debugging. Semantically speaking, it is true that when writing minor classes, a human programmer will not take the pain of writing code for operators that he knows will never be called. Although such refinement could be included with much pain in EASEA (a first pass could determine which operators of which classes will be needed), the only drawback is that the selection scheme will deal with slightly larger objects than necessary.

However, this cost is negligible, mainly owing to two facts:

1. EASEA generates source code, which is destined to be compiled by an extremely evolved C++ compiler. The code optimisation taking place in the C++ compiler will minimise the lack of optimisation of the EASEA output.
2. EASEA-generated code only concerns the manipulation of genome objects which usually represents only a few percent of the total execution time of an evolutionary algorithm (usually dominated by the user-written evaluation function).

6 Real-world applications

6.1 Matching reviewers and papers for PPSN VI

The present PPSN VI conference received 155 papers which were to be reviewed by 178 selected reviewers. Assigning papers to reviewers is a painstaking chore which necessitates a couple of full-time human days, as it had been (at first) decided that three reviewers should read each paper.

The quality of the match is essential, as judging a paper can rapidly become problematic if the subject falls outside of the reviewer’s field of competence. Honest reviewers will tell the organiser they cannot evaluate the paper, causing delays due to the necessary redirection while shy ones will probably give an average mark, unless the paper is utterly unreadable.

A bad match between reviewers and papers therefore has at least two disagreeable consequences:
1. Embarrassed reviewers, wishing they had not accepted to review papers for this conference in the first place.
2. And above all, a boring conference, where uninteresting papers have been selected by badly chosen reviewers.

**Matching criteria** A basic match between papers and reviewers supposes that at least two constraints be satisfied:

1. Each paper must be examined by \( N \) reviewers.
2. A reviewer must not review a paper of which he is an author.

Nicely matching papers with reviewers involves maximising an evaluation function, depending on the following criteria:

1. If 155 papers are to be reviewed by 178 reviewers and if each paper should be reviewed \( N = 3 \) times, this means that each reviewer should be given an average of \( 155 \times 3/178 = 2.61 \) papers to review. It is understandable that we should try to avoid having too many reviewers with 0 or 6 papers to review.
2. So as to avoid biases, it is preferable that reviewers should not know authors personally. Although this is difficult to determine considering the little information available in the databases, we can attempt to minimise such “risks” thanks to the two last fields of e-mail addresses.
3. Finally, the requirement is to match papers and reviewers who have a maximum of keywords in common, taking into account the reviewers’ preferences.

**Implementation with EASEA** Implementation (using EASEA v0.35) was quite straightforward.

Knowing \( N \) (number of reviewers per paper), the simplest genome structure capable of representing a solution is an array of \( P \) papers to which \( N \) reviewers are assigned, resulting in a bidimensional \( P \times N \) array. This has the advantage of automatically satisfying constraint number 1.

The *Classes* section contained the different classes needed by the genome and the genome itself:

```cpp
Match { int reviewer[3]; }
Genome { Match paper[200]; }
```

As EASEA v0.35 could not yet handle multidimensional arrays, we defined the genome as an array of 200 papers of type *Match*. By doing so, the second reviewer of paper 23 is accessed by `paper[22].reviewer[1]`.

The *Standard functions* sections contained the specific genetic operators, namely:

1. The initialisation function (matching all papers with random reviewers).
   Each paper in the array is randomly assigned \( N \) reviewers (constraint 1 is satisfied, while constraint 2 is taken care of by the evaluation function).
2. The crossover function. To keep it simple, a single point cross-over can be easily defined as follows:

Let parent1 and parent2 be the two genomes out of which child1 and child2 must be generated, and let L be the locus where the cross-over will take place:
- child1 will inherit papers 0 to \( L - 1 \) from parent1 and will inherit papers \( L \) to \( P - 1 \) from parent2.
- child2 will inherit papers 0 to \( L - 1 \) from parent2 and will inherit papers \( L \) to \( P - 1 \) from parent1.

3. The mutation function. Thanks to the remarkable robustness of evolutionary algorithms, we can allow ourselves to randomly change the reviewers of some papers in the genome.

4. The evaluation function. The fitness function has two aims: maximising the quality of the solution, and making sure constraint 2 is satisfied.
- Here again, to keep things simple, the easiest way to ensure that constraint 2 is not broken is to punish the genome with a -1000 penalty for every offending match between paper and reviewer sharing the same e-mail.
- Fulfilling reviewers’ preferences and matching keywords is done by by giving nicely chosen bonus points.

Thanks to EASEA, the end-user only has to write in procedural C++ the bodies of the “interesting” functions of an evolutionary algorithm. EASEA takes care of wrapping them into object oriented C++ to create a GALib program.

![Fig. 2. Keyword match](image)
Results The results of this automatic assignment (summed up in figures 2 and 3) were passed on to the conference organisers who used them as a basis for the final assignment.

From this starting point, for each paper, they have looked at the keyword match of the selected reviewers. If for a majority of reviewers, the result was 0 (no keyword match) or −1 (the reviewer had not given any keywords), they have replaced at least one reviewer with another one, either willing or with a keyword match. At the same time, they have tried to improve the balance the number of papers per reviewer.

Finally, according to them, “[the final assignment] is almost the GA-generated result, with some knowledge of experts for each paper.”

After this was done, the organising committee finally decided to manually add one more reviewer per paper for security, in case of defective reviewers, raising the average number of papers per reviewer to 3.50.

Other applications and academic experience With an application where EASEA was used to optimise airfoil shapes arose the necessity to parallelise evaluations (each evaluation, written as a FORTRAN function takes several minutes). A special “EASEAv0.4mpi” version was released, which automatically produces parallel code, to be used with the MPI library. The parallelisation is very basic in the sense that for each generation, the evaluation of the population is distributed on all available machines, but taking into account that no effort is required to parallelise the code other than compiling the standard .ez file with EASEAv0.4mpi, this new feature seems to fulfill the aim of EASEA. Transplant-
ing the FORTRAN evaluation function from the locally made genetic algorithm within the EASEA framework took three hours for a masters’ student who was discovering EASEA.

EASEA was also taught in a French engineering school where 19 students implemented animats with Genetic Programming “à la Koza.” In this case, the students had never heard of evolutionary algorithms before, and were able to complete their projects in one month.

7 Future work

Feedback from scientific users is quite positive although v1.0 is still far down the road. Necessary improvements include:

1. full support of other libraries (EO, DREAM project, ...),
2. utilisation of target libraries complex types and operators (arrays, lists, trees, ... and their corresponding operators),
3. implementation of default representation-independent operators for user-defined genomes.

The first point is very important, as supporting several libraries is what will guide the evolution of the EASEA language towards the abstract evolutionary programming language it aims to be.

The second and third points will drastically simplify EASEA source files: most evolutionary libraries already offer complex structures (arrays, lists, trees, ...) and their corresponding operators. As soon as EASEA is capable of making use of those complex types and their default operators, default initialisation, mutation and crossover functions will not be needed anymore in .ez files, unless the programmer feels the need to specialise some of them.

Removing genome-specific operators from .ez files is possible: in many cases, user genomes will be aggregates of available types (e.g. vectors of structures made of floats, integer and symbolic components). It is thus possible to define default crossover operators for such representations using RADCLIFFE’s ideas [8]. The three crossover operators (Random Respectful Recombination, Random Assorting Recombination and Random Transmitting Recombination) as well as the Binomial Minimal Mutation are perfect candidates for that. Of course, representation-specific operators will still be allowed in .ez files, as it is acknowledged that they are often more efficient than representation-independent operators [9]. Nevertheless, providing yet more efficient default operators will be an important step towards real newcomers in the field (e.g. “I only want to evolve my vector of structures and don’t want to hear about it in the final result”).

8 Conclusion

Many important fields in computer science have their specific languages (FORTRAN, C/C++, LISP, PROLOG, SMALLTALK, ...). Even complex applications
such as databases or spreadsheets have developed their own languages! EA programmers remain however with C++, an inadapted and difficult to use low-level object-oriented language. As a result, many scientists have no other choice than spending a lot of time becoming computer programmers and rewriting their own evolutionary algorithms. Due to thoroughly different programming techniques and languages, their programs are barely comparable, which is a great obstacle to scientific cooperation and emulation.

The simplicity of EASEA programming is demonstrated with the onemax and listsort programs provided on the web page. Although the EASEA v0.4 compiler is still minimal (it should not be necessary, for instance, to re-write completely initialisation, crossover and mutation functions for as basic a structure as an array of booleans) v0.4 can handle linked lists, trees or much more complex structures while hiding from the end-user all of the obscure uninteresting code necessary to operate object-oriented libraries.

EASEA source files are designed to be recompilable with minimal effort on different libraries, so that different research teams will be able to try out each others' implementations in their own environment.

We hope that EASEA will be able to offer the scientific community the means to try out evolutionary algorithms with a minimal time investment as far as programming is concerned. The EASEA v0.4 compiler and its manual are available on the net [1].

8.1 Acknowledgement
We would like to thank Antoine LABOUR for his contribution. His deep understanding of the intricacies of C++ allowed him to fix in one hour a subtle but major bug which had already discouraged many others.

References

1. EASEA v0.4 home page: http://www-rocq.inria.fr/EVO-Lab/.
Evolutionary Computation Visualization: Application to G-PROP

G. Romero\textsuperscript{1}, M.G. Arenas\textsuperscript{1}, J.G. Castellano\textsuperscript{1}, P.A. Castillo\textsuperscript{1}, J. Carpio\textsuperscript{1}, J.J. Merelo\textsuperscript{1}, A. Prieto\textsuperscript{1} and V. Rivas\textsuperscript{2}

\textsuperscript{1} Department of Architecture and Computer Technology
University of Granada
Campus de Fuentenueva
E. 18071 Granada (Spain)

\textsuperscript{2} Department of Computer Science
University of Jaén
Avda. Madrid, 35
E. 23071 Jaén (Spain)

e-mail: todos@geneura.ugr.es URL: http://www.geneura.org

Abstract. Software visualization is an area of computer science devoted to supporting the understanding and effective use of algorithms. The application of software visualization to Evolutionary Computation has been receiving increasing attention during the last few years. In this paper we apply visualization technique to an evolutionary algorithm for multilayer perceptron training. Our goal is to better understand its internal behavior in order to improve the evolutionary part of the method. As a result of applying this technique several deficiencies in the method have been discovered.

1 Introduction

Evolutionary Algorithms (EA) work in an algorithmically simple way but produce a vast amount of data. Apart from simple convergence towards the solution, the extraction of useful information to get further insight into the state and course of the algorithm is a non-trivial task. Understanding their behavior is difficult due to the fact that EAs adopt an interactive stochastic approach to searching large problem spaces.

EA users usually examine how the quality of the solutions found changes over time by using a graph of fitness versus generation number. Although such a graph illustrates the improvements in the quality of the solutions considered during the algorithm run, it does not illustrate the structure of the solutions being considered, or the regions of the search space being explored. Visualization is proposed as a useful method for solving this problem; only by understanding the search behavior of their algorithms can EA users be confident about their individual algorithm components and parameter settings.
2 Visualization techniques

Visualization techniques have been used in EA whenever possible due to their descriptive capacity. As the old saying goes, a picture is worth a thousand words. These techniques provide a baseline for a better understanding of the evolutionary process.

Visualization techniques can be divided into categories according to various criteria:

- **What is seen.** You can see the genotype, the representation of problem solutions, or the phenotype, the objective value assigned to them. In some (very easy) problems both of them, genotype and phenotype, can be the same, but it is not very common. Figure 1 shows the genotypes of a population, its internal representation, in several different states of the evolutionary process. In the last picture in Figure 1 we can see how good the individuals are in a fitness versus generation number graph.

- **How much generations data is seen at once.** You can see data produced over many generations, so you can get a picture of the progress of the evolutionary algorithm, or you can see the data produced anew for every generation, so a picture of the current state of the algorithm is obtained. In Figure 1 the first five pictures refer to different states of the evolutionary process, while the last picture shows information about the whole process. This second way of classification is the more frequently used.

![Figure 1](image-url)

**Fig. 1.** Traveling Salesman Problem with 20 cities: Sequence of populations from generation 0 to 70 and fitness convergence towards global minimum path.
2.1 Visualizing genotypes or phenotypes

Genotype is the representation to a problem solution. For an easy problem like maximizing the function in Figure 2a, it can be as simple as a pair of real numbers or a bit string representing the values of x and y. For the Traveling Salesman Problem (TSP) it can be an integer vector representing a path through the cities. For a Genetic Programming (GP) problem it can be a tree representing an algorithm in any programming language.

Phenotype refers to the objective value corresponding to every genotype. To use the same examples as above, for the function in Figure 2 it can be the value of the function with the corresponding values of x and y. For the TSP problem it can be the length of the path that the integer vector represents. For the GP problem, it can be the degree of agreement with the objective algorithm or its degree of efficiency.

The difference is evident, with one way you can see the structure of the solutions, with the other you can see and compare how good those solutions are.

Visualizing phenotypes

Visualizing phenotypes is the easiest and oldest visualization technique used in the evolutionary computation field. This technique usually considers fitness and fitness distribution in very similar ways. Normally, an individual’s phenotype, its goodness or whatever we call it, is reduced to a single number called fitness. Researchers plot this against time or the fitness of other individuals. Two examples of phenotype visualization are last picture in Figure 1 and Figure 2b. This kind of representation is standard, the same for all problems. Sometimes, when a more complex fitness is used, especially in multi-objective problems, two solutions can are possible: to plot every objective value separately or any combination of all of them.

Visualizing genotypes

This is a less common way of visualizing what is happening with our evolutionary algorithm. This is not usually as easy as showing phenotype progress. For a very easy problem, like the one in Figure 2a, we can show the evolution of the internal representation of the solutions as in Figure 3. The representation of this problem is a pair of real numbers. So we can plot them in a 2D plane, or even better, put them over a 3D function plot. Other examples are the TSP problem populations illustrated in the first five pictures in Figure 1. The tour that every individual represents is plotted and its length can be compared at a glance. The first tours use a lot of path through distant cities, an a lot of diagonals. In the last ones, however, only horizontal and vertical paths between the nearest cities are crossed.

Unfortunately, this way of visualizing genotypes depends on its internal representation, and as this internal representation is different for different problems, there is no standard way of doing it. So a different way of visualization has to be programmed for every kind of internal representation.
Fig. 2. (a) Test function: $1 + \frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$. (b) Typical fitness versus generation number graph of an evolutionary algorithm run showing maximum, average and minimum fitness of every generation.

2.2 Visualizing the state or the course of the evolution

This kind of classification depends on how many generations' data can be seen at the same time. Looking at one generation’s data will yield a picture of the state of the population at that moment. Sometimes data from many generations can be seen at the same time, showing the course of the evolutionary process. Ideally we would be able to see data from many generations and at the same time distinguish which generation these data came from. Unfortunately this is usually difficult to achieve.

Visualizing the state of the evolution

Many conclusions can be drawn from a look at the state of an evolutionary algorithm. What degree of diversity exists? What areas of the search space are being explored? All of the first five pictures in Figure 1 answer some questions about what solutions look like and how good they are. Similar information can be obtained from any of the pictures in 3.

This gives a lot of information about how the algorithm is working in just a moment, but unfortunately nothing about the whole process. However several of these pictures can be used to extract information about the progress of an algorithm.

Visualizing the course of the evolution

One of the mayor drawbacks of evolutionary computation is the lack of interaction during the process. As generations pass people hope that everything will go all right, sometimes looking at some statistical figures. By using some kind
of visualization we can follow the process much better and stop it if it is getting lost in the search space or, even better, tune it to improve its work.

Some of the previous examples can fit into this category. The last picture, or the set formed by the first five, in Figure 1, gives us information about the whole process of evolution. The length of the shortest path decreases quickly during the first 40 generations. After that, convergence towards the optimum path is slower until the best one is found around generation 70. If somebody is looking at these pictures, and the situation persists, they will be able to stop the process and restart it, or better, modify some parameters to avoid the standby. Figure 2b and Figure 3 give us similar information about how the process of maximizing the function in Figure 2a is progressing.

3 Multidimensional visualization

Most techniques for visualization are limited to representing data depending on one or two variables. This is due to the fact that human vision is limited to three dimensions but there are two possible extensions to go beyond this limitation: using color for the fourth dimension and time as the fifth dimension. Neither possibility is very common and requires practice, especially if time is used for visualizing the fifth dimension. However, if the problem incorporates more than five dimensions a method for visualizing arbitrarily high dimensions must be used.

For the visualization of multidimensional data, a method to transform multidimensional data to a lower dimension is needed, preferably to 2 or 3 dimensions. This transformation should provide a lower-dimensional picture where the dissimilarities between the data points of the multidimensional domain correspond
to the dissimilarities of the lower-dimensional domain. These transformation methods are referred to as multidimensional scaling ([1, 2]).

To measure the dissimilarity, the distances between pairs of data points is used. These distances can be genuine distance in the high-dimensional domain, for instance the Euclidean distance.

One of the best known methods for multidimensional scaling is Sammon Maps [3]. Sammon Maps use a Newton method (steepest gradient descent). However this method is not very robust and diverges without special interaction [2]. Besides, it needs to be trained again for every new point that needs to be projected.

Other multidimensional scaling methods are BFGS [4] and RPROP [5]. BFGS is a standard optimization method included in Matlab. It uses a Quasi-Newton method with a mixed linear quadratic and cubic line search procedure. RPROP is a more robust search method. The RPROP algorithm uses just the change of the sign of the gradient for step size control. It is widely used in the field of neural networks. Both algorithms produce good results. However, when using the BFGS method, multiple runs must be performed and then results must be valued by the user. The RPROP algorithm is slower but produces more consistent results. Like Sammon Maps, they have to be trained again in order to project new points.

Another method is Kohonen's Self-Organizing Maps (SOM) [6]. SOM is an artificial neural net specially adapted for the task in question. With all the other methods, once the data points are transformed from high-dimensional space to the lower-dimensional one, no new points can be projected without repeating the whole process. SOM has the advantage of being able to transform new points between spaces very easily once it has been trained.

4 Visualizing G-PROP

G-PROP is a new method for training Multilayer Perceptrons (MLP). It is based on a mixture of two methods: Quickprop [7] and evolutionary algorithms. In our research group it has been used for pattern recognition [8–12] and for functional approximation [13].

The objective of applying visualization techniques to G-PROP is to gain a better understanding of the method so that we can use it more accurately, especially as regards the evolutionary algorithm.

For the experiments, we used various well known training sets from several papers on the neural network field. Our goal in this case was to recognize the underlying patterns:

Cancer This dataset is from the UCI machine learning dataset "Wisconsin breast cancer database". This breast cancer database was obtained from the University of Wisconsin Hospital, Madison from Dr. William H. Wolberg [14]. An exhaustive report, by Prechelt, on this dataset (and others) is given in [15].
**DNA Helicases** This is a problem of electron microscopy image classification. The object of the classification is to study the structure of a representative hexametric helicase: the large T antigen of Simian Virus 40 [16].

**Glass** This problem consists of the classification of glass types, and is also taken from [15]. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, containers, tableware or head lamps.

As a neural net can be seen as a real number vector, formed by all its weights, it was very easy to use it with the SOM. The SOM was trained with all the individuals generated during an experimental run. After several runs of the algorithm, we picked some significant experiments to work with. To select a good experiment we chose the ones with a average results. The behavior of the algorithm was very similar from run to run, which led us to consider this method as a very robust one.

The algorithm finds very close to the best multilayer perceptron (MLP), the one with the best pattern recognition capabilities, in the first few generations. This can be seen in Figure 4 in the fitness value or in Figure 5 as an individual appearing in the white area. The algorithm dedicates most of its time to reducing the number of neurons in the hidden layer net. Figure 4 shows a pair of graphs about how classification accuracy increases with time as the number of neurons in the hidden layer decreases. The low classification capability improvement over time suggests that perhaps a premature convergence phenomenon is occurring. This led us to change the evolutionary algorithm. Some kind of fitness scaling would prevent premature convergence from occurring. By shortening the differences between fitness values the increase in diversity would be favored at the beginning of the training.

![Fig. 4. Evolution of net accuracy and size for the Cancer problem.](image)

We now describe how SOM is used to project MLPs. Every individual has a fitness value between 0 and 1, corresponding to a correct classification capability
from 0 to 100%. If two nets have the same accuracy, the smaller one, the one with fewer neurons is considered better. Then a color is assigned to every one according to its fitness. The best ones are lighter-colored while the worse ones are darker. In this way we can see a grey scale describing the fitness. To assign a color to every cell in the SOM, the closest net color was selected. Thus the SOM is trained to compare the neural networks according to their similarity (vector of weights = genotype) and not to the result produced (pattern recognition capability = phenotype). This is why color areas are not contiguous. This coloring method was selected because we want to learn about the structure of the solutions, not their objective value, which is easily comparable.

The results of the mapping of the nets can be seen in Figure 5 with screen shots corresponding to 6 stages of an experiment run. As explained above, the background color represent fitness, and crosses are the individuals. Let us comment on these:

**Generation 0** This is the initial population. As expected all but one of the individuals appear near of the worst region, the darkest one. The different runs show that populations could be initialized better, in a more dispersed way. All the individuals appear in a very small area instead of being distributed throughout the search space. As this bad start is always repeated, it should be modified as suggested in Section 5.

**Generation 20** The population begins to disperse within the search space. Diversity is increased, that is to say, the differences of the individuals to each other. Now the individuals cover a wider area both in space and in color (quality of the solutions). They expand from right to left.

**Generations 40 & 60** The evolutionary algorithm continues looking for better solutions close to those already known (exploitation in the left area) and also starts exploring new regions of the space for better solutions (exploration of top area).

**Generations 80 & 99** At the end the evolutionary algorithm dedicates most of its time to exploiting the good solutions found so far, mainly in the top area. No significant changes occur during these final generations.

## 5 Conclusions and future work

The use of visualization techniques provided a better knowledge about the evolutionary algorithm and gave us some clues about how it can be improved. The following conclusions were drawn from the application of visualization techniques to G-PROP:

- The way population is initialized should be improved. Almost all initial individuals lay in a very small area instead of being widely spread throughout the search space.
- MLPs with best (or almost best) classification capacity are found in the first few generations. This suggests that Quickprop algorithm is used to excess during initialization thus favoring super individual creation and all its
adverse consequences: rapid diversity decrease and premature convergence towards local optima. Reducing Quickprop use and applying fitness scaling, or even better, fitness sharing, would prevent this phenomena from happening.

These are some future lines of work on the optimization of the G-PROP algorithm:

- We must solve the problems previously enumerated, i.e. bad initialization and premature convergence, using some kind of fitness scaling or fitness sharing.
- As the fitness evaluation process is very time consuming, a SOM can be used to evaluate only potentially good individuals and avoid bad ones. This can be done by evaluating only individuals mapped onto good regions by SOM.
- G-PROP improves the net layout but hardly affects its accuracy. We want to experiment with some new genetic operators to improve both.
- Some kind of interaction during the process would be desirable: changing parameters dynamically and adding or removing individuals to/from any region of the search space using SOM neurons as MLPs.

Fig. 5. A sequence of generations from G-PROP. Background color represent fitness (the whiter, the better). Crosses are individuals.
Acknowledgements

This work has been supported in part by FEDER I+D project 1FD97-0439-TEL1, CICYT TIC99-0550 and INTAS 97-30950.

References

<table>
<thead>
<tr>
<th>Author</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agrawal, S.</td>
<td>849</td>
</tr>
<tr>
<td>Alba, E.</td>
<td>29</td>
</tr>
<tr>
<td>Albrecht, A.</td>
<td>243</td>
</tr>
<tr>
<td>Arenas, M. G.</td>
<td>902</td>
</tr>
<tr>
<td>Arnold, D. V.</td>
<td>39</td>
</tr>
<tr>
<td>Bäck, Th.</td>
<td>315</td>
</tr>
<tr>
<td>Backer, E.</td>
<td>517</td>
</tr>
<tr>
<td>Banzhaf, W.</td>
<td>201</td>
</tr>
<tr>
<td>Belaidouni, M.</td>
<td>49</td>
</tr>
<tr>
<td>Ben Hamida, S.</td>
<td>529</td>
</tr>
<tr>
<td>de Berg, M.</td>
<td>683</td>
</tr>
<tr>
<td>Berny, A.</td>
<td>601</td>
</tr>
<tr>
<td>Bessaou, M.</td>
<td>437</td>
</tr>
<tr>
<td>den Besten, M.</td>
<td>611</td>
</tr>
<tr>
<td>Beyer, H.-G.</td>
<td>39, 59</td>
</tr>
<tr>
<td>Boettcher, S.</td>
<td>447</td>
</tr>
<tr>
<td>Bosman, P. A. N.</td>
<td>767</td>
</tr>
<tr>
<td>Branke, J.</td>
<td>253</td>
</tr>
<tr>
<td>Bright, M.</td>
<td>583</td>
</tr>
<tr>
<td>Brinkers, M.</td>
<td>457</td>
</tr>
<tr>
<td>Butz, M. V.</td>
<td>735</td>
</tr>
<tr>
<td>Buydens, L. M. C.</td>
<td>702</td>
</tr>
<tr>
<td>Caprio, J.</td>
<td>902</td>
</tr>
<tr>
<td>Castellano, J. G.</td>
<td>902</td>
</tr>
<tr>
<td>Castillo, P. A.</td>
<td>902</td>
</tr>
<tr>
<td>Collet, P.</td>
<td>891</td>
</tr>
<tr>
<td>Corne, D. W.</td>
<td>839, 869</td>
</tr>
<tr>
<td>Costa, E.</td>
<td>221</td>
</tr>
<tr>
<td>Cotta, C.</td>
<td>325</td>
</tr>
<tr>
<td>Deb, K.</td>
<td>59, 849, 859</td>
</tr>
<tr>
<td>Delahaye, D.</td>
<td>777</td>
</tr>
<tr>
<td>Der, U.</td>
<td>243</td>
</tr>
<tr>
<td>van Dijk, S.</td>
<td>683</td>
</tr>
<tr>
<td>Dorigo, M.</td>
<td>611</td>
</tr>
<tr>
<td>Dozier, G.</td>
<td>561</td>
</tr>
<tr>
<td>Droste, S.</td>
<td>181</td>
</tr>
<tr>
<td>den Dulk, P.</td>
<td>457</td>
</tr>
<tr>
<td>Durbin, D. R.</td>
<td>745</td>
</tr>
<tr>
<td>Eckert, C.</td>
<td>345</td>
</tr>
<tr>
<td>Eiben, A. E.</td>
<td>315</td>
</tr>
<tr>
<td>English, T. M.</td>
<td>69</td>
</tr>
<tr>
<td>Esquivel, S. C.</td>
<td>263</td>
</tr>
<tr>
<td>Ficici, S. G.</td>
<td>467</td>
</tr>
<tr>
<td>Filipič, B.</td>
<td>693</td>
</tr>
<tr>
<td>Fonlupt, C.</td>
<td>191</td>
</tr>
<tr>
<td>Francois, O.</td>
<td>79</td>
</tr>
<tr>
<td>Gallard, R. H.</td>
<td>263</td>
</tr>
<tr>
<td>Gehlen, S.</td>
<td>293</td>
</tr>
<tr>
<td>de Gelder, R.</td>
<td>702</td>
</tr>
<tr>
<td>Goldberg, D. E.</td>
<td>385, 395, 631, 735</td>
</tr>
<tr>
<td>Gottlieb, J.</td>
<td>345, 621</td>
</tr>
<tr>
<td>Grigni, M.</td>
<td>447</td>
</tr>
<tr>
<td>Grün, A.</td>
<td>661</td>
</tr>
<tr>
<td>Hageman, J. A.</td>
<td>702</td>
</tr>
<tr>
<td>Hanke, D.</td>
<td>293</td>
</tr>
<tr>
<td>Hansen, N.</td>
<td>355</td>
</tr>
<tr>
<td>Hao, J.-K.</td>
<td>49</td>
</tr>
<tr>
<td>Herrera, F.</td>
<td>335</td>
</tr>
<tr>
<td>Heutelbeck, D.</td>
<td>181</td>
</tr>
<tr>
<td>Higuchi, T.</td>
<td>365, 589</td>
</tr>
<tr>
<td>Hogeweg, P.</td>
<td>797</td>
</tr>
<tr>
<td>Holmes, J. H.</td>
<td>745</td>
</tr>
<tr>
<td>Ikeda, K.</td>
<td>273</td>
</tr>
<tr>
<td>Inoue, Y.</td>
<td>477</td>
</tr>
<tr>
<td>Iwata, M.</td>
<td>589</td>
</tr>
<tr>
<td>Jansen, T.</td>
<td>89</td>
</tr>
<tr>
<td>Jelasity, M.</td>
<td>755</td>
</tr>
<tr>
<td>Jensen, M.T.</td>
<td>283</td>
</tr>
<tr>
<td>Joung, J.-G.</td>
<td>231</td>
</tr>
<tr>
<td>Kargupta, H.</td>
<td>99</td>
</tr>
<tr>
<td>Keymeulen, D.</td>
<td>589</td>
</tr>
<tr>
<td>Kita, H.</td>
<td>571</td>
</tr>
<tr>
<td>Knjazew, D.</td>
<td>631</td>
</tr>
<tr>
<td>Knowles, J. D.</td>
<td>839, 869</td>
</tr>
<tr>
<td>Kobayashi, S.</td>
<td>273</td>
</tr>
<tr>
<td>Kratica, J.</td>
<td>641</td>
</tr>
<tr>
<td>Kreutz, M.</td>
<td>293</td>
</tr>
<tr>
<td>Krink, T.</td>
<td>375, 817</td>
</tr>
<tr>
<td>Kubo, M.</td>
<td>477</td>
</tr>
<tr>
<td>Landrin-Schweitzer, Y.</td>
<td>109</td>
</tr>
<tr>
<td>Langdon, W. B.</td>
<td>201</td>
</tr>
<tr>
<td>Linke, S.</td>
<td>661</td>
</tr>
<tr>
<td>Liu, Y.</td>
<td>589</td>
</tr>
</tbody>
</table>
Lecture Notes in Computer Science

For information about Vols. 1–1835 please contact your bookseller or Springer-Verlag


