GENETIC AND EVOLUTIONARY COMPUTATION
Series Editors: David E. Goldberg and John R. Koza

Edited by
Rick Riolo
Terence Soule
Bill Worzel

Genetic Programming Theory and Practice V

Springer

Rick Riolo, Terence Soule and Bill Worzel (Eds.)

Genetic Programming Theory and Practice V

Genetic and Evolutionary Computation Series

Series Editors

David E. Goldberg
Consulting Editor
IlliGAL, Dept. of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
Email: deg@uiuc.edu

John R. Koza
Consulting Editor
Medical Informatics
Stanford University
Stanford, CA 94305-5479 USA
Email: john@johnkoza.com

Selected titles from this series:

Markus Brameier, Wolfgang Banzhaf
Linear Genetic Programming, 2007
ISBN 978-0-387-31029-9

Nikolay Y. Nikolaev, Hitoshi Iba
Adaptive Learning of Polynomial Networks, 2006
ISBN 978-0-387-31239-2

Tetsuya Higuchi, Yong Liu, Xin Yao
Evolvable Hardware, 2006
ISBN 978-0-387-24386-3

David E. Goldberg
The Design of Innovation: Lessons from and for Competent Genetic Algorithms, 2002
ISBN 978-1-4020-7098-3

John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, Guido Lanza
Genetic Programming IV: Routine Human-Computer Machine Intelligence
ISBN: 978-1-4020-7446-2 (hardcover), 2003; ISBN: 978-0-387-25067-0 (softcover), 2005

Carlos A. Coello Coello, David A. Van Veldhuizen, Gary B. Lamont
Evolutionary Algorithms for Solving Multi-Objective Problems
ISBN: 978-0-306-46762-2*,* 2002; ISBN 978-0-387-33254-3 (2nd Edition), 2007

Lee Spector
Automatic Quantum Computer Programming: A Genetic Programming Approach
ISBN: 978-1-4020-7894-1 (hardcover), 2004; ISBN 978-0-387-36496-4 (softcover), 2007

William B. Langdon
Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming! 1998
ISBN: 978-0-7923-8135-8

For a complete listing of books in this series, go to http://www.springer.com

Rick Riolo
Terence Soule
Bill Worzel
(Eds.)

Genetic Programming Theory and Practice V

Springer

Rick Riolo
Center for the Study of Complex Systems
323 West Hall
University of Michigan
Ann Arbor, MI 48109
rlriolo@umich.edu

Terence Soule
Department of Computer Science
University of Idaho
Janssen Engineering Building
Moscow, ID 83844-1010
tsoule@cs.uidaho.edu

Bill Worzel
Genetics Squared
401 W. Morgan Rd.
Ann Arbor, MI 48108
billw@genetics2.com

Library of Congress Control Number: 2007937580

ISBN-13: 978-0-387-76307-1 e-ISBN-13: 978-0-387-76308-8

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Contents

Contributing Authors

Arpit Arvindkumar Almal is an evolutionary engineer at Genetics Squared, Inc., a computational discovery company (aalmal@umich.edu).

Nate Barney is a Programmer in the Computational Genetics Laboratory at Dartmouth Medical School (nate.barney <nate.barney@Dartmouth.edu>).

Ying Becker is Vice President, Advanced Research Center, State Street Global Advisors, Boston, MA USA (Ying_Becker@ssga.com).

Stuart W. Card is a PhD candidate in the Dep't of Electrical Engineering and Comp. Sci. at Syracuse U. (cards@ntcnet.com).

Peng Fei is a quantitative research analyst for the advanced research center at State Street Global Advisors (SSgA), the investment management arm of State Street Corporation, Boston, MA (peng_fei@ssga.com).

Georges Gielen is a Professor of Electrical Engineering Katholieke Universiteit Leuven, Belgium (georges.gielen@esat.kuleuven.be).

Erik D. Goodman is Professor of Electrical and Computer Engineering and of Mechanical Engineering at Michigan State University (goodman@egr.msu.edu).

Gregory S. Hornby is a Project Scientist with the University of California Santa Cruz, working at NASA Ames Research Center in Moffett Field, CA (hornby@email.arc.nasa.gov).

Lee W. Jones is a researcher at Genetic Programming, Inc. in Mountain View, CA (lee@genetic-programming.com).

Michael F. Korns is Chief Technology Officer at Investment Science Corporation (mkorns@korns.com).

Mark E. Kotanchek is Chief Technology Officer of Evolved Analytics, a data modeling consulting and systems company (mark@evolved-analytics.com).

John R. Koza is Consulting Professor at Stanford University in the Department of Electrical Engineering (koza@stanford.edu).

Duncan MacLean is co-founder of Genetics Squared, Inc., a computational discovery company working in the pharmaceutical industry (dmaclean@acm.org).

Trent McConaghy is Chief Scientific Officer at Solido Design Automation Inc., as well as PhD candidate at ESAT-MICAS, Katholieke Universiteit Leuven, Belgium (trent_mcconaghy@yahoo.com).

Chilukuri K. Mohan is Professor of Electrical Engineering and Computer Science at Syracuse University (ckmohan@syr.edu).

Jason H. Moore is the Frank Lane Research Scholar in Computational Genetics and Associate Professor of Genetics at Dartmouth Medical School (Jason.H.Moore@Dartmouth.edu).

Gearoid Murphy is a Ph.D. student of Artificial Intelligence in the Biocomputing and Developmental systems research group of the University of Limerick, Ireland (gearoid.murphy@ul.ie).

Pieter Palmers is a PhD candidate at the Microelectronics and Sensors (ESAT-MICAS) Lab at Katholieke Universiteit Leuven, Belgium (pieter.palmers@esat.kuleuven.be).

Rick Riolo is Director of the Computer Lab and Associate Research Scientist in the Center for the Study of Complex Systems at the University of Michigan (rlriolo@umich.edu).

Ronald C. Rosenberg is Professor of Mechanical Engineering at Michigan State University (roserber@egr.msu.edu).

Conor Ryan is Senior Lecturer in the Department of Computer Science and Information Systems at University of Limerick, Ireland where he leads the Biocomputing and Developmental Systems Group (conor.ryan@ul.ie).

Guido F. Smits is a Research and Development Leader in the Modelling Group within the Engineering and Process Sciences R&D Organization of the Dow Chemical Company (gfsmits@dow.com).

Terence Soule is an Associate Professor of Computer Science at the University of Idaho, Moscow, ID (tsoule@cs.uidaho.edu).

Michiel Steyaert is a Professor of Electrical Engineering Katholieke Universiteit Leuven, Belgium (michiel.steyaert@esat.kuleuven.be).

Leonardo Vanneschi is Assistant Professor at the University of Milano-Bicocca (Italy), in the Department of Informatics, Systems and Communication (D.I.S.Co.) (vanneschi@disco.unimib.it).

Ekaterina Vladislavleva is a Ph.D candidate in the Department of Operations Research of Tilburg University and in the R&D Organization of the Dow Benelux B.V., Netherlands (CVladislavleva@dow.com).

Bill C. White is a Senior Programmer in the Computational Genetics Laboratory at Dartmouth Medical School (Bill C. White <Bill.C.White@Dartmouth.edu>).

Bill Worzel is the Chief Technology Officer and co–founder of Genetics Squared, Inc., a computational discovery company working in the pharmaceutical industry (billw@genetics2.com).

Preface

The work described in this book was first presented at the Fifth Workshop on Genetic Programming, Theory and Practice, organized by the Center for the Study of Complex Systems at the University of Michigan, Ann Arbor, 17-19 May 2007. The goal of this workshop series is to promote the exchange of research results and ideas between those who focus on Genetic Programming (GP) theory and those who focus on the application of GP to various real-world problems. In order to facilitate these interactions, the number of talks and participants was small and the time for discussion was large. Further, participants were asked to review each other's chapters *before* the workshop. Those reviewer comments, as well as discussion at the workshop, are reflected in the chapters presented in this book. Additional information about the workshop, addendums to chapters, and a site for continuing discussions by participants and by others can be found at http://cscs.umich.edu/gptp2007.

We thank all the workshop participants for making the workshop an exciting and productive three days. In particular we thank all the authors, without whose hard work and creative talents, neither the workshop nor the book would be possible. We also thank our keynote speakers PZ Meyers, Associate Professor of Biology at the University of Minnesota, Morris, and Wolfgang Banzhaf, Chair and Professor of Computer Science at the Department of Computer Science of Memorial University of Newfoundland, Canada. Both keynotes delivered thought-provoking talks comparing and constrasting natural biological systems to Genetic Programming and to Artifical Evolution in general, all of which inspired a great deal of discussion among the participants.

The workshop received support from these sources:

- The Center for the Study of Complex Systems (CSCS);
- Third Millennium Venture Capital Limited;
- Michael Korns, Investment Science Corporation.
- State Street Global Advisors, Boston, MA;

- Biocomputing and Developmental Systems Group, Computer Science and Information Systems, University of Limerick;
- Christopher T. May, RedQueen Capital Management;
- Dow Chemical Corporation, Midland, MI; and
- Genetics Squared, Inc, Ann Arbor, MI.

We thank all of our sponsors for their kind and generous support for the workshop and GP research in general.

A number of people made key contributions to running the workshop and assisting the attendees while they were in Ann Arbor. Foremost among them was Howard Oishi, assisted by Sarah Cherng and Mike Bommarito. After the workshop, many people provided invaluable assistance in producing this book. Special thanks go to Sarah Cherng, who did a wonderful job working with the authors, editors and publishers to get the book completed very quickly. Thanks to William Tozier for assisting in copy-editing some of the chapters. Valerie Schofield and Melissa Fearon's editorial efforts were invaluable from the initial plans for the book through its final publication. Thanks also to Deborah Doherty of Springer for helping with various technical publishing issues. Finally, we thank Carl Simon, Director of CSCS, for his support for this endeavor from its very inception.

RICK RIOLO, TERENCE SOULE AND BILL WORZEL

Foreword

It was a great joy for me to be invited to the 5th Genetic Programming Workshop on Theory and Practice, held in May 2007 in Ann Arbor. The organizers are to be congratulated to a well conceived event. The Center for the Study of Complex Systems (CSCS) at the University of Michgan was a fabulous host again this year. Much as in earlier years, participants at the workshop are a unique blend of theoreticians and practitioners in GP, and the workshop is an ideal place to move forward with ideas as both streams fertilize each other.

Although I did participate at this workshop in earlier years, I was honoured this year to give a keynote speech, along with Developmental Biologist, Dr. PZ Myers, from the University of Minnesota, Morris campus.

I was particularly impressed this year by the dedication to community progress. Frequently it was said that ideas introduced and examined in particular algorithmic variants could and should be included in all tools for GP. Among those particularly mentioned this year were information-theoretic measures of fitness and diversity in a populations, as well as recipes like restart of runs and random sampling of fitness cases.

As a result of this community effort, the GPTP workshop has developed into a driving force for progress in GP. Some participants were already planning their prospective next year's contribution, with progress anticipated from integration of new ideas discussed at this year's workshop. If such momentum can be maintained, surely GPTP workshops will go down as some of the most influential events in the history of Genetic Programming.

The keynote speakers this year were selected with an eye on the impact of biological discoveries in Evolutionary Computation. Genetic Programming, as much as this seems strange, happens to be the closest algorithmic incorporation of natural evolution. It offers a rich set of by-products in its behavior, which baffle practitioners, yet are to be expected if one compares the algorithms to their natural counterparts.

A lot could be adopted from what Biology has learned over the past 20 years, and from the feedback I received on my talk there seems to be a general recognition that learning from Biology will infuse further innovations into our field and propel Genetic Programming forward in the coming years.

In my opinion, our algorithmic implementations have just started to explore the power of evolution (and development), and there is much more to be found in the coming years. I went away from GPTP reinvigorated and wish the reader the same from studying the contributions in this collection of talks.

Wolfgang Banzhaf, Head
Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada
July, 2007

Chapter 1

GENETIC PROGRAMMING: THEORY AND PRACTICE

An Introduction to Volume V

Terence Soule[1], Rick L. Riolo[2] and Bill Worzel[3]
[1]*Department of Computer Science, University of Idaho, Moscow, ID;* [2]*Center for the Study of Complex Systems, University of Michigan;* [3]*Genetics, Squared, Ann Arbor MI.*

In 2003 the Center for Complex Studies (CSCS) of the University of Michigan organized the first Genetic Programming Theory and Practice Workshop to bring together practioners and theorists to bridge the gap between what practicioners were doing and what theorists were studying. In the introductory chapter of that volume the authors described Genetic Programming (GP) as "a young art that is just beginning to make its way into applications programming in a serious way" (Riolo and Worzel, 2004). Five years later GP is still a relatively young field, but it is rapidly maturing and has produced a substantial track record of significant successes on large-scale, real world applications. However, these successes are still generally achieved by researchers with significant expert knowledge of GP; successful application of GP to large scale problems remains out of the reach of novices. The lack of a rigorous, detailed theory that is mature enough to guide the development of GP systems to solve specific problems contributes to this weakness. Thus, there remains an on-going need to both strengthen theory and to keep it closely tied to the practice of GP. The Genetic Programming Theory and Practice Workshops remain focused on this goal.

The Fifth annual Genetic Programming Theory and Practice Workshop, supported by Third Millennium, State Street Global Advisors (SSgA), Michael Ko-

rns, Investment Science Corp, the Biocomputing and Developmental Systems Group, CSIS of the University of Limerick, Christopher T. May, Red Queen Capital Management, Dow Chemical Company, and Genetics Squared, consisted of presentations on the latest advances in the theory and application of GP, mixed with lively discussions on how to further advance the field. The discussion and the corresponding papers in this volume cover two broad research areas: understanding the foundations of GP and the development and refinement of advanced techniques to improve the performance of GP generated solutions on real world problems. These two areas lead to a question that was seen as important by both the theorists and practicioners at the workshop: what is the future of GP?

Despite the many recent advances made in the difficulty of the problems tackled and in the quality of solutions generated, Genetic Programming is still not well understood from a theoretical perspective. The papers in this volume address three fundamental questions: (1) How particular evolutionary algorithms, with the various specific mechanisms they include, affect diversity; (2) How genetic structures influence the evolutionary process; and (3) How the evolutionary process takes advantage of available information. By addressing these questions, the papers in this volume help move us toward a fuller understanding of how GP works.

A number of advanced techniques for improving GP received considerable interest from the practitioners, each of which is represented in one or more of the following chapters. Both completely novel and previously published, but under-explored, techniques were presented. These techniques include: (1) fitness and age layered populations; (2) code reuse through caching, archives, and run transferable libraries; (3) Pareto optimization; (4) pre- and post-processing; and (5) the use of expert knowledge to guide the GP. In addition, the theoreticians suggested several approaches that have previously received less attention, including negative slope coefficients, information theoretic analysis, and ensemble techniques. As in recent GPTP Workshops, the consensus was that without these, or similar, advanced techniques GP is unlikely to be sufficiently efficient for most large-scale applications.

Collectively this workshop and the papers in this volume both ask about and suggest the directions in which GP will develop. Here three fairly specific questions emerge. First, how can we develop GP tools that would be useful to the non-GP-expert? Although GP has matured to the point where it can be successfully applied to a wide range of large-scale, real world problems, success still requires significant knowledge of GP. In particular, knowledge of advanced techniques that have been specialized for particular application domains is often necessary to design a successful system. Unless GPs accessibility to non-GP-specialists can be significantly expanded it will be forced to remain a little used technique. Second, how do we address the almost exponential increase

in the number of seemingly effective, but often poorly understood, techniques for improving GP performance? Without a practical method to understand and choose between this proliferation of techniques, even someone well versed in the field is likely to be overwhelmed by the available options. Third, what will GP systems look like in the future? Recent research in evolutionary theory, genetics, and developmental biology, have radically advanced our understanding of biological genetics and evolution. Should GP undergo a similar radical change to reflex this expanded understanding?

1. Exploring the Foundations of Genetic Programming

Many of the presentations addressed issues at the foundation of genetic programming. Primary among these issues was the relationship between genetic programming and natural evolution. How closely should genetic programming be to what is currently known about biological evolution? Other foundational issues that were addressed included the role of diversity in the evolutionary process, how information is processed in evolutionary systems, and how to improve the scalability of genetic programs. These are the issues that are likely to influence the basic form of genetic programming in the future.

Each of the two keynote presentations at the Workshop addressed, more or less directly, the biological metaphor that is the foundation of GP. On day one PZ Meyers, Associate Professor of Biology at the University of Minnesota, Morris, presented an overview of developmental biology and discussed how the development of organisms interacts with the evolutionary process. His major point was that in biology an organism's developmental process can dictate which evolutionary changes are relatively easy to achieve and which are practically impossible. In contrast, the typical GP system, with a few notable exceptions, leaves out development. Including a developmental stage as a fundamental part of GP could significantly alter the possible evolutionary trajectories of evolving individuals, expanding exploration in some directions and constraining it in others, potentially significantly improving their evolvability.

On day two Wolfgang Banzhaf, Professor and Chair of Computer Science at the Department of Computer Science of Memorial University of Newfoundland, Canada, proposed an even more radical change to the basic GP field: moving from the current approach, which he termed Artificial Evolution, to a new paradigm, Computational Evolution. He argued that the current approach, although often successful, is based on "a restricted and outdated understanding of natural evolution" and thus will always be limited in what it can achieve. In particular, he presented a number of specific instances where Artificial Evolution, as it is commonly practiced and applied, falls far short of the richness of natural evolution. For example, Banzhaf pointed out that most current systems lack a developmental stage, a genotype-to-phenotype mapping, reverse infor-

mation, feedback, and regulatory links, and have fixed and fairly limited fitness requirements. He argued that including these missing components that are now known to be critical in natural evolution will lead to computational evolutionary systems whose power and promise is equivalent to that of natural evolution and that far exceed today's more limited "Artificial Evolution". Although most participants were impressed by the scope of Banzhaf's vision there was also a general agreement that for current applications—designing circuits, solving symbolic regression problems, and similar problems—the benefits of invoking the full power of natural evolution is unlikely to be worth the effort, especially since we don't have a good understanding of how simple GP systems work, let alone more complex ones.

Thus, both keynotes present ways in which the fundamental GP algorithm could be radically modified to more closely mirror the current state of biological knowledge. In contrast, in Chapter 9 Almal, MacLean, and Worzel argue that some of the behaviors we see in GP are more like natural systems than we might expect, given that GP systems are far simpler than natural evolutionary systems. Specifically, they challenge the traditional assumption that in simple GP the genotype and phenotype are the same. They argue that in all forms of GP, as in natural evolution, selection applies to the phenotype and genetic variation applies to the genotype. Therefore, an individual should be viewed as having a distinct genotype and phenotype (one subject to genetic variation and one subject to selection) even if the two are syntactically identical.

This presumed genotype/phenotype distinction has a number of important implications. It suggests that although GP may be limited in its exploration of paths through genotype space this does not necessarily limit its exploration of fitness space, which is tied to the phenotype. For example, this suggests that structurally distinct subpopulations with similar fitness may evolve. The low frequency of viable changes near the root of an individual imposes an effective constraint on the evolutionary search: most changes near the root are lethal, and the occasional viable change near the root has a very low likelihood of being reversed, making it likely that the offspring of a these novel, but viable individuals will lead to a new subpopulation. Based on this and other examples, Almal etal. conclude that the evolutionary behavior in GP systems is much richer, and more similar to natural evolution, than was previously believed.

In Chapter 3 Murphy and Ryan revisited a fundamental issue in GP (and other evolutionary techniques): how to control the rate of convergence of the population and how the rate of convergence influences the search for novel solutions. They use a number of straightforward experiments to argue that the simple methods of diversity maintenance currently in widespread use are likely to become increasingly ineffectual, because as problem difficulty increases the computational needs of those methods will outpace the availability of cheap processing power. They then introduce several related methods for maintaining

diversity based on hereditary information. While more complex than current approaches, these new techniques are shown to be significantly more effective than current ones and do not require the calculation of genotypic distances—a time consuming step that has led to the failure of many other diversity maintenance techniques.

In Chapter 6 Card and Mohan also analyze diversity in GP in terms of the mutual information contained within the members of a population. They introduce three fundamental principles to guide the use of mutual information during evolution:

1. mutual information between the target and models should not decrease in the population;
2. mutual information between the target and models should concentrate in fewer individuals;
3. mutual information between the target and models should be "distilled" from the inputs, leaving behind their excess entropies.

Based on these principles they developed and tested a number of ways of using mutual information to improve the evolutionary process. Their general approach is to build ensemble solutions out of sets of evolved solutions, incorporating the mutual information measure in the fitness calculation of individuals and in the parent selection step. These changes mitigated several of the traditional problems of GP such as premature convergence and the early loss of needed building blocks.

In Chapter 2 Vladislavleva and Smits take a different tack to the problem of optimizing information use in GP. They found that using subsets of the full data set during fitness evaluation ("fitness softening") in a symbolic regression problem significantly improved performance, efficiency, and the production of minimal expression trees. Their most successful approach was to start by evaluating individuals in a small subset (10%) of the entire data set with correspondingly large population sizes and to gradually increase the subset size used for evaluation while decreasing the population size, while maintaining a constant number of evaluations per generation. In early generations the smaller data sets act as a limit on the amount of information available to the GP. They carefully analyze the potential benefits of fitness softening and empirically show that it significantly outperforms standard GP.

Vanneschi addresses a fundamental problem in GP in Chapter 7, in which he proposes a method to choose optimal, or near optimal, parameters for a GP analysis. Vanneschi explores the use of negative slope coefficients to predict problem difficulty *as a function of parameter choice* in GP. Clearly the best choice of parameters is the one that minimizes problem difficulty. The slope coefficient is an average measure of the fitness values of individuals against the

fitness of their neighbors. The more negative the slope, the harder the problem. Traditional slope correlation difficulty measurements attempt to apply a single measure to the entire set of fitness slope data and thus may miss sub-regions of the search space that could confound a GP. The major advance introduced by Vanneshi is to partition the fitness-slope data as a function of fitness, so that separate coefficients are calculated for separate "bands" of fitness. Problem difficulty evaluation is based on the band with the most negative slope, because that indicates a deceptive region within the search space.

Using three important pharmaceutical applications he shows that the Negative Slope Coefficient does predict problem difficulty for a given set of GP parameters. Vanneschi is careful to point out that finding the optimal strategy for choosing fitness points and for apportioning them to accurately explore the search space remain important research areas. However, even without optimal choices it is clear that the technique is a very promising, low computational cost, approach for optimizing GP parameters.

One of the most critical issues in GP is how to design systems that scale well. In Chapter 8 Hornby argues that achieving scalability will require evolving designs with greater structural organization, specifically with the traits of modularity, regularity, and hierarchy. Metrics are introduced to measure these traits. Five representations that favor these traits to different degrees are tested. It is found that the representation that most favors all three (modularity, regularity, and hierarchy) is both the most successful and the most scalable. Although most clearly applicable to design problems, where successful solutions are likely to have clearly defined design traits the ideas presented in the paper should be applicable to a wide range of problems.

2. Advanced Techniques for Improving Performance

In Chapter 2 Vladislavleva and Smits set the tone for the practitioners with the statement "Better solutions faster - is the reality of the industrial modeling world''. As befits this goal much of the research presented at the GPTP-2007 Workshop is devoted to specific techniques for improving performance on practical applications. Some of these techniques are novel, such as Vladislavleva and Smits' fitness softening (Chapter 2) and Murphy and Ryan's hereditary based diversity maintenance (Chapter 3). Others are previously published, but not well understood, techniques and some involve combining several techniques into one hybrid method. Many of these advanced techniques are aimed at specific application domains, but others are more general, possibly applicable to any GP system.

Significantly, a focus of many of the proposed techniques is increasing the trustability of the evolved solutions. Because GP is a stochastic process with no inherent guarantee of performance, the quality of solutions generated over

multiple trials by the same GP system can vary considerably. This makes it unwise to apply GP generated solutions directly to critical problems without significant human oversight. Overcoming this weakness of GP is clearly a necessary step to expanding GP's usefulness.

In Chapter 15, Becker, Fox, and Fei of State Street Global Advisors describe their work on symbolic regression problems drawn from financial markets. Their goal was to evolve a function that both successfully picks stocks for investment and is simple enough to be human-understandable. The requirement that a human can understand an evolved solution, as compared to being forced to blindly apply it, was frequently raised by the practitioners. This is related to the trustability of the solutions; corporations are generally not willing to risk millions of dollars on an evolved program that can't be understood. Therefore, producing formulas that a market expert can understand and approve expands the applicability of GP. In addition, simpler programs generally suffer less from over fitting.

An additional complication for Becker et al. is that their problem requires a solution that balances investment metrics of risk and return. They compared three multi-objective algorithms in an attempt to evolve solutions that performed well on all of the metrics and were simple enough to be human readable. They found that simplest of the three approaches, the constrained fitness function approach, led to the best overall performance while producing understandable solutions.

In Chapter 13, Kotanchek and Vladislavleva directly address the problem of building trustable models. Their major contribution is the use of ensembles as a method to generate trust. Independent runs of a Pareto-aware GP are used to generate a large set of models, from these they generate an ensemble consisting of solutions with the lowest estimated error correlation. A significant benefit of their approach is that it allows the full use of the data set during training, without the need for hold-out data. It is also important in that it is part of a relatively small body of research that directly addresses the issue of the trustworthiness of GP evolved solutions and should be of considerable interest to readers who deal with high risk problems without a safe environment for testing evolved solutions.

An alternative approach to evolving trustable solutions is presented in Chapter 10 by McConaghy, Palmers, Gielen, and Steyaert. The problem they address is the design of analog circuits. They introduce a novel evolutionary algorithm, MOJITO, that uses a library of known, trusted circuit designs in the evolutionary process and a multi-objective fitness function that favors both better circuit performance and trustworthiness as measured by the number of steps from known, trusted circuits. Users can adjust the balance between novelty and trustworthiness as necessary to meet the requirements of a particular circuit design problem. Although the specific application is closely tied to circuit design

the authors make it clear how the techniques can be applied to a wide variety of problems where trustworthiness is a key goal.

A closely related problem is the design of robust circuits. In general, a robust system is more trustable because its behavior is less variable. In Chapter 11 Peng, Goodman, and Rosenberg apply GP to the problem of designing low-base filters with fifth-order Bessel characteristics. They show that robust-by-multiple-simulation (RMS), in which each design is evaluated multiple times with randomly perturbed parameters, produces circuit designs with equivalent performance and significantly higher robustness than circuits designed without RMS. Further, they show that the inclusion of active circuit components can lead to smaller circuit designs with equivalent performance and robustness as circuits evolved with only passive components. These results both broaden the success of GP in evolving circuits and, more importantly, show that GP can be used to evolve robust, and thus more trustable, circuits.

In Chapter 12, Soule and Heckendorn consider the problem of evolving ensembles, specifically multi-agent systems. Their concerns are how to balance individual performance versus cooperation and how to scale the evolutionary algorithms to train larger teams, rather than trustworthiness. They argue that optimal ensemble performance occurs when individuals both perform well and cooperate well, and that traditional techniques to evolve teams typically favor one of these two goals over the other. A new class of algorithms, termed Orthogonal Evolution of Teams (OET), is introduced and compared to more traditional approaches. OET alternates between treating the set of evolving individuals as a single population of N teams and treating it as N independent populations of individuals. Thus, there is direct evolutionary pressure for both individual performance and cooperation between the individuals on the same team.

The test problem used by Soule and Heckendorn consists of training a heterogeneous team of robot agents to investigate an unknown environment. The results show that OET balances individual performance and team cooperation more successfully than current algorithms leading to better overall performance. Additionally, teams evolved with OET can be scaled up more successfully than teams evolved using other approaches. This is likely to be a significant result as successes with autonomous robotics leads to larger and large ensembles.

In Chapter 4, Korns empirically tests a number of advanced hybrid algorithms on very large scale symbolic regression problems. Korns' test problems were some of the largest discussed in the workshops—each test function included 20 variables and 1,000,000 sample data points, making them good representatives of the scale of real-world problems. The tested techniques include hybrids of GP with particle swarm, abstract grammars with trees, multiple islands with boosting, hybrid fitness functions, and context aware crossover. The results are extremely promising, with nearly perfect results on all but the hardest of

the test functions. Korns concludes that "from a rigorous corporate perspective, [GP] is almost ready for industrial use on large scale, time constrained symbolic regression problems."

In a nice parallel to Korns' conclusion that GP is almost ready for industrial use, Moore, Barney, and White (Chapter 4) are working to make GP more accessible to the same non-experts whose problems GP is expected to solve. Beginning with the assumption that "[GP] is not a push-button problem-solver" and that "the 'vanilla' or basic GP algorithm is not appropriate for solving complex problems such as those in the biomedical sciences" they do not attempt to build a program that is either universally accessible or applicable to any problem type. Instead they created a program, Symbolic Modeler (SyMod), for applying GP to bioinformatics problems by facilitating geneticist-bioinformaticist-computer interactions. SyMod was designed with a number of requirements:

- it should be platform-independent.
- it should include a user-friendly graphic-user interface (GUI) in addition to a simple command-line interface that could be scripted.
- it should produce publication quality graphical output in the GUI.
- it should include a number of configuration options for the expert user.
- it should be able to generate and use expert knowledge that can be used to help guide the algorithms.
- it should limit data over-fitting.

The current version of SyMod appears to be quite successful. It's ability to generate useful solutions is significantly improved through a number of specialized techniques. Perhaps most importantly, over-fitting is limited through cross-validation and by using populations of full trees of the same size, so that competition only occurs between trees of the same size. In addition, expert knowledge in the form of expected or likely interaction between some input variables is incorporated into the analysis through sensible initialization of the initial population, multi-objective fitness functions, and selection.

Most GPTP-2007 attendees agreed with Moore etal. that given the current state of the field, building a GP system for a specific application domain and not trying to make the system capable of functioning independently of a computationally experienced user, is the most promising approach when applying GP to large, complicated "real-world" problems. In addition, there was considerable interest in seeing similar software built for other active application domains.

In one of the most unique applications of GP Reynolds, Ali, and Franzel used a cultural GP algorithm to simulate the growth of ancient Mexican cities (Chapter 14). Specifically, their goal is to identify the sites most likely to be

settled at various times over the 3000 year history of Monte Alban. Extensive archaeological research has identified the sites in and around Monte Alban that were settled during a number of periods throughout its 3000 year history. A number of data mining techniques were applied to these records to build classification trees. The results from the most successful technique, J48 Decision Trees, were broken down to generate rules to serve as building blocks for the GP. A cultural GP was then used to generate general, understandable rules for predicting likely occupation sites. In addition to its intrinsic value, this paper is useful in understanding the breadth of problems to which GP can be applied.

3. The Future of Genetic Programming

The papers in this volume reflect GP's growth from a novel technique to a maturing field with a track record of significant practical successes. They describe where and how GP has been applied with the most success (symbolic regression, financial modeling, and design) and address the techniques that made these successes possible, including age and fitness layering, archives, multi-objective optimization, pre- and post- processing, and the application of expert knowledge. However, GP's success, as reflected by the papers in this volume, has also engendered three significant questions regarding the future of GP:

1. How can GP be made more accessible to the non-expert? Although GP has matured to the point where it can be successfully applied to a wide range of large-scale, real world problems, success still requires significant knowledge of GP in order to design a successful system. Unless GP's accessibility to non-specialists can be significantly expanded it will remain a little used technique. However, there was also a conflicting general consensus that advanced techniques, such as those presented in this volume, and a certain amount of trial and error by an experienced GP practitioner are still necessary to solve difficult problems. This makes it difficult to design a GP system that a novice could use to effectively solve real world problems. Several participants were concerned that tools that allowed researchers to blindly apply GP would likely lead to sub-optimal results, in effect "poisoning the well" by creating a false impression that GP was not a broadly useful technique. Thus, there is a significant tension between the desire to expand accessibility and the need for expert knowledge of advanced techniques.

Moore, Barney, and White's human-human-computer model addresses this tension by assuming that a GP expert will be part of the team, but will use software tools that are more understandable and accessible to the non-GP experienced members of the team.

Several approaches for successfully making GP more accessible to the non-expert were proposed:

1. Building special purpose GP engines targeted at particular applications areas such as circuit design, symbolic regression, and bioinformatics applications, rather than producing general purpose engines.

2. "Hiding" GP within another, broadly used, application. For example, a GP engine to perform symbolic regression within Excel macro or to perform circuit design and optimization within common circuit simulation programs.

3. Replacing standard GP parameters, such as population size and number of generations, with less technical parameters, such as desired time to solution. The GP engine would then automatically use small sampling runs to automatically choose necessary parameters.

4. Building GP tools specifically designed for human-human-computer interactions involving an application specialist, a computer scientist with experience with GP, and the computer. This approach is specifically discussed in Chapter 5 where Moore, Barney, and White discuss their experiences building a GP tool for bioinformatics analysis based on their human-human-computer interaction model.

2. Is it more important to understand existing techniques or to continue to invent new ones? The field of GP now includes a plethora of advanced techniques that significantly improve performance on difficult problems. And, as noted above, these techniques are often a prerequisite for obtaining satisfactory solutions. However, most of these techniques were developed from intuition and experimentation. We lack a firm theoretical foundation that explains how they work, when they will or will not work, or how to further improve them. Without such a foundation a practitioner is forced to guess and use trial-and-error to figure out which techniques and which settings for that technique will be most effective for a particular application. Worse, a non-expert is left almost completely in the dark, making the problem of designing widely accessible GP systems that much harder. In addition, when we create new techniques and combine them with existing ones, the complexity of some of these techniques and the interactions between techniques make it difficult to confidently pick the source of any observed improvements in GP performace. Thus while it is certainly useful to continue to explore new techniques and new combinations of techniques, both to exploit the short-term performance advantages they provide and to provide new "data points" for understanding GP behavior, the field needs to continue to devote sufficient effort to furthering our understanding of existing techniques.

3. Which biological metaphors can be used to guide design and understanding of GP? It is now clear that the full range of mechanisms and features influencing biological evolution is much, much wider than what is incorporated

in even the most complex current GP system, let alone the simplest, "plain vanilla" GP system.

It seems very likely that using a richer biological metaphor could significantly increase the robustness and evolvability of current GP systems, improving the systems' ability to adapt to the requirements of particular problems and to changing conditions. This in turn would result in solutions that are better adapted, more robust, and more responsive to changing environments.

Unfortunately, richer biological metaphors are also like to produce a GP systems that are considerably slower and that are likely to be orders of magnitude more difficult to understand. In addition, adding more complexity may run contrary to the goal of an understandable system that a non-expert can successfully apply. However, for some applications, the gain in robustness and adaptability may be worth the cost associated with requiring GP-experts to be part of a research team.

With maturation comes growing pains. GP's growing pains arise from the tensions between the desire to engineer a better algorithm, the need for a firm theoretical foundation to guide future research, and the goal of making GP widely accessible. Although not completely incompatible, it is clear that these goals will be difficult to achieve simultaneously. It is our hope that the GPTP Workshop and the papers within this volume will help reach this end.

References

Riolo, Rick L. and Worzel, Bill, editors (2004). *Genetic Programming Theory and Practice*, volume 6 of *Genetic Programming*, Ann Arbor, MI, USA. Kluwer.

Chapter 2

BETTER SOLUTIONS FASTER: SOFT EVOLUTION OF ROBUST REGRESSION MODELS IN PARETO GENETIC PROGRAMMING

Ekaterina Vladislavleva[1], Guido Smits[2] and Mark Kotanchek[3]
[1]*Tilburg University, Tilburg, the Netherlands;* [2]*Dow Benelux B.V., Terneuzen, the Netherlands;* [3]*Evolved-Analytics,LLC, Midland, MI, USA.*

Abstract "Better solutions faster" is the reality of the industrial modeling world, now more than ever. Efficiency requirements, market pressures, and ever changing data force us to use symbolic regression via genetic programming (GP) in a highly automated fashion. This is why we want our GP system to produce simple solutions of the highest possible quality with the lowest computational effort, and a high consistency in the results of independent GP runs.

In this chapter, we show that genetic programming with a focus on ranking in combination with goal softening is a very powerful way to improve the efficiency and effectiveness of the evolutionary search. Our strategy consists of partial fitness evaluations of individuals on random subsets of the original data set, with a gradual increase in the subset size in consecutive generations. From a series of experiments performed on three test problems, we observed that those evolutions that started from the smallest subset sizes (10%) consistently led to results that are superior in terms of the goodness of fit, consistency between independent runs, and computational effort. Our experience indicates that solutions obtained using this approach are also less complex and more robust against over-fitting.

We find that the near-optimal strategy of allocating computational budget over a GP run is to evenly distribute it over all generations. This implies that initially, more individuals can be evaluated using small subset sizes, promoting better exploration. Exploitation becomes more important towards the end of the run, when all individuals are evaluated using the full data set with correspondingly smaller population sizes.

Keywords: partial evaluation, archiving, budget allocation, goal softening, ranking, robust solutions, nonlinear regression, ordinal optimization,

1. Introduction

Symbolic Regression via genetic programming is a stochastic iterative search technique that automatically generates a rich set of data-driven symbolic models expressing the response variable of interest as an analytical function of given input variables. Such genetic programming system exploits a population of symbolic models (= expressions = individuals = formulae) to cover multiple points of the search space simultaneously. At each iteration step the population of alternatives is evaluated, and a subset of 'best' solutions discovered so far is selected to be optimized and produce the next, hopefully better generation.

The search space of a real-life symbolic regression problem is huge. The multidimensionality, noise, and inaccuracy of data, the nature-driven complexity of underlying relationships all contribute to make the search inherently difficult. The lack of any significant structure makes the navigation through this search space intrinsically hard.

Both increasing the size of the population and the length of the GP run will escalate computational requirements. Besides, the process can still suffer from bloat and premature convergence phenomena. The problem of bloat (Langdon and Poli, 2002), i.e. unnecessary growth of size of the solutions with no considerable improvements in fitness, is diminished by a Pareto-centric selection strategy (Smits and Kotanchek, 2004), (Zitzler and Thiele, 1998), (Laumanns et al., 2002), when both the goodness of fit and the expressional complexity of a model are optimized simultaneously.

The problem of premature convergence can be avoided by maintaining the diversity of the population. In our implementation of ParetoGP we use cascades to re-initialize the population completely on a periodical basis while maintaining an archive of solutions, lying near the Pareto front of model complexity and accuracy, see Section 2 and (Smits and Kotanchek, 2004; Kotanchek et al., 2006)).

Even with these modifications the majority of the computational effort is spent on fitness evaluation. One of the natural ways around this problem is to use subsets of data for fitness evaluations. Several groups have tried to address the issue of using parts of data for fitness evaluations without a loss in quality of solutions. Gathercole (Gathercole and Ross, 1994) analyzed various selection schemes for partial fitness evaluation using subsets of data for a number of GP classification problems. This approach was later refined by Teller and Andre, (Teller and Andre, 1997), also for classification problems. Zhang and Cho, (Zhang and Cho, 1998), applied incremental subset selection to the evolution of collective behaviors for multiple robotic agents. We found no references that applied these principles to regression type problems.

While these subset selection schemes were mainly heuristic in nature, there exists a beautiful and simple theory, that, when combined with GP, has the

potential to provide a strong theoretical foundation to these approaches. This theory of Ordinal Optimization (OOpt) was developed by Yu-Chi Ho et. al (Ho, 2000). The ideas of Ordinal Optimization and more specifically of goal softening and ranking, not only fully concur with our thoughts and beliefs in evolutionary search, but couch them in strong language of mathematics and common sense.

The objective of this paper is threefold. First, we would like to further extend and analyse the concept of ordinal ParetoGP. Second, we would like to introduce some additional goal softening to further reduce the cpu-budget while effectively keeping the same quality of the results. Third, we examine the effect of allocating more cpu-budget to exploration without changing the total budget for the total run.

The chapter is organized as follows: In Section 1 we briefly describe our Pareto-based genetic programming system that is used as a reference. In Section 3 we quickly summarize the main ideas of ordinal optimization. In Section 4 we analyse the effects of goal softening introduced by the use of subsets of the data to evaluate individuals in a GP population. Finally, in Sections 5 and 6 we describe and analyse a series of experiments leading to a set of robust settings for Ordinal ParetoGP.

2. Our Pareto-based Genetic Programming System

We have been working with a particular implementation of a genetic programming system, called ParetoGP. The main features that make it different from many other GP systems are an explicit consideration of multiple objectives to guide the search and the use of an archive of promising equations that is maintained during a run (see Table 2-1).

A foundation of every evolutionary system consists of granting higher propagation rights to *better* individuals. A classical regression via GP system uses one objective to decide on the quality of individuals. Since the main purpose of any regression system is to construct an accurate approximation of the observed output as a function of given inputs, accuracy (prediction error) of a GP individual is usually used as a performance measure. We believe that adequacy of a GP individual is of the same importance as the accuracy of its prediction. Therefore, for performance evaluation, we never use the prediction error alone, but always in a combination with a complexity measure.

In all experiments of this chapter only one fitness measure and one complexity measure were used. Fitness was determined as a normalized sum of squared errors between predicted output of a model py and an observed output y:

$$NMSE(y, py) = \frac{1 - MSE(y, py)}{1 + MSE(y, py)} \tag{2.1}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(rscale(y)i - rscale(py)_i)^2 \tag{2.2}$$

$$rscale(z) = \frac{z - \min(z)}{\max(z) - \min(z)} \tag{2.3}$$

The complexity measure of a symbolic model, expressed as a tree-structure, was determined as a sum of nodes in all subtrees of the given tree. We call it expressional complexity of a tree-based GP individual, (Smits et al., 2005). A thorough analysis of the properties of this measure is given in a recent paper of Maarten Keijzer and James Foster, who called it a visitation length, (Keijzer and Foster, 2007).

If two competing objectives are used for evaluating the performance of individuals, the relation of dominance needs to be defined for these individuals in the performance (objective) space. If small values of both objectives are preferred over bigger values (we are minimizing model error and model complexity), then an individual (an alternative) A with objective values (f_1, c_1) is said to dominate an individual B with objective values (f_2, c_2), if $f_1 \leq f_2$, $c_1 \leq c_2$, and either $f_1 < f_2$, or $c_1 < c_2$. In other words, A dominates B, if A is not worse than B in both objectives, and A is strictly better than B in at least one objective.

A set of individuals, non-dominated by any other individuals, forms a set of optimal trade-offs in the given objective space, and is called the Pareto front. A Pareto front in accuracy–complexity space refers to a set of best GP individuals, which should be granted the most propagation rights.

In the current implementation of ParetoGP the archive, i.e. the set of best individuals obtained by a given step of evolution has a predetermined maximum size. We define and update it at every generation with individuals that lie at the Pareto front of the combined populations of the new generation as well as the archive of the previous generation (see Figure 2-1 and Table 2-1).

To prevent inbreeding we re-initialize a population every k generations. By doing this we segment an evolutionary run into a series of *cascades* of k generations each, (Smits and Kotanchek, 2004). Within a cascade the population is allowed to crossbreed with the archive to generate the next population. The archive is maintained during the entire run. For this reason, good solutions reappear in a population very quickly again despite re-initializations.

The performance measure that we typically use to monitor the progress of a GP run is either the fitness of the best individual in the archive or the percentage of area under the Pareto front defined by the fitness (model error) and the complexity. The latter performance metric is more robust and also reflects the balance between fitness and complexity we try to minimize (see (Smits and Vladislavleva, 2006)). In all experiments of this chapter we used the area under

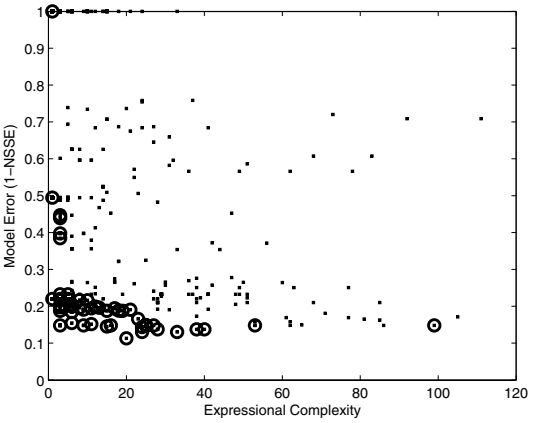


Figure 2-1. **Initializing an archive.** Black dots represent a population of 100 tree-based equations, plotted in the performance space of expressional complexity and accuracy (prediction error). Smaller values of both objectives are preferred over bigger ones. Circles represent the 50 'best' individuals obtained via a non-dominated sorting procedure. These individuals are the 50 'closest' to the Pareto front and will form the archive at the first generation.

Table 2-1. Our ParetoGP algorithm. The function $Generate$ produces $Population(t)$, from $Archive(t-1)$ and $Population(t-1)$ by means of structure changing operators. Best representatives of this set are used by the function $UpdateArchive$ to optimize the archive $Archive(t-1)$. When the iteration process is terminated, the set of solutions is determined by $Archive(t_{last})$, where t_{last} is the last iteration step. Function $UpdateArchive$ uses the union of the old archive and the new population to select a fixed number of models located at the Pareto front in complexity vs. fitness space.

```
t=0;    % generation count
% Initialize Archive
Archive(0)= ∅;
% Initialize population
Population(0)= ∅;
while search is not terminated
  t=t+1;
  % Generate new population from archive models at
  % the previous step
  Population(t) = Generate(Archive(t-1),Population(t-1));
  %Update Archive
  Archive(t) = UpdateArchive(Archive(t-1), Population(t));
end
% Return solutions of a GP run
Archive(t)
```

the Pareto front of the archive at the last generation as a measure for *a posteriori* analysis of the effectiveness of a GP run.

3. Goal Softening and Ranking

The concept of Ordinal Optimization is described extensively in (Ho, 2000). This section is a summary of the main ideas of (Ho, 2000), (Ho et al., 2000), and (Lau and Ho, 1997). The theory rests on two basic tenets:

- It is easier to find a good enough solution with high confidence than the best solution for sure. Such *goal softening* helps to smooth and direct the search.
- It is easier to determine Order than Value, or, in other words, it is easier to determine whether $A > B$ than to determine A and B exactly.

When solving *hard* problems by computationally expensive (e.g. evolutionary) search-based methods, we argue that quickly narrowing down the search for an optimum to a 'good enough' subset of the search space is more important than accurate estimation of performance of potential solutions during the search.

Goal Softening. In real-life problems the true optimum is often unattainable and the compromise is made for 'good enough' solutions.

This substitution of:

- the *best* solution, by a good enough subset of solutions,
- being *sure*, by being confident with high probability,
- getting a *closed form solution*, by an approximate solution, and
- making *rational decision*, by using heuristics,

are all examples of softening the optimization goals.

Order vs. Value. Let us assume we have to quickly select the fastest runner of two. Instead of monitoring them separately for weeks and measuring the speed on various distances, we will let them compete with each other, and will choose the one who arrives at the finish first. In contrast, it is much easier to determine whether $A > B$, i.e. to determine the order, than to exactly estimate the difference between A and B under multiple conditions, i.e. to examine the value. Thus, although there is less certainty associated with the decision based on the ordinal approach, it is obtained at a much lower cost compared with the approach based on exhaustive evaluations.

4. Discussion: How to get better solutions faster by goal softening and emphasis on ranking?

Ordinal ParetoGP – better solutions with about the same effort

Considering the vastness of the search space we can improve the effectiveness of our evolutionary search procedure in either of two ways. One way is to try to get solutions of a similar quality at a lower computational cost. A second way is to try to get solutions of a better quality at the same or similar computational cost. We question whether we need exhaustive fitness evaluations to evolve solutions of a similar quality. If fitness evaluation can be 'softened', to what extent can we then decrease the size of the subsets of the data to perform this evaluation? We also wonder what can be gained in terms of system performance and the cpu-budget, by evaluating more potential solutions at a lower cost and by gradually improving the fidelity of the fitness evaluations in the course of an evolution.

In symbolic regression via GP the bulk of the computational effort is spent predominantly on fitness evaluation of the individuals[1]. In many cases all available records of a given data set are used to determine the fitness of every individual. These fitnesses are then used to rank all models and then decide who gets the right to propagate and who does not. As we emphasized in a previous paper (Smits and Vladislavleva, 2006), it is not really critical what sort of selection system is being used – the essence is that all that is required is an ordering of the equations in terms of their performance (actually what we really need is not an ordering in terms of their current performance but an ordering in terms of their potential to generate even more fit offspring).

We also showed that the effectiveness and the reproducibility of our search can be improved considerably by introducing incomplete fitness evaluations. In the most successful scheme we used partial fitness evaluations on random subsets of the original data set while gradually increasing the subset size and decreasing the population size in consecutive generations. In the settings for this scheme we started with the subset size of 10% of the original size, and increased it to 100% during the first 200 of the total of 250 generations. At the same time the population size was decreased linearly from 1000 to 100 over the first 200 generations. The archive size was kept constant at 100 models at all times. These Ordinal ParetoGP runs outperformed standard ParetoGP runs of a 1000 generations each, both in terms of the final fitness as well as the consistency of 30 independent replicates.

[1]This holds for data sets of medium and large size, which are our main interest.

In the next section we explain the reasons for the improved performance we obtained in (Smits and Vladislavleva, 2006), by examining the influence of partial fitness evaluations on the ranking of individuals and the resulting change in the balance of exploration versus exploitation.

Better solutions because of better exploration. Instead of evaluating every model exhaustively using the full data set, we settle for using a subset of the available data to rank the models. The fact that we use a random subset to estimate the fitness of an individual introduces a certain level of noise in this estimate. By repeatedly selecting different subsets of a given size to determine the fitness of a given individual we can get an idea of what level of uncertainty we introduce by examining the resulting distributions. These distributions will obviously depend on the size of the subset we choose (smaller subsets will cause wider distributions) but will also depend on the individual itself (fitter individuals will have smaller distributions) (see Figure 5-1).

We have observed that the histograms of these distributions can be approximated by a normal distribution with the mean equal to the true fitness of the individual, and the standard deviation inversely proportional to the subset size and to the quality of the model (see Figure 5-2)[2].

When the true fitness distributions of two individuals overlap (see Figure 5-2), we can obviously make an error in taking the decision which is the best of the two. These 'mistakes' in the ranking are the first reason for enhanced exploration. A given individual can outrank another individual in the population but in addition that individual could also outperform and replace other individuals that are already in the archive. Since individuals that would be of lower fitness can now end up in the archive and contribute to creating offspring in the next generation, this increases the level of exploration in the search.

Another reason for better exploration is the fact that the use of subsets allows us to evaluate more models with the same computational budget. If we use subsets of a given size, sampled uniformly at random, they will be different for every generation. These will act as successive screens that every model will need to pass in order to survive long-term. The result is a regularizing effect and an increased robustness of the final models because there is less opportunity for pathologies or learning the noise that may be present in the data.

'About the same effort' because of archive re-evaluations. In the new approach we decided to keep the number of function evaluations per generation constant; therefore, by design, the cpu-budget spent on fitness evaluations of every new population does not change. However, an additional budget is re-

[2]This is true for smaller subset sizes (10-75%). For bigger subset sizes the distribution of model fitness deviates from the normal, and can be modeled as a Weibull distribution

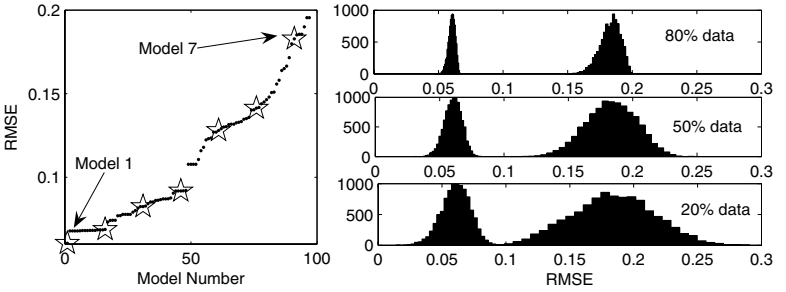


Figure 2-2. The left plot illustrates the sorted fitness values of an archive of a Kotanchek run at the last generation. Seven models (emphasized as stars) are selected from 100 archive individuals for further analysis. The plot at the right size shows the histograms of fitness distributions of model 1 and model 7 (the best and the worst from the selected set of models), computed on 10000 random subsets of 20, 50, and 80% of the training data. We see that the widths of the histograms, i.e. the deviation of the estimated fitness from the true fitness, are inversely proportional to the size of the subset, and to the quality of the individual itself. E.g., model 1, which is the best one in the archive, has the most narrow fitness distribution, which becomes narrower as the subset size increases.

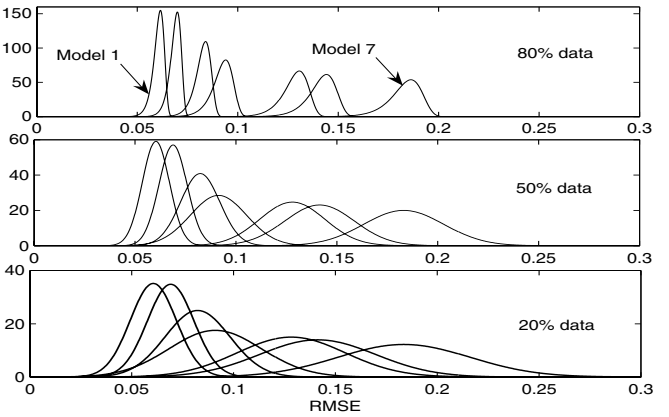


Figure 2-3. Approximated fitness distributions of the seven archive models from Figure 5-1 computed using random subsets of 20, 50, and 80% of the original data set for the Kotanchek problem.

quired due to presence of an archive in ParetoGP. At each generation we create a new archive by merging the old one of the previous generation with the new population and selecting a fixed number of individuals located at the Pareto front in fitness-complexity space.

Since we randomly select a *new* subset to evaluate the population at every generation, in principle, the fitness of the archive needs to be reevaluated using the exact same subset. We considered this necessary, to make sure that we compare apples with apples, when updating the archive for the next step. At each generation the extra computational effort is equal to the product of the archive size and the number of records in the current subset. This implies that the additional effort grows when the subset size goes up during the run. For the case where the subset size increases from 10% to 100%, the archive re-evaluation step causes an increase of the total reference cpu-budget by 45%.

The budget of 145% can be called 'about the same' effort, but the increase is certainly not negligible. This makes us wonder whether we can reduce archive re-evaluations and still get solutions of a similar quality.

Soft ordinal ParetoGP – better solutions with less effort?

Why cpu time can be saved. Given the additional effort required to re-evaluate the archive at every new generation, we can question whether we really need to evaluate models on the same subset to get a good-enough ranking.

To analyse the influence on the ranking of two individuals we have to examine another distribution which is derived from a consideration of the difference in the estimated fitness of both individuals. This second distribution gives a direct estimate of the probability that you will have a wrong estimate of the rank of the two given individuals.

Below, we model the distributions of two different cardinal approaches to estimate the ranking of two individuals – taking the difference of fitness estimates obtained on the same subset, and taking the difference of fitness estimates obtained using different subsets of a similar size (see Figure 2-4).

The results of this analysis are unexpected. Comparing apples with apples gives us the most accurate estimates of the true difference in fitness, and correspondingly in the true ranking of the individuals. Surprisingly, the error in comparing apples to oranges is still small enough to make reasonably accurate decisions on the ranking of individuals even for small subset sizes. This statement seems to be general – we compared pairs of models of similar and different quality, sampled at various stages of the evolution, for various test problems.

Does more exploration lead to yet better solutions?

We assume that the essential elements of creating robust solutions are:

Table 2-2. Parameter settings of reference ParetoGP runs. Note, that the other experiments described in Cases 1–3 will have the same settings except for the population size, and the data set used for fitness evaluations.

Number of independent runs	30
Number of cascades	10
Number of generations per cascade	25
Total number of generations	250 (500 in PGPA)
Population size	100 (200 in PGPB)
Archive size	100
Run Performance measure	Area% under Pareto front
Accuracy measure	$1 - NMSE$ (smaller values preferred)
Complexity measure	Expressional
Population tournament size	5
Archive tournament size	3
Crossover rate	0.95
Mutation rate	0.05
Rate of mutation on terminals	0.3
Function set 1(*kotanchek, tower*)	$+$,$-$,$*$,$/$,e^x,e^{-x}, x^{real}, $x + real$,$x \cdot real$
Function set 2 (*maarten*)	Function set 1 $\bigcup \sin x, \cos x$

- **Abundant exploration**, caused by the use of larger population sizes and a softer selection process.

- **Sufficient exploitation**, caused by selecting potentially good parent models for further propagation. Despite the fact that the fidelity of the selection process is lower due to partial fitness evaluation, the exploitation of good models in the archive is still sufficient, due to the fact that winning models need to survive multiple low-fidelity screens to stay in the game and keep propagating. That said, goal softening by means of partial fitness evaluations introduces new modes of exploitation. It allows bad-but-lucky models to propagate short term, but guarantees that truly good models (which are good on all data points) will survive multiple screens and keep propagating long term.

Since abundant exploration combined with sufficient exploitation is essential for creating robust solutions, why should we insist on having a constant budget per generation? This can be achieved easily by allocating more cpu-budget to the initial part of the run (for exploration) at the expense of budget allocated to the end of run (for exploitation). This will be discussed further in the section on simulation results, Case 3.

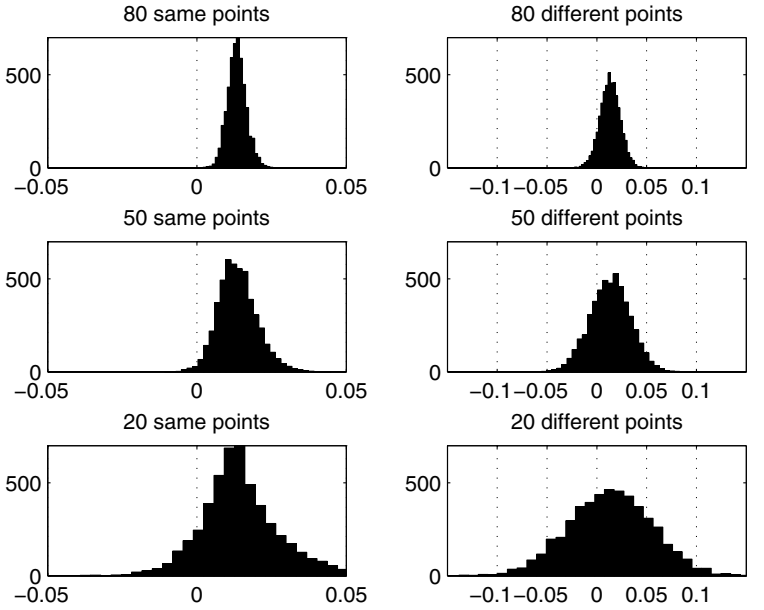


Figure 2-4. The histograms represent the difference in fitness values of the two archive models, calculated 10000 times on the same subset (the left-hand side), and different subsets of the same size (the right-hand side). The models used in this analysis are model 5 and model 6, selected in Figure 5-1. Notice, that the distributions on the left-hand side are much narrower than the ones on the right-hand side. This indicates, that a higher variance in the actual value of the difference in fitness values of the two models is introduced by evaluating them on different subsets. However, not the actual value of this difference is important. Only the sign matters to make a decision about the ranking. Thus, the fraction of the histogram area to the left of zero can be used to derive the probability of making an incorrect ranking. Notice that even in the worst case (the lower-right plot) an error in ranking is made in only 32% of evaluations. This plot corresponds to evaluations on 20% of the original data – an evaluation scheme used only at the beginning of the evolution. Given our commitment to soften the selection goals at the first generation, and only gradually improve the screening towards the end of the evolution – a 30-40% chance to make a mistake in ranking becomes more than appropriate.

Summary. Goal softening comes from:

- the use of subsets instead of the entire data set;
- use of *random* subsets of a given size at every generation; (has a regularizing effect on the models by reducing the chances for over-fitting)
- not re-evaluating the archive models at every generation, and giving them a chance to stay in the archive for a while;
- spending more cpu budget at the beginning of the evolution and less at the end.

There is a synergy of all these principles on guiding the search to better and more robust solutions faster.

5. Results

Experiment Setup

We selected three test problems for our experiments:

- *Maarten* (2.4) problem is drawn from Equation 2.4 and contains 101 records, with inputs sampled uniformly from the range $[0, 10]$,

$$f(x) = x^3 \exp^{-x} \cos x \sin x (\sin^2 x \cos x - 1); \tag{2.4}$$

- *Kotanchek* problem, drawn from Equation (2.5), consists of 100 records, with inputs sampled randomly uniformly from the box $[0, 4] \times [0, 4]$,

$$f(x_1, x_2) = \frac{e^{-(x_2-1)^2}}{1.2 + (x_1 - 2.5)^2}; \tag{2.5}$$

- *Tower* problem is an industrial data set of a gas chromatography measurement of the composition of a distillation tower. The underlying data set contains 5000 records with noise and 23 potential input variables.

The ParetoGP parameters that we used in all experiments unless indicated otherwise are shown in Table 2-2.

The overall aim of all experiments was to get the highest quality results with the largest reproducibility and the lowest computational cost. In this chapter we focus on three case studies:

CASE 1. The first experiment setup is an extension of a setup that we discussed in an earlier paper (Smits and Vladislavleva, 2006). It is based on partial fitness evaluation of GP individuals on a *random* subset of the original

Table 2-3. **Simulation Results for three test problems** The table represents the results of different experiments in terms of the median and the interquartile range of our performance measure over 50 independent runs and the normalized total number of function evaluations per run (cpu budget). The performance measure we use is the percentage of the area under the Pareto front of the archive solutions in complexity vs. fitness space. For all three quality characteristics – median and the IQR of the Pareto front area percentage, and the cpu budget – smaller values are preferred. There are three reference runs: PGP with the population size 100, and 250 generations, PGPA with twice the number of generations, and PGPB with twice the population size of the PGP runs.

Experiment	**Maarten**		**Kotanchek**		**Tower**		BUDGET
	$\tilde{\mu}$	IQR	$\tilde{\mu}$	IQR	$\tilde{\mu}$	IQR	per run
Reference							
PGP	1.81	1.39	2.20	0.735	1.55	0.325	250000
PGPA	1.23	0.93	2.05	0.786	1.39	0.334	500000
PGPB	1.57	1.12	2.06	0.630	1.41	0.334	500000
Case 1							
OPGP10	1.04	0.26	1.86	0.662	1.33	0.209	367169
OPGP20	1.02	0.34	2.02	0.633	1.37	0.368	372486
OPGP40	1.15	0.34	2.10	0.764	1.39	0.300	387703
OPGP60	1.24	0.54	2.18	0.831	1.57	0.362	404590
OPGP80	1.34	0.71	2.30	1.002	1.46	0.422	422122
Case 2							
Q10	1.05	0.31	1.93	0.431	1.32	0.250	273119
Q20	1.20	0.46	1.95	0.382	1.34	0.253	269886
Q40	1.24	0.65	2.18	0.652	1.41	0.285	268003
Q60	1.20	0.80	2.23	0.630	1.56	0.405	267790
Q80	1.76	1.89	2.14	0.710	1.44	0.380	268222
Case 3							
ER10	1.03	0.30	2.00	0.624	1.52	0.387	276610
ER20	1.10	0.29	1.92	0.571	1.35	0.296	271850
ER40	1.19	0.60	2.00	0.527	1.35	0.333	268876
ER60	1.42	1.14	2.12	0.520	1.39	0.287	268201
ER80	1.57	1.32	2.14	0.809	1.42	0.331	268379
ER100	1.65	1.75	2.32	0.802	1.64	0.327	250000

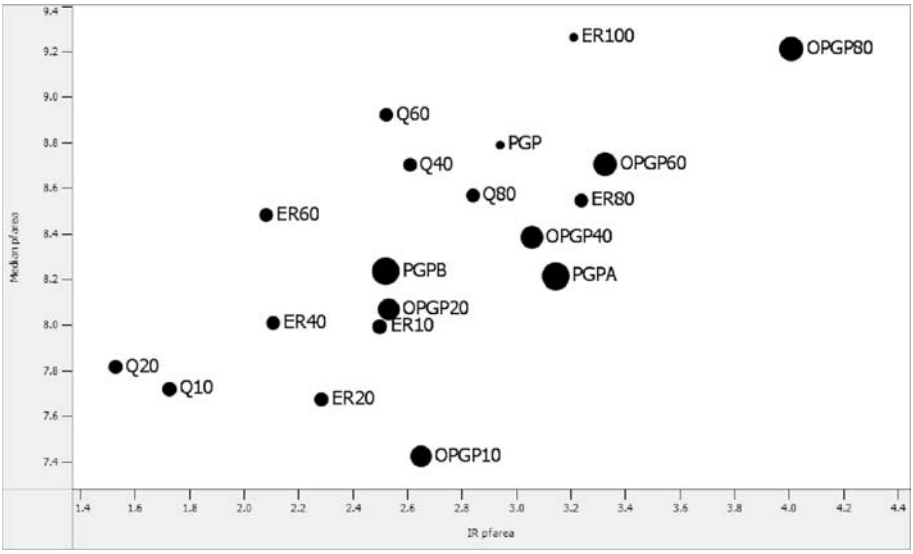
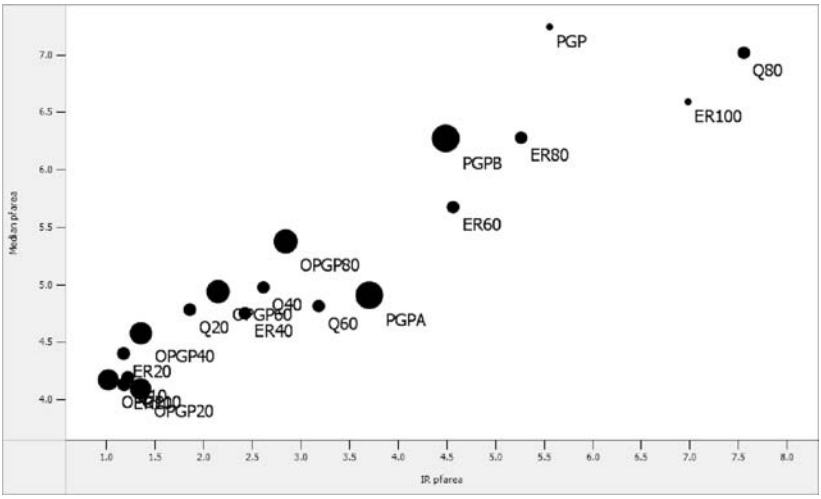


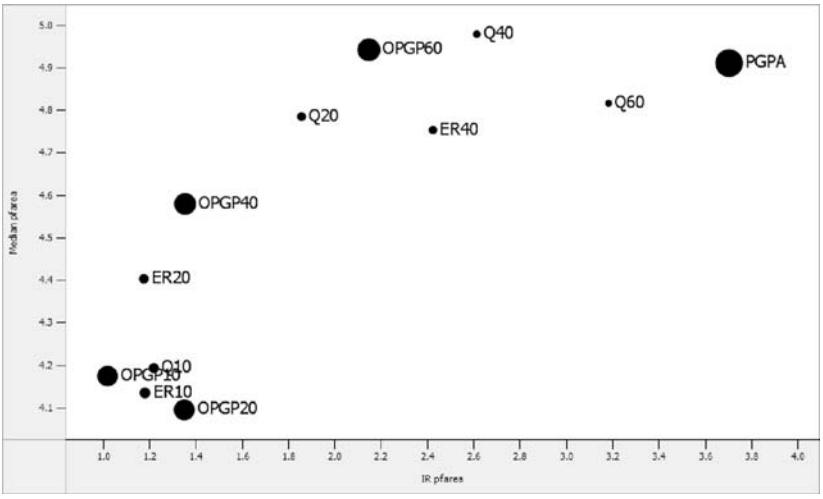
Figure 2-5. **Summary of Results for the Kotanchek problem.** The bubbles represent the results of the different experiments in Cases 1–3 in terms of the median performance measure over 50 independent runs (y-axis), interquartile range of the performance measure (x-axis), and the normalized total number of function evaluations per run (the size of the bubbles). We use the same labels as in Table 2-3. PGP – reference ParetoGP runs, OPGP – ordinal ParetoGP runs of Case 1, Q – quick ordinal runs of Case 2 with less frequent archive re-evaluations, ER – exploratory runs of Case 3 with re-allocated cpu-budget. The numbers in the labels indicate the starting level of the subset size. The area of interest is the lower-left corner of the graph. Note that the most successful experiments are Q20, Q10, ER20, OPGP10, which confirms that starting from small subsets leads to solutions superior in terms of best median fitness and consistency of independent runs.

data set. In this chapter the selection of subsets is uniformly random within the collection of the available data records. During the run we gradually increase the subset size while decreasing the population size. By doing so we pursue better exploration at the beginning of the evolution by evaluating more individuals at a time, albeit more coarsely, and gradually improve exploitation by refining the evaluations by adding more data points. Five different experiments were performed where the subset size at generation one was selected from a set of $\{10\%, 20\%, 40\%, 60\%, 80\%\}$ of the original data.

This time we want to keep the number of function evaluations per generation constant. Therefore, when the scheme of increasing the data subset size is chosen, and the number of records used for each generation is defined, the population size for a given generation is obtained by dividing the number of function evaluations (the budget) per generation by the number of records. All experiments were repeated more than 50 times to get reliable statistics.



(a) All experiments



(b) Magnified area of interest

Figure 2-6. **Summary of Results for the Maarten problem.** The bubbles represent the results of the different experiments in Cases 1–3 in terms of the median performance measure over 50 independent runs (y-axis), interquartile range of the performance measure (x-axis), and the normalized total number of function evaluations per run (the size of the bubbles). We use the same labels as in Table 2-3. PGP – reference ParetoGP runs, OPGP - ordinal ParetoGP runs of Case 1, Q – quick ordinal runs of Case 2 with less frequent archive re-evaluations, ER – exploratory runs of Case 3 with re-allocated cpu-budget. The numbers in the labels indicate the starting level of the subset size. The area of interest is the lower-left corner of the graph. The most successful experiments for the Maarten problem are OPGP10, Q10, ER10, OPGP20, which again confirms that starting from small subsets leads to solutions superior in terms of best median fitness and consistency of independent runs. Note, the substantial budget savings in Q10 and ER10, which correspond to Case 2, and Case 3 respectively.

CASE 2. In a second set of experiments we are repeating the set up described in Case 1 with one small modification. We are no longer re-evaluating individuals in the archive at every generation, but only at every 10th generation. The cpu-budget assigned to these archive re-evaluations is cut by 90% compared with Case 1. This introduces further softening of the selection process, since archive individuals that accidentally obtain (unrealistically) high fitness values, can still survive in the archive for up to ten generations, and, hence, compete and propagate their features to their offspring. As noted in the previous section, the error in ranking caused by this approach are relatively small, and should still produce solutions comparable with the solutions of Case 1.

CASE 3. In the previous two cases the cpu-budget per generation was kept constant. In the third set of experiments we examine the effect of re-distributing the total cpu-budget in such a way, that we spend 50% more function evaluations at the first generation, gradually decrease the budget over the run, and end up with spending 50% less at the last generation, compared to the corresponding runs of Case 1 and 2. The total budget per run, however, does not change. The intention is to have even more exploration at the beginning of the run at the expense of exploitation at the end of the run. The scheme for archive re-evaluations is the same as in Case 2 – once every ten generations.

Simulation Results

The simulation results for the three cases are summarized in Table 2-3. We used the following criteria for estimating the performance of different experiments: the median and the interquartile range of the percentage of the area under the Pareto front for 50 independent replicates, and the total number of function evaluations per run normalized by the size of the training data set. For each criterion lower values indicate better performance.

The results from Table 2-3 are also displayed as bubble charts in Figures 2-5, 2-6, and 2-7. The size of the bubbles in these charts is proportional to the budget that was spent for a particular experiment.

When we examine the results for **Case 1**, we observe a clear improvement of the median fitness as well as the IQR as we start with smaller and smaller subsets. The surprising fact is that even for very small datasets, like Maarten and Kotanchek, the optimal strategy is to start with a low subset size of 10 to 20%. This is most probably related to the low dimensionality of these problems. Also note, that the runs starting with the smaller subset sizes also consume less cpu-budget because the archive reevaluations are cheaper. The ordinal runs starting with small subset sizes (OPGP10 and OPGP20) considerably outperform the reference ParetoGP runs with a constant population size of 100 (PGP), and even those with twice the number of generations (PGPA) or twice the population size (PGPB).

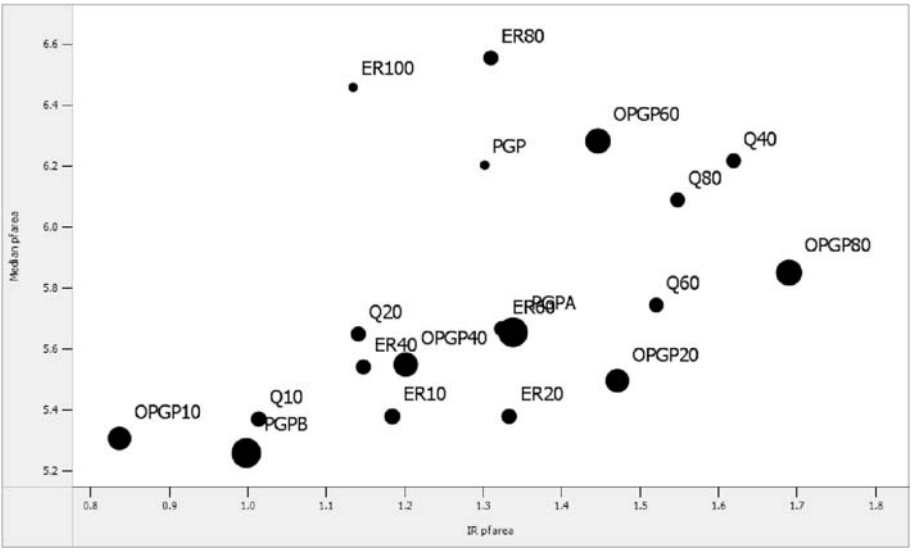


Figure 2-7. **Summary of Results for the Tower problem.** The most successful experiments for the Tower problem are OPGP10, Q10, and PGPB. The first two are no surprise, we observe the same results for other test problems. The PGPB corresponds to the reference ParetoGP runs that uses twice the population size for the same number of generations. The high quality of these results is caused by the nature of the tower problem. This real industrial problem is more challenging with respect to the variable selection (23 candidate inputs). Besides, the exact underlying input – output relationship may not exist. It is no surprise that PGPB runs with a bigger population size , and, hence, better exploration, produce better solutions than exploitative PGPA runs with more generations. It is important to note that this improvement requires a higher cpu-budget.

In **Case 2**, we examine the effect of not reevaluating the archive individuals at every generation but only once every ten generations. The bubble charts show only a slight deterioration in the results for the median fitness, and the IQR, compared to the ordinal runs from Case 1. Considering the budget, these experiments are nevertheless very competitive. The advantage is that the cpu-budget now is actually very close to the original budget of the PGP reference run.

In **Case 3** the aim was to examine whether there was any benefit in relocating a part of the cpu-budget from the end to the beginning of a run. The results are comparable with the results of Case 2, however there is a clear deterioration in reproducibility of the independent replicates. We can speculate, that by relocating 25% of the cpu-budget from the second half of the run into the beginning, we actually assign too much weight to exploration and not enough to exploitation. While a thorough analysis of other budget distribution schemes is

required, the provisional conclusion is that a constant cpu-budget per generation is quite a good strategy to be used as a default.

6. Conclusions

We confirmed and refined our earlier findings that the use of goal softening consistently generates results that are superior both in terms of median fitness, consistency between independent runs, and required computational budget. We conclude that starting with subset sizes of only 10 to 20% generates optimal results for all three test problems.

We also confirmed that there is no need to reevaluate the archive individuals at every generation, keeping essentially the same quality of results at the lower cpu-budget. Finally we showed that keeping the number of function evaluations per generation constant over the run, is a good default setting and seems to provide the necessary balance between exploration and exploitation in soft ordinal ParetoGP.

Acknowledgment

The authors would like to thank Arthur Kordon for providing them with the data set corresponding to the tower problem. The authors would also like to thank Dick Den Hertog from Tilburg University for many discussions on ordinal optimization and evolutionary computation.

References

Gathercole, Chris and Ross, Peter (1994). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Yuval, Schwefel, Hans-Paul, and Männer, Reinhard, editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Jerusalem. Springer-Verlag.

Ho, Y.-C., Cassandras, C.G., Chen, C.-H, and Dai, L (2000). Ordinal optimization and simulation. *Journal of the Operational Research Society*, 51:490–500.

Ho, Yu-Chi (2000). Soft optimization for hard problems, computerized lecture via private communication/distribution.

Keijzer, Maarten and Foster, James (2007). Crossover bias in genetic programming. In Ebner, Marc, O'Neill, Michael, Ekárt, Anikó, Vanneschi, Leonardo, and Esparcia-Alcázar, Anna Isabel, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, Valencia, Spain. Springer.

Kotanchek, Mark, Smits, Guido, and Vladislavleva, Ekaterina (2006). Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic*

Programming Theory and Practice IV, volume 5 of *Genetic and Evolutionary Computation*, chapter 3, pages –. Springer, Ann Arbor.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Lau, T.W. Edward and Ho, Yu-Chi (1997). Universal alignment probabilities and subset selection for ordinal optimization. *J. Optim. Theory Appl.*, 93(3):455–489.

Laumanns, Marco, Thiele, Lothar, Zitzler, Eckart, and Deb, Kalyanmoy (2002). Archiving with guaranteed convergence and diversity in multi-objective optimization. In *GECCO*, pages 439–447.

Smits, Guido, Kordon, Arthur, Vladislavleva, Katherine, Jordaan, Elsa, and Kotanchek, Mark (2005). Variable selection in industrial datasets using pareto genetic programming. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 6, pages 79–92. Springer, Ann Arbor.

Smits, Guido and Kotanchek, Mark (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, Ann Arbor.

Smits, Guido and Vladislavleva, Ekaterina (2006). Ordinal pareto genetic programming. In Yen, Gary G., Wang, Lipo, Bonissone, Piero, and Lucas, Simon M., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 3114 – 3120, Vancouver. IEEE Press.

Teller, Astro and Andre, David (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA. Morgan Kaufmann.

Zhang, Byoung-Tak and Cho, Dong-Yeon (1998). Genetic programming with active data selection. In McKay, R. I. Bob, Yao, X., Newton, Charles S., Kim, J.-H., and Furuhashi, T., editors, *Simulated Evolution and Learning: Second Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98. Selected Papers*, volume 1585 of *LNAI*, pages 146–153, Australian Defence Force Academy, Canberra, Australia. Springer-Verlag. published in 1999.

Zitzler, Eckart and Thiele, Lothar (1998). Multiobjective optimization using evolutionary algorithms - a comparative case study. In Eiben, A. E., Bäck, Thomas, Schoenauer, Marc, and Schwefel, Hans-Paul, editors, *PPSN*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–304. Springer.

Chapter 3

MANIPULATION OF CONVERGENCE IN EVOLUTIONARY SYSTEMS

Gearoid Murphy[1] and Conor Ryan[1]

[1] *Biocomputing and Developmental Systems Group, University of Limerick, Ireland*

Abstract Convergence is a necessary part of any successful GP run, but is also a great weakness due to material lost through the hemorrhage of genetic content through the established evolutionary dynamics. This chapter examines the phenomenon of convergence commonly observed in evolutionary systems before introducing a number of functionally distinct mechanisms are analysed and tested with respect to their ability to modulate the loss of potentially useful genetic content. Each of these methods use little or no explicit measurements to calculate diversity, and we show that they can have a dramatic effect on empirical performance of GP.

Keywords: convergence manipulation

1. Introduction

Convergence in any domain intuitively invokes analogies of stabilisation, specialisation and exploitation. In the domain of genetic programming, convergence refers to the characteristic behaviour of the population dynamics; wherein the initial random set of expressions undergo a series of alterations which results in an increase in the relevance of the expressions to the requirements of the target domain. The concentration of the content of the expressions within a relatively constrained area of the expression space is another effect of convergence.

In general the term of convergence is used fairly intuitively within the Genetic Programming community and is usually meant to invoke the previous description. For the duration of the chapter we will also subscribe to this generic philosophy, in favour of more specific definitions.

The notion of convergence has particularly broad connotations within the context of AI paradigms, more specifically within that subset of AI techniques

which utilise some stochastic process in their initial projections. This is self-evident in GP wherein the systems initial random projection is convoluted by the evolutionary dynamics so as to conform to the objective feedback signal but is also observed in gradient descent methods such as Neural Networks.

In the context of Neural Networks, the phenomenon of convergence manifests itself as the system's ever increasing conformity with the training signal. The effectiveness of Neural Networks as a search method and for gradient descent methods in general depends heavily on what is referred to as the alpha factor. The parameter modulates the magnitude of the changes in the "position" of each learning weight with respect to the output error. Choosing an alpha factor which is too large means that the "jumps" around the expression space are too large, and potentially useful states are missed, alternatively choosing an alpha factor which is too small means that a great deal of unproductive areas of the expression space are evaluated.

This effect of the alpha factor on the convergence behaviour of neural networks succinctly summarise the challenges associated with exploiting the dynamics of convergence in the need to adequately explore the expression space whilst at the same time exploiting the areas of the expression space already occupied. Unfortunately, no single parameter can account for the convergence behaviour of genetic algorithms in so salient a manner as the alpha parameter does for Neural Networks, and we are instead faced with number of possibly interacting parameters, such as population size and selection methods.

The primary consequence of this observation is the motivation to investigate, understand and if possible exploit the dynamics of convergence within a Genetic Programming context. This will improve the viability of the paradigm as an AI solution. To achieve this we put forward a series of experiments whose purpose is to show the potential of the experimental techniques to improve the genetic content of the population whilst as the same time preventing the loss of potentially useful sub solutions.

The layout of the paper is as follows, Section 2 gives some background on the previous work done on the manipulation of convergence. Section 3 describes the specific parameters that are common to all experiments. The experimental work then follows. A discussion of the work and future aspirations is given in Section 8 followed by our conclusions in Section 9.

2. Background

Ever since the inception of contemporary evolutionary algorithms in Hollands seminal work, (Holland, 1975), the proliferation of a single dominant solution throughout the population and the consequent evolutionary gridlock it results in, has been recognised as a core concern of the science. This sec-

tion reviews some of the work done to address this issue and the nature of the resulting strategies produced.

Using the fitness of the population as a mechanism to prevent premature convergence was one of the first such strategies developed in; (Goldberg and Richardson, 1987). By modulating the finesses of the population to inhibit dense congregations of homogeneous solutions, *fitness sharing* techniques attempted to segregate the population into *niches*. Inspired by observations of niche specialisation within natural biological ecosystems, many variants of this strategy have been pursued such as sequential niching, (D. Beasley and Martin, 1993), speciation with implicit fitness sharing and co-evolution, (Darwen and Yao, 1997) and a niching method known as clearing, (Sareni and Krahenbuhl, 1998).

While effective at encouraging the occupation of multiple local optimum, fitness sharing methods suffer from the need to set a priori parameters such as the similarity measure needed to define a minimum distance between optima. Such measures are hard to define and may need to change in the later stages of evolution, (Sareni and Krahenbuhl, 1998). Using fitness as the primary segregation measure has also been demonstrated in the Hierarchal Fair Competition Model, (Hu and Goodman, 2002).

Using age as the method by which convergence may be inhibited has also been investigated. The Age Layered Population Structure (ALPS), (Hornby, 2005) has been used to great effect on a difficult optimisation problem. The method uses age "bands" to define the population pools the individuals of the population may occupy. New individuals are continually introduced so as to maintain exploration. Normal evolutionary dynamics are present within the age "bands".

An alternative way in which age may be used to modulate population convergence is given in (Naoyuki Kubota and Shimojima, 1994). This technique undertakes the hypothesis that the effectiveness of convergence is inhibited by the loss of potentially useful genetic content due to the replacement strategies of steady state wherein the worst individual is replaced by the new offspring. This dynamic is inhibited by allowing individuals to exist for multiple evolutionary cycles within the population. An age value is used to determine how many cycles they may exist on the population. So, rather than trying to *segregate* the population, this method attempts to *protect* relatively poor individuals, thus allowing them the opportunity to propagate their genetic content.

Both age and fitness have thus far been demonstrated as valid strategies for manipulating the dynamics of convergence. Another strategy discussed here employs spatial segregation and is known as the *Island Model*, (Dehmeshki et al., 2003). In this model, multiple initial populations are allowed to evolve, the idea being that each population will develop towards different optima. After a pre-specified number of evolutionary cycles, the "islands" are allowed to in-

tercommunicate. By such communication, the models are producing a number of beneficial effects.

Primarily, the homogenisation of each island is delayed with the introduction of such new material, with beneficial consequences for the ultimate convergence of the entire population of islands. Also, separate aspects of the final solution may be present in different islands, by communication these sub solutions are allowed to coexist and intermingle. The incompatibility of highly evolved solutions of the same domain but from different evolutionary runs is a well known phenomenon, and inter island communication avoids this by maintaining coherence between the islands.

An alternative mechanism for mechanism for preventing convergence is to inhibit crossover events between individuals whose genetic lineage is similar, with the intention of preventing the proliferation of a dominant "family" of expressions, (Braught, 2005). This mechanism also encourages exploration by forcing solutions whose origin lies in relatively different areas of the expression space into crossover events.

This concludes our background for the work on manipulating convergence. The domain is so vast that an exhaustive description of all the techniques is infeasible but we feel we have described the work with most relevance to the experiments described in this chapter.

3. Overview of experiments

The parity domain is presented as the domain of choice for the experiments performed in this chapter. This problem is difficult without the inclusion of the XOR or EQ functions, thus providing a reasonably difficult test of the viability of the experiments whilst at the same time being reasonably manageable in terms of time to compute. All parity experiments used 5 inputs with a maximum attainable fitness is 32. Each experimental result is the averaged behaviour of 30 independent runs.

The experimental parameters used to define the runs are given in Table 3-1. These remain constant for all experiments except were explicitly stated otherwise. The function set used by the the experiments is given in Table 3-2. All populations were initialised using the ramped half and half technique taken from (Koza, 1994). All models use the steady state evolutionary dynamics unless explicitly stated otherwise. Individual sections of this chapter contain any further details needed to implement the experiments described in them.

The order of the experiments reflect the chronology of their execution. Each experiment begins with a particular hypothesis and tests it in some way. Discrepancies between the projected behaviour and the observed dynamics serve as focal points for the development of further hypotheses.

Table 3-1. Experimental parameters used throughout the chapter. Deviation from the parameters will be explicitly stated during each experimental instance.

Parameter	Value
pop size	100
tournament size	2
max initial depth	6
min initial depth	4
mutation rate	0.0

Table 3-2. Functional primitives used in all experiments.

Function	Arity
And	2
Or	2
Nand	2
Nor	2

In order to provide insight into the dynamics of the behaviour of the models, we employ fitness comparison graphs which plot the average best fitness characteristics of the systems against a standard GP system. Error bars are omitted from the fitness graphs for clarity. We will also exploit the fact that our experiments are situated in a Boolean domain by mapping the binary output sequence associated with each expression over the entire set of test cases onto an integer number. By counting the number of unique integers within the population of candidate solutions we can derive a sense of how well the various strategies are maintaining different independent solutions. These simple representations of solution diversity give an immediate intuition about the convergence behaviour of the systems.

Explicit metrics of phenotypic structural diversity are not shown as they were never used in the actual experimental processes and to include them out of context would detract from the primary motivation of the paper, which is to manipulate convergence, not to promote such diversity. Improving convergence and promoting such diversity are not necessarily mutually exclusive but convergence can be improved with a loss of such diversity metrics.

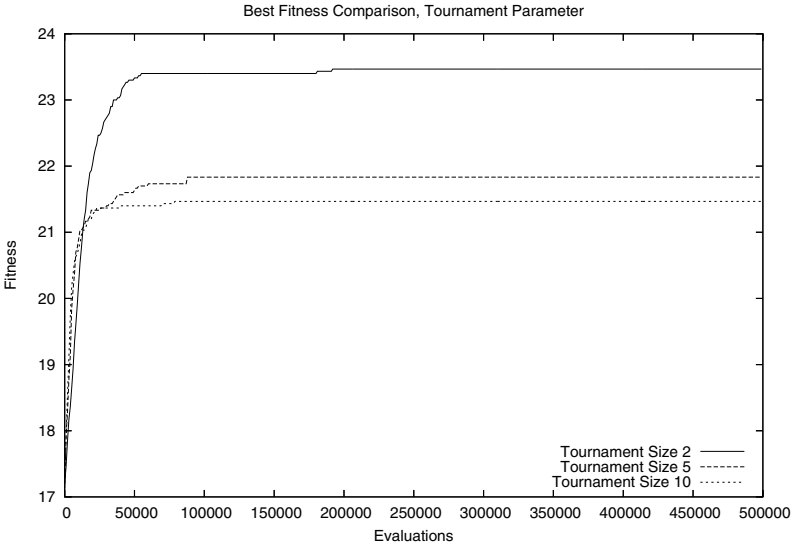


Figure 3-1. Best fitness averaged over 30 independent runs for a variety of tournament sizes. A tournament size of two randomly selects parents with the fitter parent contributing the root branch in the crossover event.

4. Experiment : Modulation of Tournament Size and Population Size

This section presents a number of experiments showing the effect that modulating a number of different parameters and mechanisms can have on the dynamics of convergence. We stress that these results are well established within the community and that conventional wisdom would dictate that there is nothing to be gained from reinventing the wheel, however the conclusions drawn from these results are instrumental in the conception of the next experiment, so for the convenience of contextual congruity and effective communication we include them.

The first set of results in Figure 3-1 illustrates the effect of the size of the tournament on the population dynamics. It is immediately obvious that reducing the size of the tournament used to select the parents of a crossover event has a beneficial effect for the convergence dynamics of the population. In a situation where there is a relatively large tournament size, a few of the fittest individuals will consistently transmit the genetic content of their expressions amongst the population very quickly.

In rare situations, such high convergence rates will settle on a sustainable convergence dynamic wherein the quality of the population as it converges increases substantially. Unfortunately, such behaviour is the exception rather

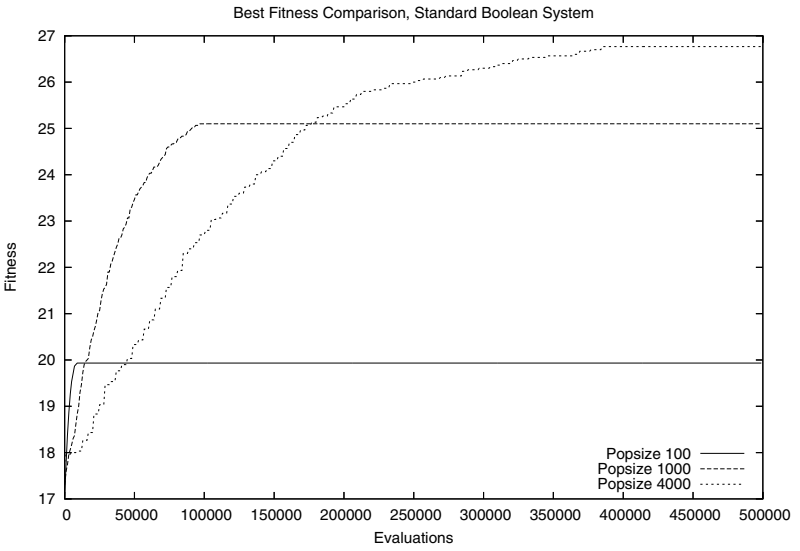


Figure 3-2. Best fitness averaged over 30 independent runs for a variety of population sizes. Increasing the population size delays premature convergence but does not avoid it.

than the rule, as high tournament sizes generally have a detrimental effect on the convergence behaviour of the system.

Our results showed that the best performing tournament strategy simply picked two parents at random, with the better parent contributing the root branch to the crossover event. If the resulting offspring was better than the worst parent, it was reinserted into the worst parents place.

The importance of this result is that it illustrates that the established dynamic of primarily proliferating the genetic content of the fittest individuals serves only to prematurely accelerate the system into an inescapable local optimum, particularly in a difficult domain such as parity.

The next set of results focused on the effect of changing the population size. Figure 3-2 shows the progression of the best fitness for population sizes 100, 1000 and 4000. It is well known that in order to achieve better results in a given domain using Genetic Programming, one can improve results simply by increasing the population size. This significantly increases the length of time it takes for one good individual to start dominating the population, thus allowing time for the exploration of less fit but potentially useful expressions, however even the largest population still converged before coming close to the max fitness of 32. However, simply increasing the population size, whilst undeniably effective, suffers from a harsh law of diminishing returns, and so is only effective up to a certain point. We use *solution density* graphs to demonstrate how the

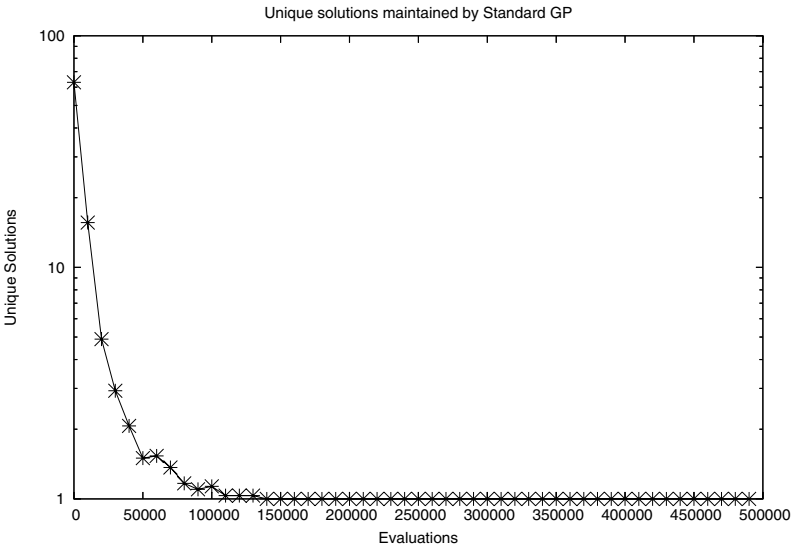


Figure 3-3. Diagram illustrating rapid loss of unique solutions within the population over time. Note that this measure does not perform any explicit diversity analysis of the genetic content of the population, rather, it looks at the environmental identity of the expressions, in the context of the boolean domains.

number of unique solutions changes amongst the population during the run. Consider the population in Figure 3-3.

The most pressing observation from these diagrams is that the pressure on the system to converge, even with the low tournament size, is very high. So high in fact, that the population quickly concentrates itself into dense pools of homogeneous expressions, exhibiting the characteristics of a "greedy" algorithm and consequently devastating its potential for improvement through parts of the expression space which do not immediately provide a return in fitness.

We conclude that the most important factor modulating the concentration of the population into dense pools of homogeneous expressions was not the prevalence of a single solution but the loss of so many potentially useful untried expressions. This motivated the next experimental setup wherein the loss of expressions was manipulated by using age-based dynamics.

5. Experiment : Dynamically Sized Population with Age Based Dynamics

This section describes an experiment implemented to test the hypothesis that by slowing the flow of relatively unfit but potentially useful genetic content out of the population, that the quality of the evolved population will be improved.

Table 3-3. Parameters used to implement the Age based system.

Parameter	Value
Population Size	100
Tournament Size	2
Population Increase	10
Positive Crossover Feedback	2
Negative Crossover Feedback	-1
Initial Individual Age	40

We implement this dynamic by changing one of the fundamental mechanisms in the tournament selection, wherein the worst individual of a tournament is *not* replaced by better offspring from the winning parents of the tournament. Instead each newly created individual is given a number of cycles of existence within the population. Furthermore we augment this dynamic by rewarding parents which successfully produce a valid offspring with more cycles of existence and punishing parents producing unfit offspring by reducing the number of evolutionary cycles they have been allocated.

After each generation, the produced offspring are sorted and a pre designated number of the best offspring are allowed into the population, controlled by the population increase parameter in Table 3-3. This combination of parameters results in a dynamically sized population. After the initial growth spurt, the population will stabilise at some size, depending on the chosen parameters, and stay around that point, as the flow of individuals out of the population matches the flow of new individuals into the population.

From Figure 3-4 we can see that a population using this method to does better than a population utilising a standard steady state algorithm with tournament selection and with an initial population ten times larger. This reinforces the hypothesis that manipulating the convergence dynamics can substantially improve the effectiveness with which a given expression space can be searched. Furthermore, it also shows us that even relatively small populations, will to a large extent, contain quite a lot of the expressions needed to solve the problem.

However, the approach is inconsistent, sometimes doing very well and sometimes not, Figure 3-5. Observing the fitness density graph in Figure 3-6 indicates why. In contrast to previously shown solution density graph taken from standard GP implementations, the age based system is clearly maintaining discrete pools of expressions. These pools can only be maintained by their being continually replenished with positive feedback from successful crossover events. This accounts for the systems resistance to convergence as evidenced by the improved

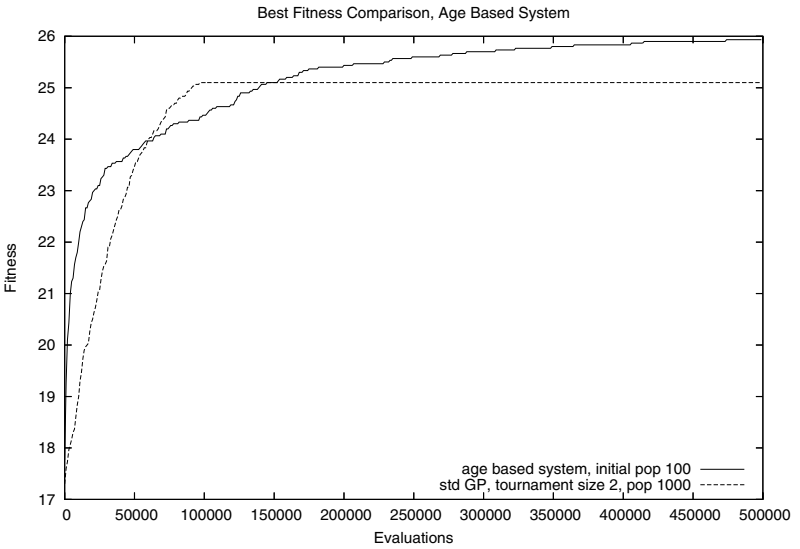


Figure 3-4. Best fitness averaged over 30 independent runs, comparison between age based model with standard GP model. Even though the standard system starts off with a population 10 times larger than the age model, it is still outperformed by the age model.

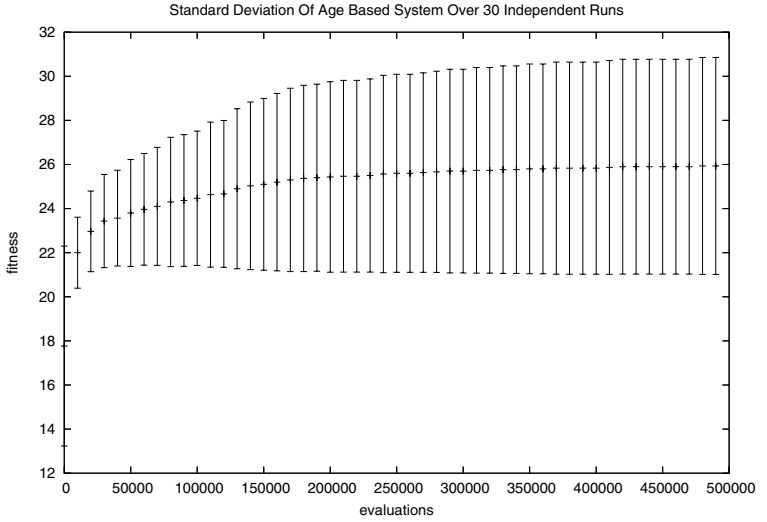


Figure 3-5. Figure showing high levels of statistical variance in performance of the age based dynamics model.

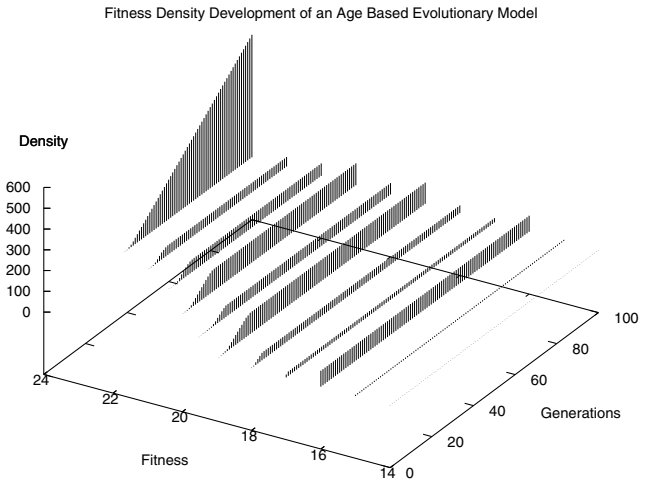


Figure 3-6. Fitness density diagram for the age model. Note the distinctly identifiable evolutionary "niches" occupied. These are maintained by the feedback from positive crossover events. Also, observe the increase in size of the highest fitness gradient, consistent with the dynamic population size of the model

results however, it is plainly evident that the vast majority of the population is still concentrating itself into a high density pool of expressions.

So, while the age-based dynamics have improved the performance to some extent, the system is still vulnerable to situations where the pressure to converge onto an sub optimum is too powerful to resist and consequently neutralises the benefit of the age based dynamics.

Given the observation that convergence is such an unavoidable evolutionary phenomenon, how can we improve its chances of avoiding sub optimal attractors?

6. Experiment : Interacting Subpopulations

Genetic Programming implementations such as RTL, (Keijzer et al., 2005), have demonstrated the viability of improving the convergence behaviour of evolutionary systems by continually improving the functional content of the populations initial generation. However these are most effective in domains where the functional structure is relatively salient and are not as generally applicable to a broad range of problems like the methods presented here.

This experiment attempts to mimic aspects of RTL's behaviour in that it attempts to continually improve the initial genetic content for a sequence of evolutionary runs as well as implementing the necessary functionality to pre-

vent the dense homogeneous expression pools characterising highly converged populations as observed in Section 4.

Our design solution to these requirements was an evolutionary model consisting of a population pool which is generated from the expressions derived from *multiple evolutionary* runs of subpopulations of an earlier population pool. The subpopulations are obviously much smaller than the population pool – in fact in our runs we have found that the best size for a sub population is only two individuals. Given that we can expect convergence to produce a highly homogeneous subpopulation at the end of the run, having a small subpopulation size prevents the propagation of a particularly dominant individual.

The subpopulations are seeded with random individuals from the current population pool, ignoring any preference applicable due to fitness, so as to improve the convergence dynamics in the manner observed in the experiments in Section 4.

The subpopulations are allowed to run for a specified number of generations. At the end of a subpopulation run, a taboo filter is passed over the converged subpopulation so as to prevent duplicate expressions entering into the next generation population pool. The taboo filter is a simple test for equality between two expressions. As the individuals are picked at random and as a considerable number of subpopulation runs may be needed to generate the next population, it is highly likely that a a single individual may be part of multiple subpopulations.

Figure 3-7 shows the best fitness performance of the system for a number of different subpopulation generation lengths. This parameter was considered the most influential on the behaviour of the system as it directly controlled the investment of effort made by the system in improving the genetic content of the subpopulations. This can be seen quite clearly in the graph as the higher subpopulation generations exhibit the best performance.

The relationship between the interacting subpopulation model and previously established evolutionary models should be acknowledged. In particular the similarities between Island Models and the interacting subpopulation model are strong. The subpopulation model was originally envisaged as an iterative mechanism to improve the content of a population whilst slowing convergence. Island models provide the same form of functionality, albeit in a different method of execution, what it illustrates though is the validity of mixing the results of multiple sub evolutions so as to provide a satisfactory strategy of both exploitation and exploration. Also, when the subpopulations are only allowed to evolve for 1 generation, the system is very similar to a generational model.

As an instantiation of the hypothesis that improving the content of a population whilst preventing the homogenisation of the expressions in the population, the interacting subpopulation model is reasonably successful. One of the core principles of the system, selecting the subpopulations at random, was imple-

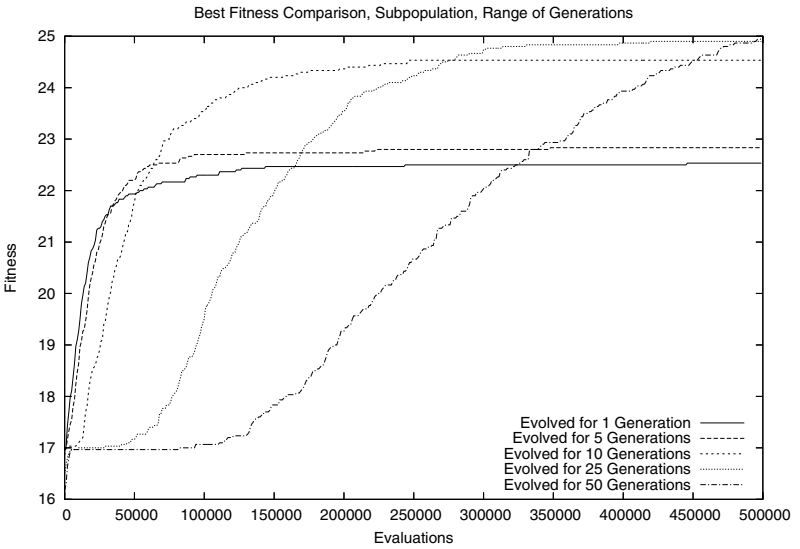


Figure 3-7. Best fitness averaged over 30 independent runs. Each experiment modulated the amount of time each subpopulation spent evolving. Its effect is clearly beneficial to the system, however it can inhibit the performance increase of the population when too much time is spent attempting to improve the fitness of the subpopulations, as evidenced by the experiment which used 50 generations to improve the subpopulations. The latency observed in the fitness of the higher generation experiments is symptomatic of the high number of evaluations applied to the very poor initial content of the populations.

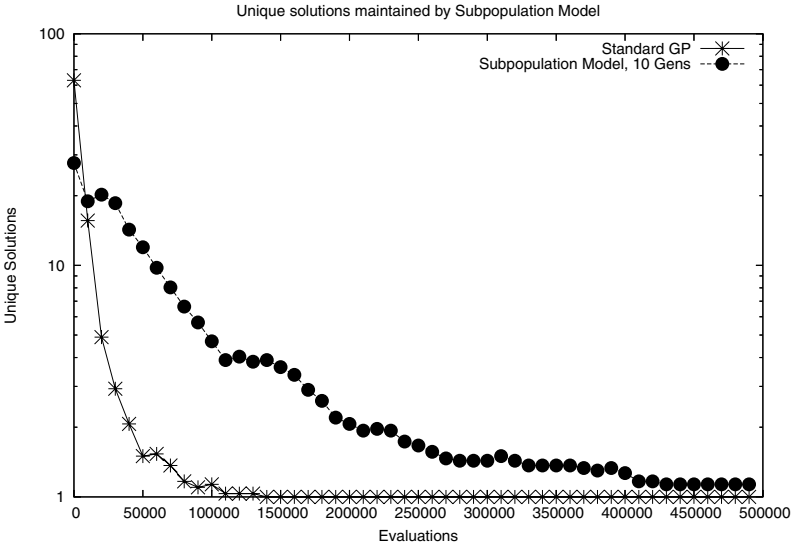


Figure 3-8. Solution density development for an interacting sub population model. Note the systems relative resistance to being trapped by a single sub-optimum, however, it is still susceptible to such inescapable attractors

mented on the observation from Section 4 that by relieving the selection pressure one could prevent the rapid takeover of a population by a particularly dominant expression.

However, this practice does not completely eliminate the phenomenon of dominance by a sub-optimal individual, as the evolutionary dynamics do not facilitate the mixing of expressions from relatively different areas of the search space. The next experiment attempts to do just that.

7. Experiment : Hereditary Repulsion

This section details the final experiment presented for this chapter. It was designed to offset the homogenisation of a population of expressions by forcing different areas of the expression space to mix. By continually facilitating a vigorous convolution of the genetic content as well as only accepting individuals with a marked improvement over their parents, this evolutionary model attempts to capture the best aspects of both exploration and exploitation.

The most distinctive aspect of its functionality is the manner in which it encourages different location areas in the expression space to be involved in the crossover events. It does this through hereditary repulsion. The idea behind the mechanism is that each expression has a specific family tree associated with it. By taking two individuals and counting the number of common ancestors

they have, a rough measure of their common lineage can be derived. The notion of lineage as measure to increase diversity by seeding tournaments with diverse lineages has been examined in (Braught, 2005). This setup can still allow relatively high fitness individuals to start dominating the population. Our modification to the lineage paradigm is to base the tournaments on minimising the hereditary overlap.

To this effect, our selection method picks an expression at random. It then picks a tournament set of expressions and determines the hereditary overlap between the initial random expressions and the tournament set of expressions. The expression with the least overlap is judged as being the most viable one to combine. In this way, exploration is encouraged.

The next critical aspect of the model's behaviour is that the resulting expression from the crossover must be better than both its parents. This mechanism is in place to ensure that the population content will always improve. A taboo operator is employed to prevent the proliferation of a single good expression. The model operates on a generational population mechanism.

It is evident to the informed reader that such a mechanism will result in a significant number of discarded expressions. However, as shown in Figure 3-9, the model produces excellent results despite this, outperforming all previous implementations. These results are a strong confirmation of the hypothesis that; by providing a powerful combination of both exploration and exploitation; the convergence dynamics of the population of expressions will be significantly improved. The ability of Hereditary Repulsion to avoid convergence is clearly seen in the solution density Figure 3-10.

8. Discussion and Future Work

This section presents a discussion on the papers contents and future work.

Utilisation of the Models

All the models presented in this paper are experimental systems. They were not meant to provide an exhaustive analysis of the various operating characteristics of the algorithms. Because of this, there most likely exist a significant number of augmentations to these systems which could further improve their performance. Work done in (Hornby, 2005), for example, illustrates the effectiveness of a well executed age based dynamics evolutionary system.

The effectiveness of the hereditary repulsion model is an encouraging result, considering the fact that there are no optimisation efforts made on the system. Such efforts would likely focus on preventing repetition of previously rejected expressions and would ultimately reduce the number of evaluations required.

The very fact that the techniques presented here have already been touched upon, in one form or another, as well as their conspicuous absence from the bulk

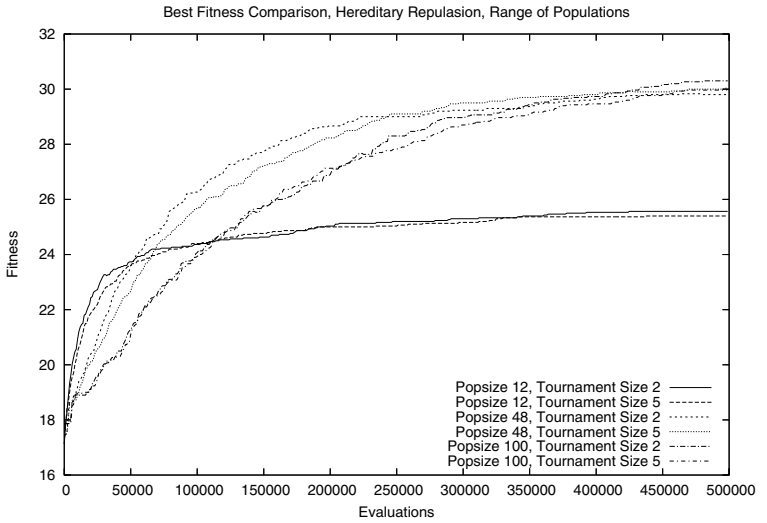


Figure 3-9. Best fitness averaged over 30 independent runs. Diagram clearly shows the significant improvement the hereditary repulsion model confers on the performance of the population. Note the high performance of the experiment using only a population size of 12. This clearly illustrates the capacity of HR to make the best use of the genetic content it has been given.

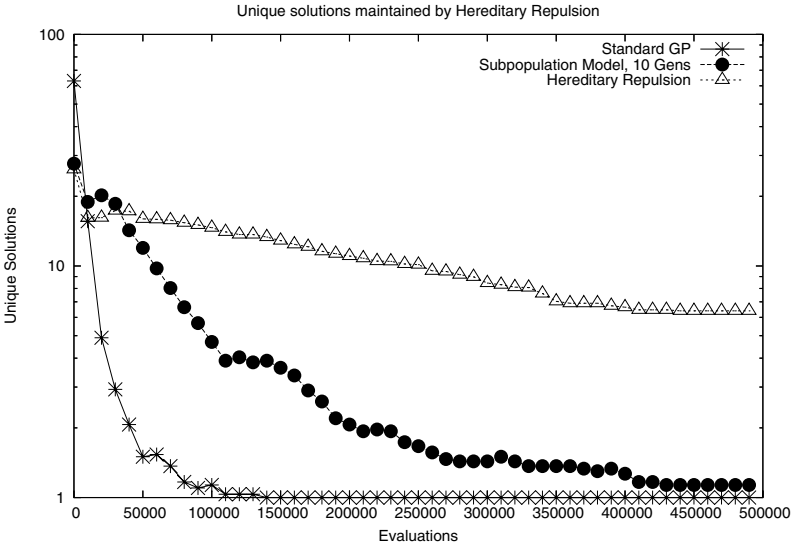


Figure 3-10. Solution densities for Hereditary Repulsion. The strength of the algorithm is seen in the way it successfully maintains a rich set of independent solutions over time, in contrast to the other methods

of the literature reflects the tendency of practitioners to answer the requirements of the problem of premature convergence with basic strategies described in Section 4.

The reason this is acceptable is that, by and large, such a strategy solves the problem at hand. The need for powerful algorithms such as these has become less pressing given the prevalence of powerful and cheap computing power. However, as the problems solved by the paradigm become more and more difficult, a well executed algorithm can preempt thousands of hours of computation.

The power behind evolutionary methods is well established within the Genetic Programming community and the onus is on us to proliferate the deployment of this technology throughout both industry and other scientific disciplines as a valid and powerful scientific methodology.

Analysis Techniques

Deducing the performance of the various techniques though the fitness graphs, while effective, is a very shallow evaluation of the algorithm. Future work will focus on developing efficient algorithmic techniques to map out the search space of the boolean domains. Whilst this represents a considerable technical challenge the benefits of such an analysis mechanism would be considerable.

Previous work done by the authors has found that artificially created problems created for analysing the behaviour of evolutionary models to be highly unstable. Problems were either too hard or too difficult. Because of this, the boolean domain presents a relatively difficult search space whose mapping is well within the capacity of modern computational hardware.

Having such an analysis mechanism would allow the observation of various critical sub solutions as they appeared with the expression space of the population. This would facilitate the further development of experimental hypotheses as the internal functioning of the model would be laid bare to the observer.

Further Development of Models

As previously mentioned, the models here were not fully developed to the extent of their potential. A viable source of future research would be the further development of these models.

The age based dynamics model has been very successful in (Hornby, 2005) and our experience with the model suggests that it certainly has a lot of potential for improvement.

The hereditary repulsion method has also been successfully tested on other unrelated domains, such as the BUPA liver disorders classification problem, as seen in Figure 3-11, however, an exhaustive decomposition of its core parameters has not yet been investigated. Of particular interest is the observation in

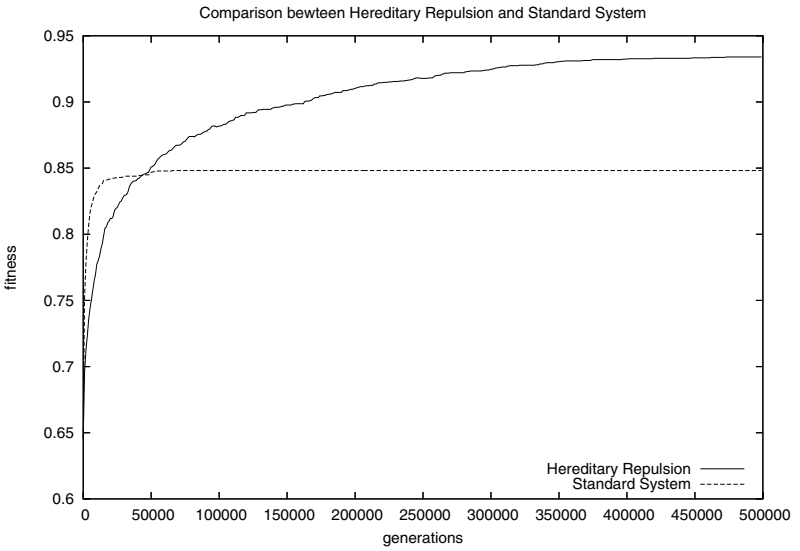


Figure 3-11. Best fitness averaged over 30 independent runs on the BUPA Liver Disorder Classification Problem. Figure compares performance of an Hereditary Repulsion model and standard model with the parameters described in Section 2. HR has performed well on a wide variety of problem domains.

individual runs of particularly high fitness individuals appearing in one cycle, only to disappear in the next.

Since the sampling rate is very high for each evolutionary cycle in Hereditary Repulsion, it is unlikely that the fit individuals have not been involved in crossover events. However, the constraint that the result of the crossover events must be better than *both* the parents seems the likely explanation. The disappearance of these high fitness individuals suggests that the system was not able to improve on them, and as such, represented an evolutionary deaden. An analysis technique such as the one previously described would verify the validity of this hypothesis.

9. Conclusions

This chapter has presented a number of techniques which have improved the performance of Genetic Programming based evolutionary models over standard evolutionary models. The primary purpose of this chapter was to detail the results of a series of experiments which employed variations of existing techniques within the science to improve the convergence dynamics of artificial evolutionary systems.

We have successful shown that by focusing on the requirement of balancing exploration with exploitation; the effectiveness of the algorithm can be signifi-

cantly improved. We hope that this has been of benefit to the community and will enhance the way they implement these algorithms.

References

Braught, G.W. (2005). Learning and lineage selection in genetic algorithms. In *IEEE Transactions on Evolutionary Computation.* IEEE Computational Intelligence Society.

D. Beasley, D. R. Bull and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. In *Evoluionary Computation*, volume 1, pages 101–125.

Darwen, P.J. and Yao, Xin (1997). Speciation as automatic categorical modularization. In *IEEE Transactions on Evolutionary Computation*, volume 1. IEEE Computational Intelligence Society.

Dehmeshki, J., Siddique, M., Lin, Xin-Yu, Roddie, M., and Costello, J. (2003). Automated detection of nodules in the ct lung images using multi-modal genetic algorithm. In *IEEE Transactions on Evolutionary Computation*, volume 1. IEEE Computational Intelligence Society.

Goldberg, D. E. and Richardson, J (1987). Genetic algorithms with sharing for multimodal function optimisation. In *Proc. 2nd Int Conf. Genetic Algorithms*, pages 41–49.

Holland, John .H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.

Hornby, Gregory S. (2005). Alps: the age-layered population structure for reducing the problem of premature convergence. In *Gecco 2005*.

Hu, Jianjun and Goodman, Erik D. (2002). The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In Fogel, David B., El-Sharkawi, Mohamed A., Yao, Xin, Greenwood, Garry, Iba, Hitoshi, Marrow, Paul, and Shackleton, Mark, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 49–54. IEEE Press.

Keijzer, Maarten, Ryan, Conor, Murphy, Gearoid, and Cattolico, Mike (2005). Undirected training of run transferable libraries. In Keijzer, Maarten, Tettamanzi, Andrea, Collet, Pierre, van Hemert, Jano I., and Tomassini, Marco, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 361–370, Lausanne, Switzerland. Springer.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Naoyuki Kubota, Toshio Fukuda, Fumihito Arai and Shimojima, Koji (1994). Genetic algorithm with age structure and its application to self organising manufacturing system. In *IEEE Symposium on Emerging Technologies and Factory Automation*.

Sareni, Bruno and Krahenbuhl, Laurent (1998). Fitness sharing and niching methods revisited. In *IEEE Transactions on Evolutionary Computation*, volume 2. IEEE Computational Intelligence Society.

Chapter 4

LARGE-SCALE, TIME-CONSTRAINED SYMBOLIC REGRESSION-CLASSIFICATION

Michael F. Korns[1]
[1]*Investment Science Corporation, 1 Plum Hollow, Henderson, Nevada 89052 USA*

Abstract This chapter demonstrates a novel method combining particle swarm, differential evolution, and genetic programming to build a symbolic regression tool for large-scale, time-constrained regression-classification problems. In a previous paper we experimented with large scale symbolic regression. Here we describe in detail the enhancements and techniques employed to support large-scale, time–constrained regression and classification. In order to achieve the level of performance reported here, of necessity, we borrowed a number of ideas from disparate schools of genetic programming and recombined them in ways not normally seen in the published literature. We discuss in some detail the construction of the fitness function, the use of abstract grammars to combine genetic programming with differential evolution and particle swarm agents, and the use of context-aware crossover.

Keywords: artificial intelligence, genetic programming, particle swarm, differential evolution, portfolio selection, data mining, formal grammars, quantitative portfolio management

1. Introduction

This is the story of the problems encountered by Investment Science Corporation in using genetic programming techniques to construct a large-scale, time-constrained symbolic regression tool which could also perform classification.

Without delving in-depth into our financial methods, which is strictly forbidden by our corporate policy, we introduce our financial motivations very briefly and from these motivations quickly construct the requirements of a generic symbolic regression tool which could be used for classifying common stocks into long and short candidates for financial applications.

In our previous paper (Korns, 2006), our pursuit of industrial scale performance with large-scale, time-constrained symbolic regression problems required us to reexamine many commonly held beliefs and, of necessity, to borrow a number of ideas from disparate schools of genetic programming and recombine them in ways not normally seen in the published literature. We continue this tradition in this current paper. Of special interest is combining fitness functions to support both symbolic regression and classification of common stocks into long and short candidates, and combining ideas from particle swarm and differential evolution to provide more fine grained control during the Genetic Programming process.

These disparate schools of genetic programming and evolutionary programming were combined into the following unusual hybrids:

- *Combined:*"Hybrid combination of particle swarm agents and GP"
- *Combined:*"Hybrid combination of abstract grammar and tree-based GP"
- *Combined:*"Hybrid combination of multiple island populations and boosting with GP"
- *Combined:*"Hybrid fitness measure supporting symbolic regression and classification of long and short candidates"

This narrative describes an applied research project at Investment Science Corporation involving many years of engineering effort and hundreds of experiments. Each of the hybrid solutions required to overcome the challenges are described in detail. This research project produced a generic symbolic regression-classification tool capable of processing one million row by twenty column data mining tables in less than fifty hours on a single workstation computer (specifically an Intel©Core 2 Duo Processor T7200 (2.00GHz/667MHz/4MB), running our Analytic Information Server software generating LISP agents that maximize the on-board Intel registers and on-chip vector processing capabilities).

Financial Motivation

Our experimental market-neutral trading system selects, from a universe of the 800 most liquid exchange traded common stocks, eighty *(the worst 10%)* securities to sell short, and another eighty *(the best 10%)* to buy long. A successful market-neutral trading system makes a profit greater than that obtained from a market indexing strategy regardless of market conditions.

Consider a quantitative (quant) trading system for the top 800 exchange-traded common stocks with the largest dollar-volume traded in the prior week (Yu et al., 2004; Caplan and Becker, 2004). These securities are so active that we will be able to move millions of dollars in and out of these investments without

appreciably perturbing their prices. We will retrain the system weekly using a sliding training window of five years or 1,250 training days of historical data (Yu et al., 2004). This allows relatively frequent system retraining (weekly) while providing a relatively long retraining period of fifty hours (the weekend).

Our first challenge is selecting a basket of 20 sample column data points from the over 500 available data points such as Open, High, Low, Close, Volume, EPS, Analyst Rating, etc. We solve this issue by implementing multiple independent trading systems. For instance, one might have a value trading system with one set of 20 training points, a growth trading system with another set of training points, and a chartist trading system with yet another set of training points. If one has a farm of 100 workstations, each workstation could retrain each of 100 independent trading systems once per week.

Our second challenge is to perform a symbolic regression every retraining period on a table of one million rows by twenty columns. If we retrain our market-neutral trading system weekly, using the previous five years of market data, a large volume of data must be fed into the system on every retraining period (1,250 historical daily samples for each of 800 common stocks is 1,000,000 rows of training data by 20 columns). In this paper, we construct a generic regression-classification tool which can perform a single 1,000,000 row by 20 column symbolic regression in less than 50 hours on a single workstation computer (so training can complete over the weekend). Clearly such a tool would prove useful not only in our own financial application, but in many other large-scale, time-constrained applications.

Experimental Setup

Our experimental universe consists of a set of nine different fictitious markets driven by different model functions ranging from simple linear to more difficult multi-modal. Statistical best-practice is employed to rigorously separate training and testing data sets so that all experiments are scored on testing data sets very different from the data sets they were trained on. We have crafted nine separate test cases (model formulas), from simple to complex. All of our test cases are trained on one million row by M column randomly generated training matrices (where M is either 1, 5, or 20). Then a separate randomly generated one million row by M column testing matrix is used for scoring. All of our nine test case formulas are shown in Table 4-1 (generated with five columns).

Table 4-1. Test Case Formulas

linear

```
y = 1.57 + (1.57*x0) - (39.34*x1) +
        (2.13*x2) + (46.59*x3) + (11.54*x4);
```

hidden

```
y = 1.57 + (2.13*sin(x2));
```

cubic

```
y = 1.57 + (1.57*x0*x0*x0) -
        (39.34*x1*x1*x1) + (2.13*x2*x2*x2) +
        (46.59*x3*x3*x3) + (11.54*x4*x4*x4);
```

ellipse

```
y = 0.0 + (1.0*x0*x0) + (2.0*x1*x1) + (3.0*x2*x2) +
        (4.0*x3*x3) + (5.0*x4*x4);
```

hyper

```
y = 1.57 + (1.57*tanh(x0*x0*x0)) -
        (39.34*tanh(x1*x1*x1)) + (2.13*tanh(x2*x2*x2)) +
        (46.59*tanh(x3*x3*x3)) + (11.54*tanh(x4*x4*x4));
```

cyclic

```
y = 14.65 + (14.65*x0*sin(x0)) -
        (6.73*x1*cos(x0)) - (18.35*x2*tan(x0)) -
        (40.32*x3*sin(x0)) - (4.43*x4*cos(x0));
```

cross

```
y = -9.16 - (9.16*x0*x0*x0) -
        (19.56*x0*x1*x1) + (21.87*x0*x1*x2) -
        (17.48*x1*x2*x3) + (38.81*x2*x3*x4);
```

mixed

```
if (mod(x0,4) == 0){
    y = (1.57*log(.000001+abs(x0))) -
            (39.34*log(.000001+abs(x1))) +
            (2.13*log(.000001+abs(x2))) +
            (46.59*log(.000001+abs(x3))) +
            (11.54*log(.000001+abs(x4)));
    }
elseif (mod(x0,4) == 1){
    y = (1.57*x0*x0) - (39.34*x1*x1) +
            (2.13*x2*x2) + (46.59*x3*x3) +
            (11.54*x4*x4);
    }
elseif (mod(x0,4) == 2){
    y = (1.57*sin(x0)) - (39.34*sin(x1)) +
        (2.13*sin(x2)) + (46.59*sin(x3)) +
```

```
        (11.54*sin(x4));
    }
elseif (mod(x0,4) == 3){
    y = (1.57*x0) - (39.34*x1) +
        (2.13*x2) + (46.59*x3) +
        (11.54*x4);
    }
```

ratio

```
if (mod(x0,4) == 0){
    y = ((1.57*x0)/(39.34*x1)) +
            ((39.34*x1)/(2.13*x2)) +
            ((2.13*x2)/(46.59*x3)) +
            ((46.59*x3)/(11.54*x4));
    }
elseif (mod(x0,4) == 1){
    y = ((1.57*x0)%(39.34*x1)) +
            ((39.34*x1)%(2.13*x2)) +
            ((2.13*x2)%(46.59*x3)) +
            ((46.59*x3)%(11.54*x4));
    }
elseif (mod(x0,4) == 3){
    y = 0.0 - (39.34* log(.000001+abs(x1))) +
            (2.13* log(.000001+abs(x2))) +
            (46.59*log(.000001+abs(x3))) +
            (11.54* log(.000001+abs(x4)));
    }
```

Our nine test cases vary from simple (linear) to complex (formulas with embedded if-then-else expressions). Finally, to add difficulty, we sometimes train and test our nine test cases with random noise added using the following formula:

```
;; Modify each y in Y or each ty in TY with random noise.
y = (y * .80) + (y * random(.40));
```

The addition of random noise makes each test case inexact and theoretically undiscoverable. Nevertheless, given our application, we need to test our symbolic regression tool against inexact data.

Fitness Measure

- *Combined:* "Hybrid fitness measure supporting symbolic regression and classification of long and short candidates"

Standard regression techniques often utilize least squared error as a fitness measure; however, we would also like to classify securities into long and short candidates. Specifically we would like to measure how successful we are at

predicting the future top 10% best performers (*long candidates*) and the future 10% worst performers (*short candidates*).

Let the dependent variable, Y, be the future profits of a set of securities. If we were prescient, we could automatically select the best future performers *actualBestLongs (ABL)* and worst future performers *actualBestShorts (ABS)* by sorting on Y and selecting an equally weighted set of the top and bottom 10%. Since we are not prescient, we can only select the best future estimated performers *estimatedBestLongs (EBL)* and estimated worst future performers *estimatedBestShorts (EBS)* by sorting on EY and selecting an equally weighted set of the top and bottom 10%. Clearly the following will always be the case.

- -1 <= ((EBL - EBS) / (ABL - ABS)) <= 1

A situation where ((EBL - EBS) / (ABL - ABS)) > 0 indicates we are making money speculating on our short and long candidates. Obviously 1 is a perfect score *(we might as well have been prescient)* and -1 is a very imperfect score. Clearly, considering our financial application, we are interested in regression fitness measures which also classify as well as possible. In fact, even if the regression percent error is poor but the classification is good, we can still have an advantage, in the financial markets, with our symbolic regression tool.

We combined a normalized average percent error score with a classification score to produce an optimal fitness measure for our financial application as follows:

- avgDifY = average of abs(Y[n]-avgY) for all n in N
- avgErrY = average of abs(EY[n]-Y[n]) for all n in N

Our regression error and our classification scores as constructed as follows:

- errPct = avgErrY / avgDifY
- classify = (((EBL - EBS) / (ABL - ABS)) + 1) / 2

Finally, our fitness score is constructed as follows:

- fitness = (errPct + (.001 * classify))

Abstract Grammars

- *Combined:* "Hybrid combination of abstract grammar and tree-based GP"

Recently, informal and formal grammars have been used in genetic programming (O'Neill and Ryan, 2003) to enhance the representation and the efficiency

of a number of applications including symbolic regression. In (Korns, 2006), we discovered that alternative genome representations and evolutionary operators provided less added value than the use of multiple grammars themselves.

Therefore we settled on a hybrid combination of tree-based GP and formal grammars where the head of each sublist is a grammar rule agent with polymorphic methods for mutation, crossover, etc. Different grammar rules communicate with each other by message passing (a staple of object-oriented and agent-oriented software engineering). We use standard mutation and crossover operations (Koza, 1992) and support two regression grammar rules, one for simple regression and one for multiple regression as follows.

- *REG Grammar:* regress(EXP);
- *MVL Grammar:* mvlregress(EXP,EXP,...,EXP);

Our numeric s-expressions, the EXP grammar, are standard JavaScript-like numeric expressions with the variables *x0* through *xm* (where m is the number of columns in the regression problem), real constants such as *2.45 or -34.687*, and with the following binary and unary operators + - / % * < <= == ! = >= > expt max min abs cos cosh cube exp log sin sinh sqroot square tan tanh. To these we add the ternary conditional expression operator " *(...), ?, ..., :, ... ;*".

In this chapter we add an additional abstract numeric expression grammar, the AXP grammar, which is identical to the EXP grammar except that AXP expressions contain abstract real constants *c0* through *ck* (where k is the number of unique abstract real constants in the expression), and abstract variables *v0* through *vj* (where *j* is the number of unique abstract variables in the expression).

For instance, the following concrete expression *regress(3.4 * sin(x3/x5))*, when evaluated, has a fitness score based upon regressing 3.4 times the sine of column three divided by column five. However, the following abstract expression *regress(c0 * sin(v0/v1))*, which must be evaluated in a particle swarm agent, has a fitness score based upon the particle swarm's choice of actual real constant for c0 and the choices of actual columns v0 and v1.

It is our intent, by using an abstract expression grammar with imbedded particle swarm evaluation, to experiment with more fine-grained control during the genetic programming process.

Overview of Symbolic Regression Tool

In (Korns, 2006) we constructed a large agent complex for high volume symbolic regression applications consisting of one million rows and from five to twenty columns. Due to the heavy resources required to evaluate a candidate well-formed-formula (WFF) across one million rows, we cannot afford to evaluate the same candidate twice. Therefore, every WFF which we have

ever evaluated is saved during the course of a single training cycle. All WFF candidates are saved in a collection sorted by their fitness scores. A user option setting restricts the survivor WFF population to the "F" most fit WFF candidates. User option settings support single or multiple island populations and other potentially useful clustering of candidate WFFs. Parenthetically, all the tool's user option settings are available at run time; therefore, it might be possible for the tool to evolve itself, although we have not attempted anything of that nature.

Within the survivor population, mutation and crossover occur in the same fashion as with standard genetic programming. Each WFF survivor is visited once per each evolution. A user-option determines the probability of mutation and another determines the probability of crossover. If warranted, Crossover occurs between the visited individual and another randomly selected individual, from the survivor population. The tool supports multiple grammars in the same training cycle as described previously.

Standard genetic programming practice encourages the use of multiple independent training *runs*. Each run incorporates one initialization step and "G" generational steps during which evolutionary operators are applied. It is standard practice for the experimenter to perform multiple independent training runs of G generations each and then report the results of the fittest individual evolved across all runs (the champion individual).

Since our symbolic regression tool is to be used in a fully automated setting, there can be no human intervention to decide how many independent training runs to perform; therefore, the concept of automatic multiple independent training runs has been incorporated into the tool. A user-option determines the number of evolutions "without fitness improvement" after which the system starts a new independent training run. After each independent training run, the best-of-breed champions from the previous run are saved and the training cycle restarts from scratch. Thus, a training cycle of "G" generations may involve more or fewer separate independent training runs depending on the occurrence of long gaps without fitness improvement.

Vertical Slicing

In (Korns, 2006) we made use of a new procedure *Vertical slicing*. First, the rows in the training matrix X are sorted in ascending order by the dependent values, Y. Then the rows in X are subdivided into S *vertical slices* by simply selecting every Sth row to be in each vertical slice. Thus the first vertical slice is the set of training rows as follows *X[0], X[S], X[2*S], ...* . Each vertical slice makes no assumptions about the underlying probability distribution and each vertical slice contains evenly distributed training examples, in X, across the entire range of *ascending* dependent values, in Y.

Vertical slicing reduces training time by subdividing the training data into "vertical slices" each of which is representative of the whole training data set over ascending values of Y. We then randomly select one of the vertical training data slices as our "sample" training slice; furthermore, we modify each agent WFF and the memo cache to record the sample-fitness. There are now two different fitness scores for each WFF: sample-fitness and fitness. During evaluation, each WFF is first scored on the "sample" training slice and its sample-fitness is recorded. Next the sample-fitness of the WFF is compared to the sample-fitness of the least-fit WFF in the survivor population. If the sample-fitness of the WFF is greater than or equal to the sample-fitness of the least-fit WFF in the survivor population, the WFF is then scored against the entire training data and its true fitness is recorded. This approach will produce false negatives but no false positives.

Adding the AXP Grammar to standard GP

- *Combined:* "Hybrid combination of particle swarm agents and GP"

In this section our goal is to describe the use of our abstract expression grammar, AXP, with standard genetic programming techniques. In a *concrete* grammar expression, such as *regress(EXP)*, obtaining the fitness score requires little more than evaluating the concrete expression once. Therefore the *regress(EXP)* expression can be compiled into a relatively simple agent which evaluates the expression, EXP, at each point and computes the fitness score.

In an *abstract* grammar expression, such as *regress(AXP)*, obtaining the fitness score requires much more than evaluating the abstract expression once. Therefore the *regress(AXP)* expression must be compiled into a complex agent which evaluates the expression, AXP, multiple times at each point and computes the fitness score for each iterative guess at the proper values of the abstract real constants *c0,...,ck* and the proper column choices for each of the abstract variable references *v0,...,vj*.

For instance, the following concrete expression *regress(3.4*sin(x3/x5))*, when evaluated on any point *x=(x0,...,xm)*, has a concrete fitness score. However, the following abstract expression *regress(c0*sin(v0/v1))*, cannot be evaluated on any point *x=(x0,...,xm)*, without choosing concrete values for the *c0, v0, and v1*.

A straightforward method for obtaining a fitness score for this abstract expression, first compiles *regress(c0*sin(v0/v1))* into an agent, then selects three substitutions *(c0=3.4, v0=x3, v1=x5), (c0=-.89, v0=x10, v1=x2), (c0=302.24, v0=x0, v1=x9)* at random, proceeds to evaluate each of the three substitutions, and finally selects the one substitution with the highest fitness. One could then cache the original abstract expression, *regress(c0*sin(v0/v1))*, and its final

fitness score, along with a memo denoting the chosen substitution, *(c0=-.89, v0=x10, v1=x2)* which allowed the abstract expression to achieve the fitness score.

The techniques of particle swarm optimization (Eberhart et al., 2001) and differential evolution (Price et al., 2005) offer an approach to training agents which have been compiled from an abstract grammar such as AXP. Both particle swarm and differential evolution techniques can be applied in both a discrete or a continuous vector space. We support either technique at the option of the user.

The compiler prepares WFF agents for optimization by recognizing each abstract constant and each abstract variable reference. Assuming that after compilation there are K abstract constants and J abstract variable references. The compiled agent will contain a vector, *C*, of real values of length K, and a vector, *V*, of integer values of length J. The *C* and *V* vectors will be used to memoize the concrete choices for each abstract constant and each abstract variable reference. The WFF agent's code is then compiled with indirect indexed references into the *C* and *V* vectors. For instance, the expression, *regress(c0*sin(v0/v1))*, is compiled as, *regress(C[0]*sin(x[V[0]]/x[V[1]]))*. Assuming that the compiled WFF agent contains vectors *C=(3.4,5.6,-2.5)* and *V=(3,2,0,1)*, after particle swarm or differential evolution have made their choices, then the indirect indexed code, *regress(C[0]*sin(x[V[0]]/x[V[1]]))*, is equivalent to, *regress(3.4*sin(x[3]/x[2]))*, which is equivalent to, *regress(3.4*sin(x3/x2))*.

Detailed descriptions of the REG, MVL, and EXP grammars plus our parameters for standard GP and our methods of managing well-formed-formula (WFF) agent candidates can be found in (Korns, 2006). Clearly when we add abstract WFFs to the system we add a layer of evolved optimization underneath that of the genetic programming. In our system the handoff is seamless and occurs when the GP machinery tells the compiled WFF agent to compute its fitness score.

In adding particle swarm and differential evolution to our system, the basic algorithmic components are abstract and concrete WFFs, a memo cache of WFFs, the survivor population of the fittest WFFs, and a list of champion WFFs. We used the *regressGSOSR* option settings which are almost directly in line with (Koza, 1992) and are as follows. At the initialization step of every training "run," exactly 1000 randomly generated WFFs, in the *REG(AXP)* grammar, are evaluated.

Evaluation of each *REG(AXP)* candidate involves a particle swarm (PS) or differential evolution (DE) optimization within the candidate agent. After evaluation, the *C* and *V* vectors will be filled with the concrete choices (for the abstract constants and the abstract variable references) which result in the best fitness score. The size of the PS and/or DE population pool is 25 and the number of generations to optimize is also 25.

All evaluated WFFs are memoized (saved in a memo cache) and also saved in sorted order by fitness score (in the survivor population). The top twenty-five WFFs participate in the genetic operations of mutation and crossover (Koza, 1992) during each incremental generation. The probability of mutation is 10%, and the probability of crossover is 100%. When a WFF is chosen for crossover, its mate is chosen at random from the WFFs of lower fitness within the top twenty-five fittest individuals (in the survivor population). Crossover is always performed twice with the same parents and always produces two children which are evaluated, memoized, and saved in sorted order by fitness score (in the survivor population). The maximum number of generations before training halts is provided at the start of training time. If ten generations pass with no change in the fittest WFF, then system saves the fittest WFF in its list of champions, clears all WFFs in the survivor population (but not the memo cache) and evaluates 1000 randomly generated WFFs, starting a new "run". Any new "run" does not reset the generation count. Training always proceeds until the maximum number of generations have been reached. If G represents the maximum number of generations allowed for a fully automated training cycle, then the maximum number of independent "runs" is (G/10). Depending upon the progress in training, there may only be a single "run" during the entire training process. At the completion of training, the fittest champion WFF (the fittest WFF ever seen) is chosen as the result of the training process.

Adding Context-Aware Crossover

In (Majeed and Ryan, 2006) an extension of standard GP crossover is devised. In standard GP crossover (Koza, 1992), a randomly chosen snip of genetic material from the father s-expression is substituted into the mother s-expression in a random location. In context-aware crossover, a randomly chosen snip of genetic material from the father s-expression is substituted into the mother s-expression *at all possible valid locations*. Where standard crossover produces one child per operation, context-aware crossover can produce many children *depending upon the context*.

Context-aware crossover holds-forth the promise of greater coverage of the local search space, as defined by the candidate s-expressions' roots and branches, and therefore a greater control of the evolutionary search at a fine–grained level.

We further extended context-aware crossover such that *all possible valid* snips of genetic material from the father S-expression are substituted into the mother s-expression *at all possible valid locations*. Whereupon *all possible valid* snips of genetic material from the mother S-expression are then substituted into the father S-expression *at all possible valid locations*.

As an example, this extended context-aware crossover between a father *regress((v0 + c0) / v1)* and a mother *regress(sin(v2*c1))* produces the following children:

- regress(sin(v2*c1) / v1)
- regress((sin(v2*c1) + c0) / v1)
- regress((v0 + sin(v2*c1)) / v1)
- regress((v0 + c0) / sin(v2*c1))
- regress(v2 / v1)
- regress((v2 + c0) / v1)
- regress((v0 + v2) / v1)
- regress((v0 + c0) / v2)
- regress(c1 / v1)
- regress((c1 + c0) / v1)
- regress((v0 + c1) / v1)
- regress((v0 + c0) / c1)
- regress(sin(((v0 + c0) / v1)*c1))
- regress(sin(v2*((v0 + c0) / v1)))
- regress(sin((v0 + c0)*c1))
- regress(sin(v2*(v0 + c0)))
- regress(sin(v0*c1))
- regress(sin(v2*v0))
- regress(sin(c0*c1))
- regress(sin(v2*c0))
- regress(sin(v1*c1))
- regress(sin(v2*v1))

We add our extended context-aware crossover to all GP runs in our system with a varying probability by generation. During the first generation of every GP run, the probability of extended context-aware crossover is 100%. This probability declines linearly until the probability of the extended context-aware crossover is 0% during the final generation.

Boosting Using Island GP with Multiple Grammars

We introduce genetic diversity via boosting across multiple independent island populations. There is a correlation between the fittest individuals in a training run and genetic diversity in the population (Almal et al., 2005). Furthermore the fittest individuals, in a training run tend to cluster around a set of common root expressions (Daida, 2004) and (Hall and Soule, 2004). Capitalizing on these observations, we set our symbolic regression tool to support boosting across multiple islands with the simple linear regression, *REG*, grammar.

We use as many independent islands as there are columns in the regression problem. Each island is evolved for a total of 10 generations using the abstract grammar *REG(AXP)*. If there are *M* columns, we repeat the boosting process *M* times. When the mth island population has produced its champion WFF agent, the estimation vector *EY* is subtracted from the dependent variable *Y* to produce *Ym* the new dependent variable for the $m+1$ island population to regress upon.

At the termination of the Mth island symbolic regression, we have M abstract simple linear WFF champions each of which have regressed on the boosted dependent variable. We now assume that the following multiple linear regression will best model that original dependent variable.

- mvlregress(wff0,...,wffM)

Each of the M champions are converted from their abstract AXP grammar representations into concrete EXP grammar representations, and a single *mvlregress(wff0,...,wffM)* candidate, known as *Eve*, is entered into the final island as the first individual. Using extended context-aware crossover on the individual WFF S-expressions contained in *Eve*, we create and additional 1000 individuals. A standard GP run, as in (Korns, 2006), is then run for 10 generations and the resulting most fit *mvlregress(wff0,...,wffM)* model is chosen as our final champion.

Final Results

Our final experiment was to use the system with multiple island boosting, using the simple linear REG(AXP) grammar and then populate a final "island of champions" using the multiple regression MVL(EXP0,...,EXPM) grammar[1]. The results of training on the nine test cases, using the *regressGSOBOOST* option settings on 1 million rows and twenty columns with 40% random noise, are shown in the table below.

[1]This entire experimental setup can be chosen by selecting the system's *regressGSOBOOST* option settings.

Table 4-2. Result For 1M rows by 20 columns Random Noise

Test	*Minutes*	*Train-Error*	*Test-Error*	*Classify*
cross	2820	0.83	0.67	0.72
cubic	2278	0.39	0.40	0.91
hyper	2154	0.85	0.86	0.47
ellipse	3171	0.70	0.55	0.82
hidden	2386	0.11	0.00	0.99
linear	2400	0.10	0.01	0.99
mixed	2845	0.67	1.55	0.64
ratio	2582	0.30	0.94	0.00
cyclic	2336	0.43	0.32	0.05

Description of table headings:

- *Test:* The name of the test case
- *Minutes:* The number of minutes required for training
- *Train-Error:* The average percent error score for the training data
- *Test-Error:* The average percent error score for the testing data
- *Classify:* The classification score for the testing data

Fortunately, training time is mostly within our 3000 minute (50 hour) limit (only the *ellipse* test case is slightly over). In general, average percent error performance is poor with the *linear* and *hidden* problems showing the best performance. Extreme differences between training error and testing error in the *mixed* and *ratio* problems suggest over-fitting. Surprisingly, long and short classification is fairly robust in most cases with the exception of the *cyclic* and *ratio* test cases. If we were to run a market neutral hedge on hypothetical markets, driven by these nine test models, we would have lost money in none of the markets, broken even in the markets driven by the *ratio* and *cyclic* models, and made good money in all other markets.

We were nearly prescient on the *linear*, *hidden*, and *cubic* market models realizing over 90% of theoretically possible profits. We achieved more than 60% of theoretically possible profits even in the more difficult *cross*, *ellipse*, and *mixed* market models.

Summary

Genetic Programming, from a corporate perspective, is almost ready for industrial use on large scale, time constrained symbolic regression problems.

Adapting the latest research results has created a symbolic regression tool whose promise is exciting. Financial institutional interest in the field is growing while pure research continues at an aggressive pace. Further applied research in this field is absolutely warranted.

Clearly we need to experiment with techniques which will improve our performance on the *mixed* and *cyclic* test cases. Areas for future research include: (i) using standard statistical and Bayesian analysis to help build conditional WFFs for multi-modal markets and (ii) experimentation with additional experimentation with grammar expressions to increase the speed of evolutionary training and to develop a better understanding of hill-climbing operators from a root grammar viewpoint.

Acknowledgements

This work was completed with the generous help of Conor Ryan who educated us on Context-Aware Crossover and Timothy May who pioneered the Strategy Central Project at Investment Science Corp.

References

Almal, A., Worzel, W. P., Wollesen, E. A., and MacLean, C. D. (2005). Content diversity in genetic programming and its correlation with fitness. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 12, pages 177–190. Springer, Ann Arbor.

Caplan, Michael and Becker, Ying (2004). Lessons learned using genetic programming in a stock picking context. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 6, pages 87–102. Springer, Ann Arbor.

Daida, Jason (2004). Considering the roles of structure in problem solving by a computer. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 5, pages 67–86. Springer, Ann Arbor.

Eberhart, Russell, Shi, Yuhui, and Kennedy, James (2001). *Swarm Intelligence*. Morgan Kaufman, New York.

Hall, John M. and Soule, Terence (2004). Does genetic programming inherently adopt structured design techniques? In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 10, pages 159–174. Springer, Ann Arbor.

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Majeed, Hammad and Ryan, Conor (2006). Using context-aware crossover to improve the performance of GP. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 847–854, Seattle, Washington, USA. ACM Press.

O'Neill, Michael and Ryan, Conor (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Dordrecht Netherlands.

Price, Kenneth, Storn, Rainer, and Lampinen, Jouni (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer, New York.

Yu, Tina, Chen, Shu-Heng, and Kuo, Tzu-Wen (2004). Discovering financial technical trading rules using genetic programming with lambda abstraction. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 2, pages 11–30. Springer, Ann Arbor.

Chapter 5

SOLVING COMPLEX PROBLEMS IN HUMAN GENETICS USING GENETIC PROGRAMMING: THE IMPORTANCE OF THEORIST-PRACTITIONER-COMPUTER INTERACTION

Jason H. Moore[1], Nate Barney[1] and Bill C. White[1]

[1]*Dartmouth College, One Medical Center Drive, HB7937, Lebanon, NH 03756 USA*

Abstract Genetic programming (GP) shows great promise for solving complex problems in human genetics. Unfortunately, many of these methods are not accessible to biologists. This is partly due to the complexity of the algorithms that limit their ready adoption and integration into an analysis or modeling paradigm that might otherwise only use univariate statistical methods. This is also partly due to the lack of user-friendly, open-source, platform-independent, and freely-available software packages that are designed to be used by biologists for routine analysis. It is our objective to develop, distribute and support a comprehensive software package that puts powerful GP methods for genetic analysis in the hands of geneticists. It is our working hypothesis that the most effective use of such a software package would result from interactive analysis by both a biologist and a computer scientist (i.e. human–human–computer interaction). We present here the design and implementation of an open-source software package called Symbolic Modeler (SyMod) that seeks to facilitate geneticist–bioinformaticist–computer interactions for problem solving in human genetics. We present and discuss the results of an application of SyMod to real data and discuss the challenges associated with delivering a user-friendly GP-based software package to the genetics community.

Keywords: Genetic Analysis, Genetic Epidemiology, Genetic Programming, Open-Source Software, Symbolic Discriminant Analysis, Symbolic Regression

1. Introduction

Human genetics is transitioning away from the study of single-gene Mendelian diseases such as cystic fibrosis to tackling common complex diseases such as cancer and cardiovascular disease that represent the majority of the public health burden. The transition to more complex diseases and the widespread availability of high-throughput technologies for measuring genes necessitates powerful analytical methods that are able to model the relationship between multiple genetic and environmental factors and susceptibility to disease in the context of high-dimensional datasets. The ultimate goal of these endeavors is the identification and characterization of genetic risk factors that can be used to improve the detection, prevention and treatment of disease.

Genetic algorithms, genetic programming, and other biologically-inspired machine learning methods show great promise for solving complex biomedical problems (Fogel and Corne, 2003). This is especially true in human genetics where these methods have been used to identify genetic risk factors for disease. Unfortunately, many of these methods are not accessible to biologists and biomedical researchers for applied studies. This is partly due to the complexity of the algorithms that limit their ready adoption and integration into an analysis or modeling paradigm that might otherwise only use univariate statistical methods. This is also partly due to the lack of user-friendly, open-source, platform-independent, and freely-available software packages that are designed to be used by biologists for routine analysis.

It is our objective to develop, evaluate, distribute and support a comprehensive software package that puts powerful genetic programming methods for the genetic analysis of complex human diseases in the hands of geneticists and epidemiologists. It is our working hypothesis that the most effective use of such a software package would result from real-time interactive analysis by both a biologist and a computer scientist (i.e. human–human–computer interaction). We present here the design and implementation of an open-source software package called Symbolic Modeler (SyMod) that seeks to facilitate geneticist-bioinformaticist-computer interactions for complex problem solving in the domain of human genetics (Section 2). We then present a real-world example of how this software has been applied to modeling combinations of DNA sequence variations in a genetic study of atrial fibrillation (Section 3). We end with a discussion of the lessons learned from the development and application of SyMod (Section 4) and some ideas about the barriers to moving toward better human–human–computer interaction.

2. The Symbolic Modeler (SyMod) Software Package

Design Objectives

Our goal was to develop a software package that would make available powerful genetic programming methods for data mining and machine learning to the human genetics community. There were several important objectives to the software design and development. First, it was important for the software to be platform-independent. Second, it was important for the software to include a user-friendly graphic-user interface (GUI) in addition to a simple command-line interface that could be scripted. Third, it was important to include publication quality graphical output in the GUI. Fourth, it was important to include a number of configuration options for the expert user. Finally, it was important for the software to be able to generate and use expert knowledge that can be used to help guide the algorithms.

Data Mining Methods

There are two primary data mining methods implemented in SyMod. The first, symbolic discriminant analysis (SDA), was developed as a flexible alternative to linear discriminant analysis for modeling discrete outcomes or classes (Moore et al., 2002; Moore et al., 2001; Moore and Parker, 2001; Reif et al., 2003; Reif et al., 2004). With SDA, the user provides a list of mathematical functions and attributes that are used to build symbolic discriminant functions using a search algorithm such as genetic programming. Quality or fitness of an SDA model can be assessed by estimating the accuracy of the classifier using methods such as cross-validation. Here, accuracy is defined as $(TP + TN)/(TP + TN + FP + FN)$ where TP are true positives, TN are true negatives, FP are false positives, and FN are false negatives. Balanced accuracy can be used for imbalanced datasets. The goal is to find an SDA model that has a maximum accuracy. Ideally, this is assessed using a testing or replication dataset to determine the generalizability of the model. This helps to address issues associated with overfitting.

The second method implemented in SyMod is symbolic regression. Symbolic regression is similar to SDA except that the endpoint that is modeled is continuous. The goal of symbolic regression is to identify a best fitting regression model that can take any shape given a set of candidate functions and attributes. As with SDA, it is common to use a stochastic search algorithm such as genetic programming for identifying the best fit model. Here, we use sums of squared error (SSE) as the measure of model quality or fitness although other measures such as R^2 could be used. The goal is to identify a symbolic regression equation that minimizes the SSE. As with SDA, this is ideally assessed using independent data to address overfitting.

Genetic Programming

SyMod uses genetic programming (GP) as a stochastic search algorithm for identifying optimal SDA and symbolic regression models. Genetic programming is an automated computational discovery tool that is inspired by Darwinian evolution and natural selection (Banzhaf et al., 1998; Koza, 1992; Koza, 1994; Koza et al., 1999; Koza et al., 2003a; Langdon, 1998; Langdon and Poli, 2002). The goal of GP is to evolve computer programs to solve problems. This is accomplished by first randomly generating computer programs that are composed of the building blocks needed to solve or approximate a solution to a problem. Each randomly generated program is evaluated, and the good programs are selected and recombined to form new computer programs. This process of selection based on fitness and recombination to generate variability is repeated until a best program or set of programs is identified. Genetic programming and its many variations have been applied successfully to a wide range of different problems including data mining (Freitas, 2002) and bioinformatics (Fogel and Corne, 2003). The advantage of GP and other evolutionary computing algorithms is that they carry out a parallel or beam search of the fitness landscape by considering hundreds or thousands of solutions simultaneously. Recombination makes it possible to sample multiple peaks in a rugged fitness landscape, which is desirable for most complex biological problems. Here, symbolic discriminant and symbolic regression equations are represented as expression trees.

Cross-Validation Strategy

As mentioned above, assessing the generalizability of a model plays an important role in addressing problems associated with overfitting the data. This is especially true for computational intelligence strategies such as SDA and symbolic regression. Implementing standard cross-validation methods is difficult for stochastic search algorithms such as GP, because it is likely that models with different functional forms will be discovered for each of the *n* partitions of the data. This makes model selection and model interpretation difficult. With SDA, this has been previously addressed by first running SDA m times with n-fold cross-validation and then selecting the $m * n$ best models (Moore et al., 2002). Then, the number of times each attribute is discovered across the $m * n$ models is recorded, and a threshold used to select the most interesting attributes. The number of times an attribute is discovered in n cross-validation intervals has been called cross-validation consistency or CVC (Moore, 2003; Ritchie et al., 2001). In this example, permutation testing was used to establish an empirical significance level that was used as the threshold. Thus, only those attributes that were discovered more than would be expected by chance in random data were selected. Finally, SDA was run a final time on the whole dataset using just

the significant attributes. This approach was successful for mining real data (Moore et al., 2002; Schwartz et al., 2005).

Another approach to the overfitting problem is to limit the order or size of the models being evaluated so that bigger models are not competing against smaller models in the same population. This is an approach that has been successfully implemented previously for other data mining and machine learning methods such as multifactor dimensionality reduction (MDR) (Moore, 2007; Ritchie et al., 2001). We have implemented this approach in SyMod and describe it briefly here. First, we limit the GP to generating and evaluating full expression trees at a certain depth (e.g. depth one, with a root node and two children). All models are constrained to this size and shape. Models of this size are evaluated using a three-fold cross-validation framework similar that of Rowland (Rowland, 2003). Here, the data are split into three equal pieces that are used for training, testing, and validation. The top n models from the GP search using the training set are selected based on their training accuracy. Each of these n models is then evaluated using the testing set and assigned a testing accuracy. The model with the best testing accuracy is then evaluated using the validation set and assigned a validation accuracy. These three accuracy estimates are reported for the single best model of the given order. This is repeated for each order or tree depth (e.g. depth one, two, three, and four). The advantage of this approach is that smaller models are not competing with larger models, which are likely to have better training accuracies simply because they include more attributes. Once the best model of each order is identified, they can then be safely compared using the validation accuracy, which should be less likely to be influenced by model size.

Exploiting Expert Knowledge

Genetic programming and other evolutionary computing methods work best when there are good building blocks present in the population that can be recombined to discover good models. When building blocks are not present or are poorly defined a GP may not perform any better than a random search. This is consistent with previous experiments in the domain of human genetics, for example, where interactions among attributes may be more important than independent effects (Moore, 2007; Moore and White, 2006b; Moore and White, 2006a; White et al., 2005). This is also consistent with the idea of a competent genetic algorithm (cGA) reviewed by Goldberg (Goldberg, 2002). Goldberg argues that understanding and exploiting building blocks (schemata) is essential to the success of GAs and by extension to GP (Sastry and Goldberg, 2003). There are two important issues here. The first issue is to make sure the building blocks needed to construct good solutions are present. The second is to make sure the good building blocks are used and exploited during evolution.

It was noted by O'Reilly et al. (O'Reilly et al., 2005) that providing rewards for building blocks is necessary for complex adaptation. As such, we have implemented several methods for generating and exploiting expert knowledge in SyMod.

Expert knowledge can come from multiple different sources including pre-computed attribute quality or weights that come from knowledge about the biological function or importance of the attributes (e.g. genes). SyMod can generate expert knowledge using the ReleifF algorithm, and can accept expert knowledge from any external source such as an input file. The availability of domain-specific expert knowledge raises the question of the best way to use it in a GP. This is a topic that has received some attention in recent years. Jin (Jin, 2006) covers the use of expert knowledge in population initialization, recombination, mutation, selection, reproduction, multi-objective fitness functions, and human-computer interaction, for example. Here, we have implemented sensible initialization of the initial population, multi-objective fitness functions, and selection. We have shown previously that these are all effective strategies for improving the performance of GP when there are otherwise no building blocks present in genetic studies of human disease (Moore, 2007; Moore and White, 2006b; Moore and White, 2006a).

Software Implementation

The data mining and genetic programming methods described above were implemented in a user-friendly and open-source software package that was programmed entirely in Java. A software engineering protocol was followed that included design specifications, programming, and software testing. The Symbolic Modeler (SyMod) software package is available for download from `www.epistasis.org` or `www.symbolicmodeler.org`.

SyMod - Configuration Tab. The configuration tab, shown in Figure 5-1, allows the user to quickly choose global parameter settings, search parameter settings, functions for the function set and settings for the use of expert knowledge. In the Global Parameters box the user can set the random seed, the range of tree depths to be evaluated, and a range of constants to be included along with a probability that defines the likelihood that a constant will be picked instead of an attribute for a terminal in the expression tree. Included is an option to include the mean of the dependent variable as a constant for regression problems. Also included in this box is a spinner to set the landscape size. This allows the user to set how many best models are evaluated from the training set. The check box labeled Parallel allows the user to turn off the option to run the algorithm in parallel if the computer being used has multiple processors.

The Search Parameter box allows the user to set parameters for the stochastic search algorithm to be used. Currently, GP and random search are the only

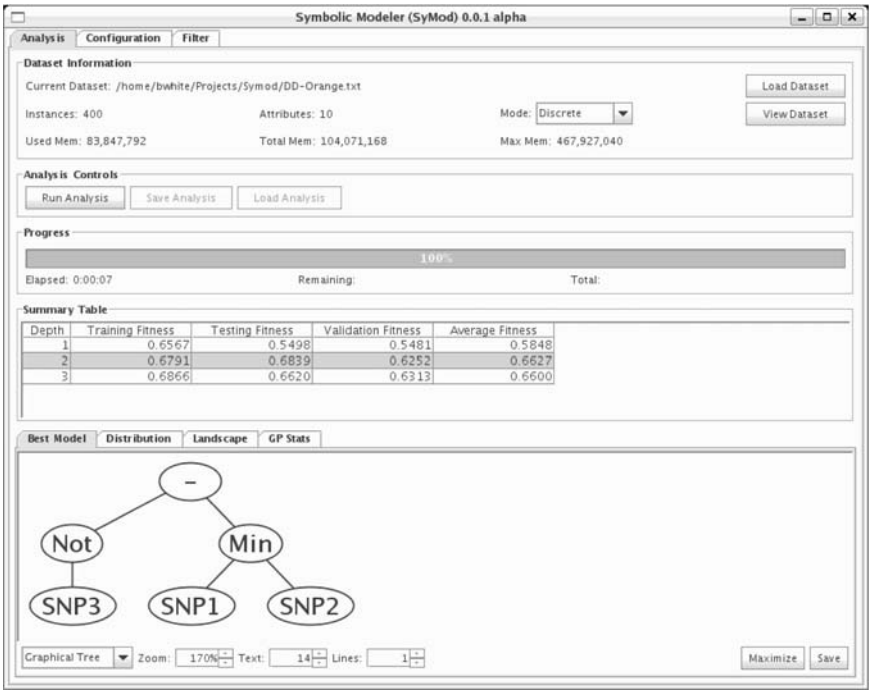


Figure 5-1. Screenshot of the SyMod software Configuration tab.

options. Other stochastic search algorithms will be added later. Here, the population size, generations, crossover rate, mutation rate, and whether to use an elitist strategy can be set when GP is used. For random search the user can select a total number of random models to be evaluated.

The Function Set box allows the user to select among a number of different mathematical functions for the function set. Individual functions can be toggled on or off, or entire sets can be selected or deselected. Since the software is open-source other functions can easily be added to this list.

The Expert Knowledge box allows the user to load a file containing weights for each attribute in the dataset. This information can be used to help guide initialization, fitness calculations and/or selection.

SyMod - Analysis Tab. Once a user has selected the configuration options and has run an analysis the results are displayed and can be explored in the Analysis tab, shown in Figure 5-2. The first box shows the Dataset Information. This is where the user loads a dataset. The number of instances and attributes in a dataset is shown along with whether the class attribute is discrete or continuous. This determines whether SDA or symbolic regression is used. Also illustrated is the memory usage of the software.

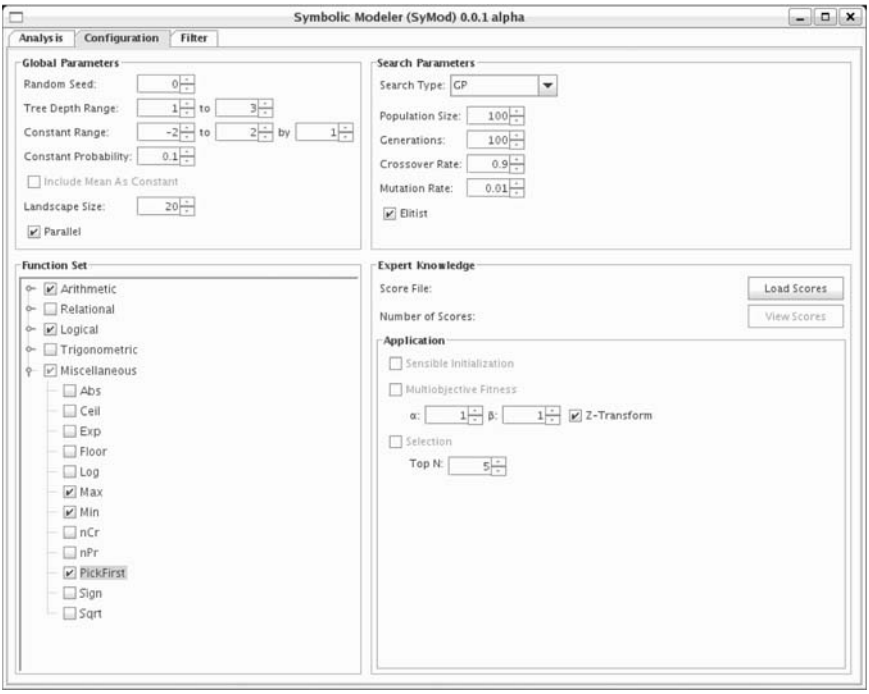


Figure 5-2. Screenshot of the SyMod software Analysis tab.

The Analysis Controls box allows the user to launch and stop an analysis while the Progress box provides a progress bar and the estimated time to completion for a run.

The Summary Table summarizes the results for each of the best models. The first column of the summary table shows the depth of the expression tree while the remaining columns show the fitness of the model for each of the three intervals of the data along with the average. This information can be used to select an overall best model.

The bottom panel of the Analysis tab provides different types of output to allow the user to more completely visualize and assess the analysis results. The Best Model tab shows the best model as a tree or as a function in infix notation or as an S-expression. The distribution tab shows the distribution of symbolic discriminant scores for each class for and SDA analysis or shows a plot of the predicted versus actual values for symbolic regression. The raw data can also be visualized as text. The Landscape tab plots the fitness values for all best models considered. The GP Stats tab shows the change in best, worst, and average fitness across generations. This is useful for assessing the variability of the GP population. All graphical and text results can be saved to a file.

SyMod - Filter Tab. The Filter tab provides tools for computing the quality of the attributes in the dataset using the ReliefF algorithm. This can be used to filter a list of attributes or to provide quality estimates that can be used as expert knowledge to help guide the GP search. All results can be visualized and saved to files. Additional filters such as entropy-based measures and statistical filters such as chi-square will be added later.

3. Application of SyMod to the Genetic Analysis of Atrial Fibrillation

No one stochastic search algorithm is optimal for every fitness landscape. Despite its parallel nature, GP is no exception to this rule. Given this general limitation, there are steps that can be taken to improve the chances of success. For example, one common mistake is to make an *a priori* assumption about the parameter setting for a stochastic search algorithm. One of the most important parameters in GP is the population size. How many solutions should be generated and evolved for a given problem? How many generations should the GP run? What is the optimal function set for symbolic modeling? How big should the models be? We have previously developed a five-step framework for implementing SyMod in combination with other statistical and computational algorithms for competent problem-solving in human genetics (Moore et al., 2007). Collectively, this multi-step process tailors GP for solving complex genetic modeling problems. Strategies such as these are being used to enhance the ability of GP to solve complex problems in finance and engineering, for example (Yu et al., 2006). We briefly describe the five-step framework and then review its application to a real dataset.

A Five-Step Framework for Symbolic Modeling

The goal of the five-step framework is to provide a competent or intelligent approach to symbolic model discovery that (1) employs a full factorial experimental design to optimize search parameters, (2) carries out a coarse-grained search using GP, (3) generates expert knowledge by statistically modeling best solutions, (4) carries out a fine-grained stochastic search using an estimation of distribution algorithm (EDA), and (5) uses function mapping and interaction dendrograms to interpret symbolic models (Moore et al., 2007).

Step 1: Experimental Design for Parameter Optimization. The goal of Step 1 is to determine the optimal parameter settings for the GP using a full factorial experimental design. Here, different population sizes, different generation lengths, different tree depths, and different function sets are evaluated. For each combination of factors the GP is run using 10 different random seeds and for each run the best model along with its average accuracy is recorded.

To determine which of the factors is a significant predictor of GP performance a four-way ANOVA for fixed effects is employed. Tukey's HSD is used for post-hoc analysis. All results are considered statistically significant at a type I error rate of 0.05.

Step 2: Coarse-Grained Search using Genetic Programming. Once the parameter settings are established in Step 1 the GP is run m times with different random seeds. The best model from each run is selected using the three-way cross-validation framework described above. The n best models are ranked according to their average accuracy and the best m selected for further consideration in Step 3.

Step 3: Generation of Expert Knowledge using Graph-Based Modeling. The goal of Step 3 is to develop a graphical model of the n best SDA models generated in Step 2. What can we learn from the m best models? Are there functions and/or attributes that show up frequently? Are there consistent bivariate dependencies among functions or among functions and attributes across the n trees? Across the n best trees we record the percentage of time each single element or each adjacent pair of elements is present. This information is used to draw a directed graph. The directed graph represents expert knowledge about the problem and its solutions. Each node in the graph is a function, an attribute, or a constant. The edges (connections) in the graph are directed from parent nodes to child nodes. For example, an arrow from the function $+$ to the constant 2 indicates that 2 was a child of $+$. A threshold of 1%, for example, can be employed to show only the most frequent connections between nodes.

Step 4: Fine-Grained Search using an Estimation of Distribution Algorithm. The goal of Step 4 is to carry out a fine-grained stochastic search of the fitness landscape using the expert knowledge generated in Step 3. That is, we now want to sample more completely the regions of the search space that were defined by the coarse-grained search. It is well-known that GP is an excellent coarse-grained search algorithm but is not particularly good at fine-tuning. We carry out the fine-grained search by first transforming the directed graph to a probability distribution function (pdf) for univariate and bivariate components of the expression trees that met the 1% threshold criteria. Using pdfs to sample search spaces has been referred to as estimation of distribution algorithms or EDAs (Larrañga and Lozano, 2002). Here, the pdf is sampled n times to generate expression trees with a given maximum depth. The accuracy of each tree is estimated using the entire dataset. At this stage of the modeling process cross-validation has already been employed to choose attributes and functions, and thus there is less concern with overfitting.

Step 5: Model Interpretation using Function Mapping and Interaction Dendrograms. An important criticism of many machine learning methods is that the models represent a black-box that is not interpretable. Expression trees generated using GP often have the same criticism. The goal of Step 5 is to provide a statistical interpretation of the best models generated using the previous four steps. The novel ''function mapping" approach starts by showing the levels of each attribute in the model (Moore et al., 2007). Also shown is the accuracy associated with that attribute alone and its associated odds ratio and 95% confidence interval. This facilitates a quick assessment of the magnitude of the univariate effects in relation to the multivariate effects. Next, each combination of inputs is shown with the corresponding function at each node in the expression tree. The mapping of the inputs and the range of output values is illustrated. The accuracy and odds ratio for the output of each node is shown. As a whole, the function mapping tree summarizes the mapping of inputs and outputs along with their corresponding effects on the endpoint, so that the tree can be decomposed and interpreted. The root node provides the final output values that are then used as discriminant scores to classify cases and controls. We show the distribution of cases and controls for each discriminant score and the corresponding classification label.

The function mapping method permits a tree to be decomposed into its component parts. However, it does not provide information about which parts of the tree contain synergistic, redundant, or independent pieces of information. As a final interpretation step the output of each node is saved as an attribute and then used analyzed using interaction dendrograms to measure the bivariate dependencies as described previously (Jakulin and Bratko, 2003; Moore et al., 2006).

Application to real data

We have previously carried out an analysis of 250 patients with documented nonfamilial structural atrial fibrillation (AF) and 250 controls that were matched to cases on a 1-to-1 basis with regard to age, gender, presence of left ventricular dysfunction, and presence of significant valvular heart disease (Tsai et al., 2004). The goal of the study was to determine whether a set of DNA sequence variations in genes from the renin-angiotensin system are predictive of atrial fibrillation in this dataset. The *ACE gene insertion/deletion (I/D)* polymorphism, the *T174M*, *M235T*, *G6A*, *A-20C*, *G152A*, and *G217A* polymorphisms of the *angiotensinogen* gene, and the *A1166C* polymorphism of the *angiotensin II type I receptor* gene were studied. Tsai et al. (Tsai et al., 2004) provide the details for the study. We previously applied the five-step symbolic modeling approach described above to this dataset (Moore et al., 2007).

Implementation of the parameter sweep in Step 1 determined that a population size of 500, 100 generations, a tree depth of 3, and a function set consisting of arithmetic and relational operators was optimal ($P < 0.05$). In Step 2 these parameter settings were used along with a crossover frequency of 0.9 and a mutation frequency of 0.01 to run SDA in SyMod 200,000 times. A total of 100 best models were selected from these 200,000 runs. In Step 3 we estimated the univariate and bivariate frequencies of nodes and terminals and used this information to draw a directed graph. The directed graph indicated significant patterns of dependencies among the trees. For example, of the attributes, the *ACE I/D*, *T174M*, *G152A*, *G6A*, and *M235T* polymorphisms showed up most frequently and had connections to parent nodes. Of the functions, $=$, $+$, and max showed up most frequently. As for the bivariate relationships, it is interesting to note that the *ACE I/D*, *T174M*, and *G152A* polymorphisms were often children of $>=$, $>$, and $<$ while the *G6A* and *M235T* polymorphisms were not.

In Step 4 we used a simple estimation of distribution algorithm to sample the probability distribution of models defined by the knowledge captured in the directed graph. This simple algorithm starts by probabilistically picking a root node using the univariate information from the directed graph. For each branch of the tree, the EDA decides whether to add another node (i.e. function) or to add a terminal (i.e. attribute) with probability 0.5. The specific node or terminal that is selected once that decision is made is defined by the probabilities from the directed graph. The depth of these trees was limited to six. The final best model had an accuracy of 0.644, a tree depth of six, and consisted of 13 nodes and 16 terminals. It is important to note that the fine-grained search was able to generate a better model than the coarse-grained search. All five of the most frequent attributes from the directed graph are represented in this best tree along with several of the functions. The best function was $((M235T + ACEI/D) * (((G6A = Max(M235T, T174M)) + ((T174M > 1)! = ((G152A * 2) >= ACEI/D)))$.

In Step 5, function mapping and interaction dendrograms were used to interpret the final best model. Visual inspection of the dependencies quickly identified a nonlinear interaction between the *T174M* and *ACE I/D* polymorphisms. This is consistent with the previous analysis of this dataset by the original authors (Tsai et al., 2004), although some new patterns were discovered.

4. Lessons Learned from the Development and Application of SyMod

We have presented a user-friendly open-source software package for symbolic modeling in the domain of human genetics. Each of our design goals was accomplished in the Symbolic Modeler (SyMod) software, yielding a GP-based

modeling tool that is both powerful and easy to use. We first discuss our experience applying SyMod to a real-world biological problem and then provide some ideas about how SyMod can be more effectively used in the domain of human genetics.

Genetic programming is not a push-button problem-solver

It is increasingly clear that the ''vanilla" or basic GP algorithm is not appropriate for solving complex problems such as those in the biomedical sciences. This is evident, for example, when good building blocks are not present in the population, as is the case in data mining when the attributes to be modeled are associated with the class attribute primarily through nonlinear interactions. In this case, the single attributes look no different from the noisy attributes and thus don't represent good building blocks. We have documented this problem in the domain of human genetics (Moore, 2007; Moore and White, 2006b; Moore and White, 2006a). What this means is that GP and other related methods need help in the form of other machine learning methods, and in the form of expert knowledge that can be used to guide the algorithm. For these reasons, GP is not a push-button problem-solver. This has significant implications for the design and implementation of user-friendly software packages such as SyMod.

Although the SyMod software package puts powerful computational intelligence algorithms in the hands of geneticists, it does not represent a comprehensive analysis package. This is illustrated by the five-step modeling process presented above that highlights the importance of a parameter sweep using a formal experimental design. It is often difficult to guess what the optimal population size or number of generations should be, for example, in a GP run. Statistically evaluating this first can save a lot of time with trial and error runs, and can significantly improve the performance. We found with the atrial fibrillation dataset that the choice of population size, the tree depth and the function set were all highly significant predictors of the quality of the SDA models that the GP discovered. The ability to establish and use optimal parameter settings greatly improved the results.

The parameter sweep was successful but may be very difficult to implement in SyMod. This is because carrying out a thorough parameter sweep requires hundreds or even thousands of GP runs that is not practical on a desktop computer. The parameter sweep for the atrial fibrillation analysis required 1890 total runs and was completed on a parallel computer system by running SyMod under different settings on multiple processors with a Perl script. This many runs on a desktop would have been prohibitive.

The parameter sweep was not the only computationally intensive aspect of the five-step modeling process. In Step 2, we ran the GP 200,000 times. In Step 4, we ran the EDA 10^9 times. In fact, the entire five-step process required

approximately four days of computing time on 100 processors. This raises the question of whether it is realistic to assume that a single software package can implement GP for solving complex problems that is both accessible to the practitioner (i.e. the biologist) and computationally feasible on the practitioner's desktop computer.

The importance of human-human-computer interactions

Let's assume that a push-button software package that is able to carry out a modern GP analysis is not practical at this time. How is the data mining and machine learning community going to make these powerful algorithms available to practitioners? We propose that it is possible to make this a reality by making user-friendly software packages that can be used jointly by both a practitioner and a computer scientist. That is, make available software that is intuitive enough for a biologist, for example, to use and powerful enough for a computer scientist to use. To be intuitive to a biologist the software needs to be easy to use and needs to provide output that is visual and easy to navigate. To be powerful the software needs to provide the functionality that would allow a computer scientist the flexibility to explore the more theoretical aspects of the algorithm and to carry out higher-level analyses such as a parameter sweep. The key, however, to the success of any such software package is the ability of the practitioner and the computer scientist to sit down together at the computer and jointly carry out an analysis. This is important for several reasons. First, the practitioner can help answer questions the computer scientist might have that are related to domain-specific knowledge. Such questions might only arise at the time of the analysis and might otherwise be ignored. Similarly, the practitioner might have ideas about specific questions to address with the software that might only be feasible with the help of the computer scientist. For example, a question that requires multiple processors to answer might need the assistance of someone with expertise in parallel computing.

The idea that practitioners and computer scientists should work together is not new. Langley (Langley, 2002) has suggested five lessons for the computational discovery process. First, traditional machine learning notations are not easily communicated to scientists. This is important because a computer science model may not be interpretable by a practitioner. Second, scientists often have initial models that should influence the discovery process. Domain-specific knowledge can be critical to the discovery process. Third, scientific data are often rare and difficult to obtain. It often takes years to collect and process the data to be analyzed. As such, it is important that the analysis is carefully planned and executed. Fourth, scientists want models that move beyond description to provide explanation of data. Explanation and interpretation are paramount to the practitioner. Finally, scientists want computational as-

sistance rather than automated discovery systems. Langley (Langley, 2002) suggests that practitioners want interactive discovery environments that help them understand their data while at the same time giving them control over the modeling process. Collectively, these five lessons suggest that synergy between the practitioner and the computer scientist is critical. This is because each has important insights that may not get expressed or incorporated into the discovery process if either carries out the analysis in isolation.

The ultimate goal for the development and distribution of SyMod is to carefully consider our own lessons and the lessons of others such as Langley (Langley, 2002) to produce a software package that is truly user-friendly, powerful, and that can be used effectively by practitioners and computer scientists working together at the same time.

Acknowledgment

This work was supported by grants AI59694 and LM009012from the National Institutes of Health. We thank Mr. Todd Holden for expert assistance with some of the design aspects of SyMod.

References

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA.

Fogel, G.B. and Corne, D.W. (2003). *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann Publishers.

Freitas, Alex (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag.

Goldberg, D. E. (2002). *The Design of Innovation*. Kluwer.

Jakulin, A. and Bratko, I. (2003). Analyzing attribute interactions. *Lecture Notes in Artificial Intelligence*, 2838:229–240.

Jin, Y. (2006). *Multi-Objective Machine Learning*. Springer.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Langdon, William B. (1998). *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston.

Langley, P. (2002). Lessons for the computational discovery of scientific knowledge. *Proceedings of First International Workshop on Data Mining Lessons Learned*, pages 9–12.

Larrañga, P. and Lozano, J.A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston.

Moore, J. H. (2007). Genome-wide analysis of epistasis using multifactor dimensionality reduction: feature selection and construction in the domain of human genetics. In *Knowledge Discovery and Data Mining: Challenges and Realities with Real World Data*. IGI.

Moore, J. H., Gilbert, J. C., Tsai, C.-T., Chiang, F. T., Holden, W., Barney, N., and White, B. C. (2006). A flexible computational framework for detecting, characterizing, and interpreting statistical patterns of epistasis in genetic studies of human disease susceptibility. *Journal of Theoretical Biology*, 24:252–261.

Moore, J.H. (2003). Cross validation consistency for the assessment of genetic programming results in microarray studies. *Lecture Notes in Computer Science*, 2611:99–106.

Moore, J.H, Barney, N., Tsai, C.T, Chiang, F.T, Gui, J., and White, B.C (2007). Symbolic modeling of epistasis. *Human Heridity*, 63(2):120–133.

Moore, J.H. and Parker, J.S. (2001). *Evolutionary computation in microarray data analysis*. Kluwer Academic Publishers, Boston.

Moore, J.H., Parker, J.S., and Hahn, L.W. (2001). Symbolic discriminant analysis for mining gene expression patterns. *Lecture Notes in Artificial Intelligence*, 2167:191–205.

Moore, J.H, Parker, J.S., Olsen, N.J, and Aune, T. (2002). Symbolic discriminant analysis of microarray data in autoimmune disease. *Genetic Epidemiology*, 23:57–69.

Moore, J.H. and White, B.C. (2006a). Exploiting expert knowledge in genetic programming for genome-wide genetic analysis. *Lecture Notes in Computer Science*, 4193:969–977.

Moore, J.H. and White, B.C. (2006b). *Genome-wide genetic analysis using genetic programming: The critical need for expert knowledge.* Springer.

O'Reilly, U.-M., Yu, T., Riolo, R., and Worzel, B. (Eds.) (2005). *Genetic Programming: Theory And Practice*. Springer.

Reif, D.M, White, B.C., and Moore, J.H. (2004). Integrated analysis of genetic, genomic, and proteomic data. *Expert Review of Proteomics*, 1:67–75.

Reif, D.M, White, B.C., Olsen, N.J., Aune, T.A., and Moore, J.H. (2003). Complex function sets improve symbolic discriminant analysis of microarray data. *Lecture Notes in Computer Science*, 2724:2277–2287.

Ritchie, M. D., Hahn, L. W., Roodi, N., Bailey, L. R., Dupont, W. D., Parl, F. F., and Moore, J. H. (2001). Multifactor dimensionality reduction reveals high-order interactions among estrogen metabolism genes in sporadic breast cancer. *American Journal of Human Genetics*, 69:138–147.

Rowland, J.J. (2003). Model selection methodology in supervised learning with evolutionary computation. *Biosystems*, 72(1-2):187–196.

Sastry, K. and Goldberg, D. E. (2003). Probabilistic model building and competent genetic programming. *Genetic Programming Theory and Practice*.

Schwartz, S.A., Weil, R.J., Thompson, R.C., Shyr, Y., and Moore, J.H. (2005). Proteomic-based prognosis of brain tumor patients using direct-tissue matrix-assisted laser desorption ionization mass spectrometry. *Cancer Research*, 65:7674–7681.

Tsai, C. T., Lai, L. P., Lin, J. L., Chiang, F. T., Hwang, J. J., Ritchie, M. D., Moore, J. H., Hsu, K. L., Tseng, C. D., Liau, C. S., and Tseng, Y. Z. (2004). Renin-angiotensin system gene polymorphisms and atrial fibrillation. *Circulation*, 109:1640–6.

White, B. C., Gilbert, J. C., Reif, D. M., and Moore, J. H. (2005). A statistical comparison of grammatical evolution strategies in the domain of human genetics. *Proceedings of the IEEE Congress on Evolutionary Computing*, pages 676–682.

Yu, T., Riolo, R., and Worzel, B. (Eds.) (2006). *Genetic Programming Theory and Practice III*. Springer.

Chapter 6

TOWARDS AN INFORMATION THEORETIC FRAMEWORK FOR GENETIC PROGRAMMING

Stuart W. Card[1] and Chilukuri K. Mohan[1]
[1]*EECS Department, Syracuse University, Syracuse, NY 13244-4100 USA.*

Abstract An information–theoretic framework is presented for the development and analysis of the ensemble learning approach of genetic programming. As evolution proceeds, this approach suggests that the mutual information between the target and models should: (i) not decrease in the population; (ii) concentrate in fewer individuals; and (iii) be "distilled" from the inputs, eliminating excess entropy. Normalized information theoretic indices are developed to measure fitness and diversity of ensembles, without *a priori* knowledge of how the multiple constituent models might be composed into a single model. With the use of these indices for reproductive and survival selection, building blocks are less likely to be lost and more likely to be recombined. Price's Theorem is generalized to pair selection and rewritten to show key factors related to heritability and evolvability. Heritability of information should be stronger than that of error, improving evolvability. We support these arguments with simulations using a logic function benchmark and a time series application. For a chaotic time series prediction problem, for instance, the proposed approach avoids familiar difficulties (premature convergence, deception, poor scaling, and early loss of needed building blocks) with standard GP symbolic regression systems; information-based fitness functions showed strong intergenerational correlations as required by Price's Theorem.

Keywords: genetic programming, information theory, ensemble models, building blocks, diversity, fitness, entropy, mutual information, redundancy, synergy, sufficiency, necessity, heritability, evolvability, group selection, Price's Theorem

1. Introduction

The essence of evolutionary learning consists of information flows between the natural or artificial environment and the entities differentially surviving and reproducing therein. Information theory enables rigorous definition of metrics for quantifying these information flows, as well as other notions such as epistasis. It has been applied to machine learning (Principe et al., 2000), but rarely to evolutionary computation – a task that this chapter addresses.

Our information-theoretic approach was initially motivated by practical difficulties encountered in model construction by Genetic Programming (GP). GP symbolic regression applied to stochastic chaotic time series prediction should perform system identification with minimal preconceptions as to model form, producing not only predictions, but also parsimonious meaningful descriptions, capturing local and global characteristics of stochastic attractors, yielding insight into the hidden dynamics. However, we encountered difficulties such as premature convergence (early loss of diversity) when attempting to learn the defining equation for the simplest known chaotic flow (Sprott, 1997; Sprott, 2000):

$$\dddot{x} = -2.017\ddot{x} + \dot{x}^2 - x \tag{6.1}$$

Due to deception and poor scaling, relatively high fitness individuals in early generations did not contain the building blocks needed to evolve correct solutions in later generations.

A search for the underlying causes of these difficulties motivated the exploration of an information theoretic perspective, and a reformulation of the goals and structure of the evolutionary algorithm. For instance, commonly used fitness measures, such as root mean squared error (RMSE), often fail to reward individuals whose presence in the population is necessary to explain substantial portions of the data variance. Fitness indicators must be developed that reward individuals for their potential incremental contributions to solution of the overall problem, perhaps by explicitly identifying building blocks suitable for recombination. Populations at any stage of evolution can then be considered as precursors to better populations, or as ensemble models.

This motivates use of Mutual Information (MI) as a fitness indicator, conveying how much one item (a candidate model in the GP population) or ensemble reveals about another (the target data set). MI is invariant under transformations that can otherwise conceal an individual's potential contribution to solutions (M. Keijzer, private correspondence). In particular, MI is invariant to invertible transformations and degraded at most by a calculable small amount for most reasonable non-invertible transformations, so it should be effective at identifying high fitness simple forms in early generations (Koza et al., 2005) that are likely to be good building blocks for later generations (Daida, 2005).

Information theory is also applicable in other aspects of evolutionary algorithms. For instance, extant diversity indicators are often arbitrary, may reflect diversity irrelevant to solving the problem, and are incommensurate with fitness measures. By contrast, information theoretic functionals can be developed that are objective, justifiable, general, computable, and commensurate measures of fitness and diversity. They can quantitatively evaluate ensemble models, without requiring knowledge of how the constituent models might be composed into a single more complex model. They can be used to guide terminal set, reproductive and survival selection. We argue that they should be heritable across mutation and recombination, and on problems studied thus far we have found them to remain so as parental fitness increases, thereby maintaining evolvability.

Gain or loss of information should be explicitly considered in evolutionary algorithm theory, operator and representation design practice, and fixing or adapting population size to maintain diversity. There are three fundamental desiderata – as evolution proceeds, the mutual information between the target and models should:

(i) not decrease in the population;

(ii) concentrate in fewer individuals; and

(iii) be 'distilled' from the inputs, leaving behind their excess entropies.

Applying Shannon's theory to evolutionary learning, we identify four "channels" to be considered: (1) the hidden process (between system input and output variables); (2) the observation process (between those variables and their measurements); (3) the selection and genetic variation processes (between parents and offspring); and (4) the evolutionary learning process (between environment and genome).

Past work on application of information theory to evolutionary computation includes: selection of terminal sets for GP (Deignan et al., 2002); measurement of fitness (Aguirre and Coello, 2004) and diversity (Liu et al., 2001) and of both commensurately (Card and Mohan, 2005). Our work attempts to develop a more comprehensive framework: using information theoretic metrics for all phases of selection; and studying evolvability and heritability of information across genetic variation.

In Section 2, we review information theoretic preliminaries. In Section 3 we normalize entropy and mutual information to define various indices and use these in turn to define commensurate measures of diversity and fitness. In Section 4 we apply these indices to selection in GP, generalize Price's Theorem (Altenberg, 1994) to pair selection and define a measure of effective fitness. In Section 5 we argue that a prerequisite for evolvability is heritability and that both these qualities should be improved by using information based rather than error based fitness. In Section 6 we support these arguments with experimental results. Section 7 summarizes our findings and outlines directions for future work.

2. Preliminaries

This section presents the basic definitions of mutual information, redundancy and synergy.

In our notation, boldface indicates vector-valued inputs, outputs, functions, etc. We consider functions $\mathbf{f}_j$ to be model genotypes and output data sets $\mathbf{Z}_j$ (given the input data set $\mathbf{X}$) to be model phenotypes; the genotype–phenotype correspondence is in general many-to-one. Applying the unknown target function $\mathbf{f}$ to the ith data point yields $\mathbf{y_i} = \mathbf{f}(\mathbf{x_i})$, whereas applying the jth model function yields $\mathbf{z_{i,j}} = \mathbf{f_j}(\mathbf{x_i})$. The input, target and model output data vectors comprise sample distributions: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_n\}$; $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2 \ldots \mathbf{y}_n\}$; and $\forall j \in [1...m] : \mathbf{Z}_j = \{\mathbf{z}_{1,j}, \mathbf{z}_{2,j} \ldots \mathbf{z}_{n,j}\}$.

Mutual Information

For discrete distributions with d distinct values, Shannon's entropy is

$$H(\mathbf{Y}) = \sum_{k=1}^{d} p(\mathbf{y}_k) log(p(\mathbf{y}_k)) \tag{6.2}$$

The *Mutual Information* (MI) between output data sets of the target function and candidate model j is

$$I(\mathbf{Y}; \mathbf{Z}_j) = H(\mathbf{Y}) + H(\mathbf{Z}_j) - H(\mathbf{Y}, \mathbf{Z}_j) \tag{6.3}$$

where $H(\mathbf{Y}, \mathbf{Z}_j)$ is the *joint entropy*, i.e., the entropy of the joint distribution of Y and Z. MI quantifies explanation of variance in the target output data set by variance in the model output data set (and vice versa).

Redundancy & Synergy

We can quantify the information about $\mathbf{Y}$ provided jointly by outputs $\mathbf{Z}_j$ and $\mathbf{Z}_k$ (of two candidate models $\mathbf{f}_j$ and $\mathbf{f}_k$) by their *incremental mutual information* or *marginal redundancy*:

$$I(\mathbf{Y}; \mathbf{Z}_j, \mathbf{Z}_k) = H(\mathbf{Y}) + H(\mathbf{Z}_j, \mathbf{Z}_k) - H(\mathbf{Y}, \mathbf{Z}_j, \mathbf{Z}_k) \tag{6.4}$$

Information conveyed jointly (as above) may exceed or fall short of the sum of that conveyed severally by its constituents. In some cases, the mutual information between the target and each of the models may be zero, but the two models may jointly explain the target completely due to their synergy.

Example: The simplest example of pure synergy flips two fair coins x_1 and x_2 and XORs them to produce $y = f(x_1, x_2) = x_1 \otimes x_2$. Two simplistic candidate models are observations of each of the coins:

$$\begin{aligned} z_j &= f_j(x_1, x_2) &= x_1 \\ z_k &= f_k(x_1, x_2) &= x_2 \end{aligned}$$

The entropies of the target and the models are $H(\mathbf{Y}) = H(\mathbf{Z}_j) = H(\mathbf{Z}_k) = 1$. The pairwise MI values are $I(\mathbf{Y};\mathbf{Z}_j) = I(\mathbf{Y};\mathbf{Z}_k) = I(\mathbf{Z}_j;\mathbf{Z}_k) = 0$. Yet, the synergy of the two models enables them jointly to completely explain the target: $I(\mathbf{Y};\mathbf{Z}_j, Z_k) = 1$.

Building blocks cannot be identified if their fitness only becomes apparent when they are combined. However, when synergy between individuals in a population can be detected, it can be exploited to guide evolutionary steps. Hence we adopt the *synergy-redundancy* measure (Chechik, 2003):

$$S(\mathbf{Y};\mathbf{Z}_j,\mathbf{Z}_k) = I(\mathbf{Y};\mathbf{Z}_j,\mathbf{Z}_k) - (I(\mathbf{Y};\mathbf{Z}_j) + I(\mathbf{Y};\mathbf{Z}_k)) \tag{6.5}$$

$H()$ and $I()$ must be non-negative, but $S()$ can be positive (synergy exceeding any redundancy), negative (redundancy exceeding any synergy) or zero (possibly (i) independence, or (ii) synergy exactly balancing redundancy).

3. New Evaluation Measures for GP

This section develops new metrics to evaluate (sets of) individuals in an evolving population: sufficiency, necessity, and other normalizations of MI.

Sufficiency

Assuming that the outputs are deterministic functions of the inputs, that we wish to fully model the hidden process, and that observations of both inputs and outputs are noise free, the *sufficiency* of a model is

$$I(\mathbf{Y};\mathbf{Z}_j)/H(\mathbf{Y})$$

Sufficiency ranges from 0 to 1, and describes the extent to which a model captures the information in the target. To account for non-deterministic processes, insufficient inputs, approximate models, noise-corrupted observations, and ensemble models, we define an ensemble model $\mathbf{f}^m$ to be *ϵ-sufficient iff*:

$$I(\mathbf{Y};\mathbf{Z}^m)/I(\mathbf{Y};\mathbf{X}) \geq 1-\epsilon. \tag{6.6}$$

Necessity

The *necessity* of an individual model is the fraction of its entropy that contributes to its explanation of the target. We define an ensemble model $\mathbf{f}^m$ to be *ϵ-necessary iff*:

$$I(\mathbf{Y};\mathbf{Z}^m)/H(\mathbf{Z}^m) \geq 1-\epsilon \tag{6.7}$$

Quality

Residual entropy of the target, not explained by the model, $H(\mathbf{Y}|\mathbf{Z}^m)$, is a shortfall in model sufficiency. Excess entropy of the model, which does not

contribute to its explanation of the target, $H(\mathbf{Z}^m|\mathbf{Y})$, is a shortfall in model necessity. These conditional entropies sum as the information theoretic measure of the error. If the model output requires discrete permutation (re-coding) or continuous transformation (translation, scaling, etc.) to align with the target coordinates, there will be additional error (not measured by the foregoing) between the untransformed model output and the target.

The objectives of sufficiency and necessity may be considered separately, or combined into a scalar measure of overall information theoretic solution quality based on *normalized mutual information* (NMI):

$$_NI(\mathbf{Y};\mathbf{Z}^m) = I(\mathbf{Y};\mathbf{Z}^m)/H(\mathbf{Y},\mathbf{Z}^m) \tag{6.8}$$

This indicator fairly penalizes both sources of error: a model can achieve a perfect score of one only by exactly explaining target variance, eliminating both residual target entropy and excess model entropy. A model that fully explains target entropy, but with a similar amount of excess model entropy, scores 0.5; a model that has no excess entropy, but explains only half the target entropy, also scores 0.5; and a model that explains half the target entropy, and has a similar amount of excess model entropy, scores 0.333. The NMI between the target and a null model, which produces uniformly distributed random numbers, can serve as a baseline against which to compare candidate models; this test is necessitated by spurious mutual information that can appear due to the sparseness of data sets of high dimensionality. Still assuming deterministic models, this can be generalized for insufficient inputs as

$$_NI_{\mathbf{X}}(\mathbf{Y};\mathbf{Z}^m) = I(\mathbf{Y};\mathbf{Z}^m)/(I(\mathbf{X};\mathbf{Y}) + I(\mathbf{X};\mathbf{Z}^m) - I(\mathbf{Y};\mathbf{Z}^m)) \tag{6.9}$$

Similarity, Distance, & Equivalence

We read the inequality $_NI_{\mathbf{X}}(\mathbf{Y};\mathbf{Z}^m) \geq 1-\epsilon$ as "given environmental inputs $\mathbf{X}$, target outputs $\mathbf{Y}$ and model outputs $\mathbf{Z}^m$, model $\mathbf{f}^m$ is ϵ-equivalent to target $\mathbf{f}$". Similarly, NMI can also be used to detect information theoretic ϵ-equivalence of multiple models:

$$_NI_{\mathbf{X}}(\mathbf{Z}_j;\mathbf{Z}_k) \geq 1-\epsilon$$

Alternatively, a binary (strict) equivalence relation can be defined as:

$$\psi_{\mathbf{X}}(\mathbf{f}_j,\mathbf{f}_k) = \lfloor {}_NI_{\mathbf{X}}(\mathbf{Z}_j;\mathbf{Z}_k) \rfloor \tag{6.10}$$

$_NI_{\mathbf{X}}(\mathbf{Z}_j;\mathbf{Z}_k)$ is a *similarity metric*, the complement of which is a true (triangle inequality satisfying) normalized *distance metric*:

$$_Nd_{\mathbf{X}}(\mathbf{Z}_j,\mathbf{Z}_k) = 1 - {}_NI_{\mathbf{X}}(\mathbf{Z}_j;\mathbf{Z}_k) \tag{6.11}$$

Diversity

Several useful types of diversity can be distinguished and objectively measured.

- *Genotypic representation diversity* may be measured by a normalized information distance based upon Kolmogorov complexity (K) (Li et al., 2003):

$$_N d_K(\mathbf{f}_j, \mathbf{f}_k) = (K(\mathbf{f}_j|\mathbf{f}_k) + K(\mathbf{f}_k|\mathbf{f}_j))/K(\mathbf{f}_j\mathbf{f}_k) \tag{6.12}$$

- *Phenotypic representation diversity* may be measured with distances appropriate to the domain, such as normalized Hamming distance for logic problems. However, such distances are not ideal for all evolutionary purposes, since different phenotypes may encode equivalent information. For example, x and $\bar{x}$ are different genotypes, and also express as different phenotypes; however, they encode the same information.

- *Model total information diversity* abstracts away from encoding details, and is measured by a normalized information distance based upon Shannon entropy:

$$\begin{aligned} _N d(\mathbf{Z}_j, \mathbf{Z}_k) &= 1 - _N I(\mathbf{Z}_j; \mathbf{Z}_k) && (6.13)\\ &= (H(\mathbf{Z}_j|\mathbf{Z}_k) + H(\mathbf{Z}_k|\mathbf{Z}_j))/H(\mathbf{Z}_j, \mathbf{Z}_k) && (6.14) \end{aligned}$$

- *Target relevant information diversity* refers to the information conveyed by the models about the target. It is indicated by the synergy-redundancy measure defined in Equation (6.5), extended to populations and normalized to yield an index, as follows:

$$_N S(\mathbf{Y}; \mathbf{Z}^m) = (I(\mathbf{Y}; \mathbf{Z}^m) - \sum_{j=1}^{m} I(\mathbf{Y}; \mathbf{Z}_j))/I(\mathbf{Y}; \mathbf{Z}^m) \tag{6.15}$$

  This ranges in $[-(m-1), +1]$. A value of 0 indicates that the models in the population convey independent information about the target *on average*. A value of +1 indicates that all the information conveyed jointly by the population is the result of synergy among two or more of those models; no information about the target is conveyed by any individual model. A value of $-(m-1)$ indicates that all m models in the population are fully redundant; each conveys exactly the same information about the target.

Fitness

In addition to considering intra-population competition as usual, we propose also to consider the contribution that an individual makes to the quality of the

entire population. Each individual's contribution to the population's mutual information with the target must be non-negative, but an individual may degrade population quality by contributing excess entropy, or consume computational resources while providing negligible non-redundant information about the target. The net effect of individual j on the overall solution quality of the entire population regarded as an ensemble model can be evaluated as:

$$ {}_N I_{\mathbf{X}}(\mathbf{Y}; \mathbf{Z}^m) - {}_N I_{\mathbf{X}}(\mathbf{Y}; \mathbf{Z}^m \backslash \mathbf{Z}_j) $$

However, net degradation of this measure may be required to achieve modeling goals, e.g., it may be necessary to preserve an individual that contributes only a single bit to the mutual information between the population and the target, yet adds two bits to the population's entropy. Hence, a better approach is to consider separately both the individual's incremental sufficiency and necessity:

$$ (I(\mathbf{Y}; \mathbf{Z}^m) - I(\mathbf{Y}; \mathbf{Z}^m \backslash \mathbf{Z}_j)) \quad / H(\mathbf{Y}) \tag{6.16} $$

$$ (I(\mathbf{Y}; \mathbf{Z}^m) - I(\mathbf{Y}; \mathbf{Z}^m \backslash \mathbf{Z}_j)) \quad / (H(\mathbf{Z}^m) - H(\mathbf{Z}^m \backslash \mathbf{Z}_j)) \tag{6.17} $$

Normalization must be carried out differently: target entropy does not change, but an individual that makes a positive contribution to mutual information should have positive incremental necessity (even if that contribution is less than its addition to joint entropy) so the latter denominator must be the *incremental* joint entropy of the model set. The incremental necessity can exceed 1 if there is synergy between the individual and others already present in the ensemble to which it is added. An individual should not degrade necessity unless it thereby improves sufficiency. If these are both non-positive in the context of the entire population, the individual should be considered a candidate for deletion.

4. Selection

These new evaluation measures are to be used in applying selection pressure in the evolutionary algorithm, in three different ways: terminal set selection, expression selection, and ensemble selection, discussed in this section.

Terminal Set Selection

GP terminal set selection may be nontrivial: selection is from the power set of the observables, which grows exponentially. Even where the GP literature refers to entropy and mutual information, these typically fail to address synergy among inputs. For instance, two inputs that are multiplied (in the hidden system to be identified) each may have negligible mutual information with the system outputs, yet their product may be the dominant factor in those outputs.

Thus, it is essential that inputs be considered jointly. A computationally expensive approach is to construct all the possible input ensembles, increasing ensemble cardinality until some acceptably large fraction of the target

entropy is captured, as measured by the input ensemble sufficiency index $I(\mathbf{Y};\mathbf{X}^m)/H(\mathbf{Y})$.

Expression Selection

The advantages of mutual information for expression selection are: its invariance under invertible transformations (in discrete domains, permutations, and in continuous domains, homeomorphisms); its small degradation by non-invertible transformations with few pre-images; and its detection of *all* dependencies. For example, consider the result of evaluating candidate model $f_j(x) = x^3$ against target $f(x) = x^4$ using RMSE, correlation and mutual information. RMSE of this model will be large; indeed it will be larger than that of many alternative models whose symbolic (genotypic) representations are much further in edit distance from the target than this one. If the input has zero mean, correlation of this model with the target will be zero; again, less than that of other models (which must traverse much longer evolutionary trajectories to arrive at the target equation). NMI, on the other hand, will be high: this model is fully sufficient and only one bit (the sign) shy of fully necessary.

Ensemble Selection

The unique advantage of information theoretic evaluation measures is in objectively evaluating the quality of an ensemble model without relying on a heuristic for combining the constituent models. This is particularly evident when the constituent models exhibit synergy. Consider again the XORed coin flips of Section 2.0. Individual expressions $f_j(x_1, x_2) = x_1$ and $f_k(x_1, x_2) = x_2$ show no merit in terms of RMSE, correlation or mutual information; yet the ensemble model $\{f_j, f_k\}$ has a sufficiency of 1.0 and a necessity of 0.5 for an overall quality of 0.5. This ensemble aggregates the expressions that will model the target precisely, if properly composed.

Pair Selection

A high positive value of ${}_N S(\mathbf{Y};\mathbf{Z}_j,\mathbf{Z}_k)$ indicates that the individuals (models f_j, f_k) have strong reproductive upside potential with respect to sufficiency: their synergistic elements may combine to yield higher individual sufficiency of a child than of either parent. On the other hand, much lower values of that index indicate low reproductive downside potential with respect to sufficiency: if genotypic variation is respectful of phenotypic entropies,[1] then the offspring will inherit the target-related MI that is conveyed redundantly by the parents.

[1]While this will not hold strictly and universally, it should hold approximately, most of the time, under reasonable genetic variation operators; see Section 5.

High synergy indicates that a pair is strong relative to its own members. However, those individual members might have been quite weak; high synergy does not indicate that the pair is stronger than another pair. By contrast, ${}_N I_{\mathbf{X}}(\mathbf{Y}; \mathbf{Z}_j, \mathbf{Z}_k)$ explicitly indicates fitness and implicitly incorporates relevant diversity. The raw mutual information of a child model with the target cannot exceed that of its parents jointly with the target;[2] but the NMI of the child with the target can exceed that of its parents, if the child has less excess entropy than the aggregate of its parents (as it often will).

Price's Theorem

Price's Theorem has been applied to Holland's canonical model of genetic algorithms, assuming individual selection for reproduction and random mating (Altenberg, 1994). This section presents an extension of this result to non-random mating by using information theoretic evaluations of potential parental pairings to drive reproductive sampling rate. We begin with Slatkin's transmission-selection recursion:

$$p'(f_j) = \sum_{k,l} (T(f_j \leftarrow (f_k, f_l)) \frac{w(f_k)}{\overline{w}} p(f_k) \frac{w(f_l)}{\overline{w}} p(f_l),$$

which leads to a formulation of Price's Equation:

$$\Delta F = Cov(\phi(f_k, f_l), \frac{w(f_k) w(f_l)}{\overline{w}^2}),$$

where: $F(f_j)$ is the measurement function for the property of interest as exhibited by genotype f_j; $T(f_j \leftarrow (f_k, f_l))$ is the transmission function giving the probability that genotype f_j is produced by parental genotypes f_k and f_l; $p(f_j)$ is the frequency of genotype f_j in the population at the current generation; $p'(f_j)$ is that frequency at the next generation; $w(f_j)$ is the reproductive sampling rate of genotype f_j; and $\phi(f_k, f_l)$ is the expectation of $F()$ in the offspring of parents f_k and f_l.

If we choose as our measurement function the overall information theoretic model quality index (fitness in the engineering sense) defined in Equation (6.9) applied to individuals, and set our individual reproductive sampling rate (fitness in the population biology sense) equal to that measurement function, then the population average of our solution quality index for individuals will increase. However, this may be not optimal for ensemble modeling, nor for recombination to produce improved types. Changing our assumptions (from individual reproductive selection and random mating) to non-random pair selection, we

[2]However, see the caveat in Subsection 5.0 regarding weak applicability of the Data Processing Inequality.

choose the same measurement function but apply it to pairs and set our pair sampling rate equal to that measurement function:

$$w_2(f_k, f_l) = F_2(f_k, f_l) = \ {}_N I_{\mathbf{X}}(\mathbf{Y}; \mathbf{Z}_k, \mathbf{Z}_l)$$

We accordingly slightly revise Slatkin's recursion:

$$p'(f_j) = \sum_{k,l} (T(f_j \leftarrow (f_k, f_l)) \frac{w_2(f_k, f_l)}{\overline{w_2}} p(f_k, f_l) \tag{6.18}$$

The pair frequency (joint density) factors into the individual parental frequencies, but the pair sampling rate does not. Our change to Slatkin's recursion then follows Price's proof to the analogous conclusion:

$$\Delta F = Cov(\phi(f_k, f_l), \frac{w_2(f_k, f_l)}{\overline{w_2}}) \tag{6.19}$$

Recalling the definition of $\phi(f_k, f_l)$, we define our predictive estimator:

$$\hat{\phi}(f_k, f_l) = \frac{F_2(f_k, f_l)}{\overline{F_2}}$$

yielding:

$$\Delta F_2 = Cov(\phi(f_k, f_l), \hat{\phi}(f_k, f_l)). \tag{6.20}$$

We now address the question "How strong is the covariance?", or equivalently, ''How accurate is our estimator of offspring solution quality?" Defining:

$$C(f_k, f_l) = \phi(f_k, f_l) - \frac{F(f_k) + F(f_l)}{2}$$

and rewriting to show relative change, we can show the key factors affecting heritability and evolvability, isolating the influences of selection, genetic variation, and their interaction, in the three terms of the summation:

$$\begin{aligned} \frac{\Delta F_2}{\overline{F_2}} = & \frac{\sigma_{F_2(f_k,f_l)}}{\mu_{F_2(f_k,f_l)}} \frac{\sigma_{w_2(f_k,f_l)}}{\mu_{w_2(f_k,f_l)}} \rho_{F_2(f_k,f_l), w_2(f_k,f_l)} && \textbf{\textit{[selection]}} \\ & + \frac{\overline{C}}{\mu_{F_2(f_k,f_l)}} && \textbf{\textit{[variation]}} \\ & + \frac{\sigma_{w_2(f_k,f_l)}}{\mu_{w_2(f_k,f_l)}} \frac{\sigma_{C(f_k,f_l)}}{\mu_{F_2(f_k,f_l)}} \rho_{w_2(f_k,f_l), C(f_k,f_l)} && \textbf{\textit{[interaction]}} \end{aligned} \tag{6.21}$$

Effective Fitness

If we redefine individual fitness as the expectation of the fitness of all pairs of which the individual might be a member:

$$w(f_k) = \sum_l w_2(f_k, f_l)p(f_l) \tag{6.22}$$

we recover the original short form of Price's Equation:

$$\Delta F = Cov(\phi(f_k, f_l), \frac{w(f_k)w(f_l)}{\overline{w}^2})$$

with a new interpretation. The fitness of an individual is not independent of the population of which it is a member; the individual's effective fitness is low if the rest of the population does not contain individuals that, when recombined with the given individual, are likely to produce offspring with comparably high or higher solution quality. If there are other individuals with which the given individual could be recombined favorably, but the frequencies of the two genotypes of interest are both low, then the likelihood of the favorable recombination taking place under random mating is very low. Another argument in favor of joint fitness based non-random mating (pair selection) is that it expends the few mating opportunities of rare genotypes on those partners most likely to yield fit offspring.

5. Variation

We now consider the design of mutation and recombination operators to improve *evolvability* (the propensity of a population under selection and genetic variation to improve in fitness). As parental fitness moves further to the right of the expected fitness of a randomly generated individual, their offspring fitness distribution becomes increasingly biased to the left of the parental mean. The factors identified in Equation (6.21) are critical. The fitness variance introduced by the genetic operators must be high enough so that there is significant chance of producing offspring with fitness better than the parents, but low enough that parent-offspring fitness correlation is strong. The correlation between parental fitness and the expected improvement in offspring fitness relative to those parents will be negative, but should be minimized through careful design of representations and operators. Evolvability thus requires *heritability* (intergenerational correlation) that persists as parental fitness increases.

For heritability, information-based fitness is preferable to error-based fitness for several reasons. First, mutual information is invariant with respect to invertible transformations and degraded only slightly by reasonable non-invertible transformations. Second, information measures can objectively evaluate the quality of ensemble models as such, without *a priori* knowledge of how the

constituent models might be composed into a single model. Third, measures evaluating prospective parents jointly should enable selection to maximize the upper tail of the expected offspring distribution.

Mutation

For heritability to be strong, mutation steps in genotypic representation space must correspond to small steps in phenotypic fitness space. Simple mutations will traverse only a small edit distance, which is likely to correspond to an invertible transformation, or a non-invertible transformation where only a small number of domain pre-images map to an image in the range. Hence NMI is heritable in most simple edits, whereas RMSE and correlation are not. For instance, changing additive or multiplicative constants changes RMSE, but does not affect correlation and NMI. Changing from x^{2n} to x^{2n+1} can change correlation considerably, but causes loss of only one bit of mutual information. Low RMSE implies high correlation, which implies high NMI, but the reverse implications do not generally hold. Thus *neutral mutations* will be observed more often under NMI based fitness.

Potential for useful mutation may be indicated by an individual's low necessity index. Mutation can improve necessity by discarding entropy that does not contribute to modeling the target. For example, in the symbolic regression problem of Section 6.0, a 3-expression ensemble of raw inputs was required to achieve sufficiency; squaring one expression to discard one bit of unnecessary information (its sign) improved its necessity. This reduced the expression's individual sufficiency, but not ensemble sufficiency, thereby improving overall solution quality.

Recombination

If recombination degrades either sufficiency or necessity while not improving the other, there appears to be no advantage to adding the offspring to the population. Recombinations that trade sufficiency for necessity or *vice versa* cannot be evaluated so easily, and neutral recombinations, like neutral mutations, may be needed along the evolutionary pathway. One multi-objective heuristic is to: (1) maintain population sufficiency, (2) try to achieve individual necessity, and (3) strive for both in ensembles.

Genetic representations and operators may be extended to process ensembles as individuals. This introduces a requirement for recombination not only of expressions but also of ensembles. (Radcliffe, 1993) describes set recombination operators (R3, RTR and RAR) with the desirable properties of respect, strict transmission and proper assortment. These operators cannot be readily applied to GP, and we propose an alternative: apply Radcliffe's R3/RTR operator, and if it generates offspring that violate the size constraint, fix them by deleting the

fewest members needed to return to the constrained region. When respect is observed in the genotypes of ensembles, it generically will be reflected in their phenotypic entropies as follows:

$$\begin{aligned} I(\mathbf{Z}'_k;\mathbf{Z}'_l) &\geq I(\mathbf{Z}_k;\mathbf{Z}_l) \\ I(\mathbf{Y};\mathbf{Z}'_k) &\geq -S(\mathbf{Y};\mathbf{Z}_k,\mathbf{Z}_l) \\ I(\mathbf{Y};\mathbf{Z}'_l) &\geq -S(\mathbf{Y};\mathbf{Z}_k,\mathbf{Z}_l) \end{aligned}$$

The following relationships should hold for generic recombinations:

$$\begin{aligned} H(\mathbf{Z}'_k,\mathbf{Z}'_l) &\leq H(\mathbf{Z}_k,\mathbf{Z}_l) \\ I(\mathbf{Y};\mathbf{Z}'_k,\mathbf{Z}'_l) &\leq I(\mathbf{Y};\mathbf{Z}_k,\mathbf{Z}_l) \end{aligned}$$

These relationships, based upon the Data Processing Inequality, do not hold in all cases because variation of the genotypes is not the same thing as processing by the phenotypes of the input information, so recombination may increase the mutual information between a target and offspring (vs. parental) models. Expression recombination using typical GP crossover operators will often respect entropy; set recombination using R3/RTR-like operators will always do so, except when it must be violated to permit proper assortment under a size constraint. Recombinations that lead to aggregation of expressions into larger ensembles can improve sufficiency; recombinations that compose expressions or delete them from ensembles can improve necessity.

Equation (6.10) is a binary function that indicates strict information theoretic equivalence of its arguments. To apply Radcliffe's theory, we need instead a family of such functions, each of which indicates equivalence with respect to a particular portion of the information of interest. Unfortunately the partitioning of information is arbitrary, and NMI with each input and output variable spans the space with non-orthogonal coordinate axes. Consider the XORed coin flip example from Section 2.0: if the information space coordinates of an ensemble are ${}_N I(\mathbf{X}_1;\mathbf{Z}^m)$, ${}_N I(\mathbf{X}_2;\mathbf{Z}^m)$ and ${}_N I(\mathbf{Y};\mathbf{Z}^m)$, then the points (0,0,0), (1,0,0), (0,1,0), (0,0,1) and (.5,.5,.5) are all feasible, but (1,1,0), (1,0,1) and (0,1,1), among others, are not.

6. Findings

This section presents preliminary results from GP experiments using an information-theoretic approach, with a Logic Function benchmark and a Time Series prediction problem.

Logic Function Benchmark

GP can be applied to synthesize logic functions using 2-input multiplexers (2-MUXes); (Aguirre and Coello, 2004) report evolutionary synthesis of 4-input

even parity, using MI in the fitness function. A basic stage can be defined, with 2 inputs and 2 outputs; one stage per variable is required, arranged as a ladder with criss-crossed connections. This requires reuse of stage outputs, which cannot be represented in a GP tree without a mechanism such as Automatically Defined Functions, so a simple GP representation will have redundant MUXes in the earlier stages.

We have implemented a generational GP supporting uniform, truncation and rank proportional selection for reproduction and those same choices for survival. Fitness is multi-objective and the domination check is masked to regard any subset of the fitness vector: sufficiency, necessity, overall information theoretic solution quality, inverse ensemble cardinality, correlation and inverse total ensemble size. The domination mask is dynamically set based on population fitness statistics to enable incremental evolution; for instance, to consider sufficiency only until the median individual is fully sufficient, then consider necessity also. Expression recombination is performed by traditional GP subtree swapping. Expression mutation randomly makes one of the minimal moves in tree edit distance. Individuals are ensemble models represented as multisets with parameterized constraints on the maximum number of distinct expressions and the maximum number of copies of each expression. Ensemble recombination is R3/RTR modified as described in Section 5.0. Ensemble mutation consists of randomly performing one of the following operations: composing two constituent expressions into one; deleting one expression; duplicating one expression; inserting one random primitive expression; mutating one expression; or recombining two expressions.

In experiments thus far, information theoretic GP has rapidly aggregated the raw inputs to produce fully sufficient ensembles, then more slowly composed the constituent expressions to produce ensemble models of improving necessity. The system has surprised its designers by pursuing evolutionary trajectories that were not anticipated, but which, upon inspection, proved to be following the sequentially superior building block route as prescribed by the multi-objective information theoretic fitness function. For example, the system produced ensembles that expressed as $\{x_1 \otimes x_2, x_3 \otimes x_4\}$ (100% sufficient and 50% necessary) and then appeared to stall. But then it replicated one of the constituent expressions and grafted the replica onto the other constituent expression in place of what had previously been a constant terminal input, thereby improving necessity and escaping the fitness plateau. Three more steps exactly like that lead to an optimal solution. Gene duplication (expression duplication within an ensemble) has been found to be important, both to protect against gene deletion or disruption, and to prepare for subtree composition. Recoding also

has been found to be important, both to provide composable representations and to transform model outputs into target coordinates.[3]

Time Series Prediction

We analyze the effects of our indicators on GP regression of Sprott's equation (6.1) that motivated their development. With a terminal set of 3 variables $\{\ddot{x}, \dot{x}, x\}$ and a non-terminal set of 4 basic arithmetic operators $\{+, -, *, /\}$, discarding duplicate expressions and those that reduce to constants (with neither variance for correlation nor entropy for mutual information), we form complete populations of 3 (one term), 27 (up to two term) and 296 (up to three term) distinct expressions. We apply them all to inputs $\mathbf{X}$ and compute their RMSE, correlation and mutual information with the target $\mathbf{Y}$. To enable consistent usage, we normalize mutual information as above, and compute the reciprocal of (1+RMSE). Correlation is inherently normalized.

Initial population: All three terminals are included. Normalized RMSE has significantly different values for each terminal, but all of the same order of magnitude. Correlation favors the first derivative over the others by several orders of magnitude. NMI assigns similar fitness to each terminal. The correct solution requires all three terminals.

Second generation: The population contains all expressions of up to two terms. The term ranked first by NRMSE was not a part of the correct formula, the term ranked first by correlation was a term of a formula yielding the same behavior (to within a constant) as the correct formula, and NMI ranked a term of the correct formula first.

Third generation: The population contains expressions of up to three terms. NMI, which is a generalization of correlation, approximately agreed on rankings as often with NRMSE as it did with correlation. Forms close to the correct formula were identified slightly more often by NMI than by NRMSE, both of which significantly outperformed correlation.

Correlation coefficients between 2nd generation (27 members) parental indicators and 3rd generation offspring (351 broods of 6 members each) fitnesses were calculated. Correlation of mean fitness of the brood, with the fitness of either parent, was 0.6; with the product of the fitnesses of the parents, 0.9. Correlation of mean fitness of the brood, with the joint NMI between the parents and the target, was 0.8 in the general population of 351 potential pairings. After dropping low fitness individuals from the breeding population, reducing it to

[3]Detailed results will be presented in the first author's forthcoming Ph.d. dissertation and made available via the Internet.

210 potential pairings, it still held at 0.3. Correlation of improvement of the best of the brood versus the better of the parents, with the normalized synergy of the parents, was 0.2 in that breeding population. After also dropping pairings with low joint NMI against the target, reducing the breeding population to 140 potential pairings, it was 0.4. Dropping pairings with low synergy, of the 70 pairs remaining, 45 produced broods whose best members had fitness higher than the better of their parents. Of these, 13 had offspring better than any individual in the breeding population. Of these, 10 included a parent that was an optimal subexpression. Of these, 4 pairs were composed of two optimal subexpressions. Of these, 2 sets of parents correctly paired two optimal subexpressions.

The parent/offspring fitness correlations required by Price's Theorem were strong, not only between the first and second generations, but also between the penultimate and ultimate generations, so on this problem, NMI maintained heritability, thereby improving evolvability. NMI also correctly performed terminal set selection, keeping all three inputs. This contrasts dramatically with correlation and the results of applying standard GP to this problem. These findings are strong evidence of effectiveness of information theoretic fitness and diversity indicators for evolutionary algorithms.

7. Concluding Remarks

Information theory has numerous important applications to evolutionary learning. It enables explicit treatment of information flows between the environment and the genome, and the gain or loss of information due to evolutionary steps. It enables detection and quantification of *all* dependencies of outputs on inputs, and of epistasis between inputs. It is useful in all phases of evolutionary computation, including selection of terminal and non-terminal sets that convey information about the target, reproductive selection that maximizes evolvability, survival selection that preserves essential diversity, and objective evaluation of ensemble models at end-of-run. Information theoretic indices are easily defined and provide objective, justifiable, heritable, general, computable, commensurate indicators of fitness and diversity, which are undeceived by many transformations. They can measure ensemble quality without requiring knowledge of how to compose its constituent simple models into a single complex model; this can be used to guide group selection and non-random mating.

We have identified information theory as a powerful tool for analyzing evolutionary learning and have illustrated a few of its applications. We have developed information theoretic indicators of solution quality that can be applied not only to individuals but also to pairs, ensembles and entire populations, thereby ensuring information diversity. We have shown how this can drive reproductive sampling rate of potential parental pairings, and extended Price's Equation to

show the effects thereof. We identify heritability of mutual information as a key issue, and study the effects of recombination and mutation on information theoretic measures applied to ensembles.

Two issues arise when applying information theory to GP practice. First, synergies of multiple inputs require calculation of joint entropies, which is exponential in both memory space and processor time; we are addressing this by attempting to optimize code using sparse array techniques. Second, some practical applications concern continuous valued data, whereas Shannon's entropy is defined only for discrete distributions; generalizations such as differential entropy require vast data sets and converge slowly as bin sizes are reduced (further exacerbating computional cost); we are addressing this using copulae, order statistics and information dimension heuristics.

Future Work

Many results from the machine learning field (Principe et al., 2000) and the feature set selection area might be applied profitably (Muharram and Smith, 2004) in developing information theory-based evolutionary learning. A general proof of heritability of NMI across recombination and mutation is needed. This may lead into a re-interpretation of schema or more generally forma theory in terms of information theory; we conjecture that evolvability is favored when the Kolmogorov distance between two genotypes is closely related to the Shannon distance between the two corresponding phenotypes. More work is required to understand generalization (and its ''No Free Lunch" limits) in evolutionary learning in terms of information theory; we further conjecture that the distance between the copulas of the training and testing sets should provide an upper performance bound on generalization by a learner of input-output mappings.

Sources of problem difficulty must be analyzed in information theoretic terms, starting with the fundamental question: What are building blocks in information theoretic terms? Are there problems where the population of ensembles of cardinality n of maximum overall information theoretic solution quality are not likely, under reasonable genetic operators, to produce offspring of cardinality $n-1$, that are inferior neither to their parents nor to offspring of seemingly less fit parents? This would be deception of ensemble NMI caused presumably by non-composability. As a first step, we searched for the inverse of the usual situation: it is easy to envision equivalence classes with respect to NMI where error differs, but examples exist where a genetic operation changes NMI yet leaves error unchanged.

Acknowledgment

We thank Maarten Keijzer, Ken DeJong, David Goldberg, Kishan Mehrotra and the GPTP'07 Workshop participants for informative discussions that have helped this research.

References

Aguirre, A. and Coello, C. (2004). Mutual information-based fitness functions for evolutionary circuit synthesis. In *Proc. of the 2004 IEEE Congress on Evolutionary Computation (CEC'04)*.

Altenberg, Lee (1994). The Schema Theorem and Price's Theorem. In Whitley, L. Darrell and Vose, Michael D., editors, *Foundations of Genetic Algorithms 3*, pages 23–49, Estes Park, Colorado, USA. Morgan Kaufmann. Published 1995.

Card, Stuart W. and Mohan, Chilukuri K. (2005). Information theoretic indicators of fitness, relevant diversity & pairing potential in genetic programming. In Corne, David, Michalewicz, Zbigniew, Dorigo, Marco, Eiben, Gusz, Fogel, David, Fonseca, Carlos, Greenwood, Garrison, Chen, Tan Kay, Raidl, Guenther, Zalzala, Ali, Lucas, Simon, Paechter, Ben, Willies, Jennifier, Guervos, Juan J. Merelo, Eberbach, Eugene, McKay, Bob, Channon, Alastair, Tiwari, Ashutosh, Volkert, L. Gwenn, Ashlock, Dan, and Schoenauer, Marc, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2545–2552, Edinburgh, UK. IEEE Press.

Chechik, G. (2003). *An Information Theoretic Approach to the Study of Auditory Coding*. PhD thesis, Hebrew University.

Daida, Jason M. (2005). Towards identifying populations that increase the likelihood of success in genetic programming. In Beyer, Hans-Georg, O'Reilly, Una-May, Arnold, Dirk V., Banzhaf, Wolfgang, Blum, Christian, Bonabeau, Eric W., Cantu-Paz, Erick, Dasgupta, Dipankar, Deb, Kalyanmoy, Foster, James A., de Jong, Edwin D., Lipson, Hod, Llora, Xavier, Mancoridis, Spiros, Pelikan, Martin, Raidl, Guenther R., Soule, Terence, Tyrrell, Andy M., Watson, Jean-Paul, and Zitzler, Eckart, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1627–1634, Washington DC, USA. ACM Press.

Deignan, P., Meckl, P., and Franchek, M. (2002). The mi-rbfn: Mapping for generalization. In *Proc. of the American Control Conference (ACC'02)*.

Koza, John R., Al-Sakran, Sameer H., and Jones, Lee W. (2005). Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits. In Lohn, Jason, Gwaltney, David, Hornby, Gregory, Zebulum, Ricardo, Keymeulen, Didier, and Stoica, Adrian, editors,

Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware, pages 205–214, Washington, DC, USA. IEEE Press.

Li, Ming, Chen, Xin, Li, Xin, Ma, Bin, and Vitanyi, Paul (2003). The similarity metric. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, pages 863–872.

Liu, Y., Yao, X., Zhao, Q., and Higuchi, T. (2001). Evolving a cooperative population of neural networks by minimizing mutual information. In *Proc. of the 2001 IEEE Congress on Evolutionary Computation (CEC'01)*.

Muharram, Mohammed A. and Smith, George D. (2004). Evolutionary feature construction using information gain and gini index. In Keijzer, Maarten, O'Reilly, Una-May, Lucas, Simon M., Costa, Ernesto, and Soule, Terence, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 379–388, Coimbra, Portugal. Springer-Verlag.

Principe, J., Fisher, and Xu, D. (2000). Information theoretic learning. In *Unsupervised Adaptive Filtering*.

Radcliffe, N. (1993). Genetic set recombination. In *Foundations of Genetic Algorithms 2*.

Sprott, J. (1997). Simplest dissipative chaotic flow. *Physics Letters A*, 228.

Sprott, J. (2000). Algebraically simple chaotic flows. *International Journal of Chaos Theory and Applications*, 5(2).

Chapter 7

INVESTIGATING PROBLEM HARDNESS OF REAL LIFE APPLICATIONS

Leonardo Vanneschi[1]
[1]*Dipartimento di Informatica, Sistemistica e Comunicazione (D.I.S.Co.)*
University of Milano, Bicocca Milan, Italy

Abstract This chapter represents a first attempt to characterize the fitness landscapes of real-life Genetic Programming applications by means of a predictive algebraic difficulty indicator. The indicator used is the Negative Slope Coefficient, whose efficacy has been recently empirically demonstrated on a large set of hand-tailored theoretical test functions and well known GP benchmarks. The real-life problems studied belong to the field of Biomedical applications and consist of automatically assessing a mathematical relationship between a set of molecular descriptors from a given dataset of drugs and some important pharmacokinetic parameters. The parameters considered here are Human Oral Bioavailability, Median Oral Lethal Dose, and Plasma Protein Binding levels. The availability of good prediction tools for pharmacokinetics parameters like these is critical for optimizing the efficiency of therapies, maximizing medical success rate and minimizing toxic effects. The experimental results presented in this chapter show that the Negative Slope Coefficient seems to be a reasonable tool to characterize the difficulty of these problems, and can be used to choose the most effective Genetic Programming configuration (fitness function, representation, parameters' values) from a set of given ones.

Keywords: problem difficulty, fitness landscapes, real life applications, fitness clouds, negative slope coefficient

1. Introduction

Is Genetic Programming (GP) a suitable tool to solve my problem? How can I set parameters to make GP find better solutions? Which fitness function and representation should I choose? Although GP has been applied with success to a large number of applications of many different kinds, and besides the large amount of theory that has appeared to date – see for instance (Koza and Poli,

2003) for a short survey of GP theory and applications – still the answers to these questions are, given a particular problem, in large part unknown. What practitioners usually do when they are faced with a new complex combinatorial optimization problem is execute a set of simulations using many different GP configurations, with many different parameter settings, fitness functions and representations, and possibly other alternative Machine Learning strategies. From a comparison between the results of all these simulations, practitioners often try to empirically find the "best" algorithm and configuration for their problem. This procedure, although often successful, is very time-and computational resource-consuming. Furthermore, results of GP simulations are often difficult to interpret, given that GP, like many other optimization metaheuristics, is a stochastic (and thus non-deterministic) process. On the other hand, answering the previous questions would be much easier if an *algebraic measure* existed able, given a particular GP configuration (fitness function, representation, parameters' values), to quantify its ability to solve a given problem, *without* running GP itself.

Difficulty studies in Genetic Algorithms (GAs) have been pioneered by Goldberg and coworkers – e.g., see (Horn and Goldberg, 1995). One concept that underlies many of these studies is the notion of *fitness landscape* – e.g. see (Stadler, 2002). The fitness landscape plot can be helpful to understand the difficulty of a problem, i.e. the ability of a searcher to find good solutions for that problem – see for instance (Vanneschi, 2004; Langdon and Poli, 2002) for a deep analysis. Nevertheless, fitness landscapes are impossible to plot in practice, given the generally huge size of the space of solutions and the multi-dimentionality and complexity of the possible neighborhood structures. For this reason, in the last few years researchers have been looking for an algebraic measure able to capture some of the interesting properties of fitness landscapes. Early attemps are represented by (Weinberger, 1990; Manderick et al., 1991; Kinnear, Jr., 1994). A signicant contribution to this field has been given by Jones (Jones, 1995) with the introduction of an hardness measure for GAs called *fitness distance correlation* (*fdc*). This measure has been extended to tree-based GP and proven a suitable hardness indicator in (Vanneschi, 2004; Tomassini et al., 2005). Nevertheless, these contributions have also shown that *fdc* has some flaws, the most important one being the fact that *fdc* is not predictive, i.e. the optimal solution (or solutions) must be known beforehand, which is almost unrealistic in applied search and optimization problems. Thus, it is important to investigate other approaches based on quantities that can be measured without any explicit knowledge of the genotype of optimal solutions. Preliminary results of this enquiry can be found in (Vanneschi, 2004; Vanneschi et al., 2004; Vanneschi et al., 2006), where a new measure called *negative slope coefficient* (*nsc*) has

been introduced, and its ability to predict the difficulty of a large set of test functions[1] has been empirically demonstrated.

Here, the *nsc* is used for the first time to characterize the difficulty of real-life pharmaceutical applications and to choose the tree-based GP configuration (from a set of given ones) that is more suitable to solve them. The applications consist of automatically assessing a mathematical relationship between a set of molecular descriptors from a given dataset of drugs and some important pharmacokinetic parameters. The parameters considered here are Human Oral Bioavailability (%F), Median Oral Lethal Dose (LD50) and Plasma Protein Binding levels (%PPB). The availability of good prediction tools (for instance based on GP) for pharmacokinetics parameters like these ones is critical for optimizing the efficiency of therapies, maximizing medical success rate and minimizing toxic effects (REACH, 2006). Some important results showing the suitability of GP to solve these applications have already appeared in (Archetti et al., 2006; Archetti et al., 2007a; Archetti et al., 2007b).

This chapter is structured as follows: Section 2 defines the *nsc*. The applications studied here are presented in Section 3. The experimental setting used to test the *nsc* on these applications is described in section 4. Section 5 presents the experimental results. Section 6 contains a discussion about the contents of this chapter. Finally, Section 7 concludes the chapter.

2. Negative Slope Coefficient

Negative Slope Coefficient is based on the concepts of *evolvability* and *fitness clouds*. Evolvability is a feature that is intuitively related, although not exactly identical, to problem difficulty. It has been defined as the ability of genetic operators to improve fitness quality (Altenberg, 1994). The most natural way to study evolvability is, probably, to plot the fitness values of individuals against the fitness values of their neighbours, where a neighbour is obtained by applying one step of a genetic operator to the individual. Such a plot has been presented in (Collard et al., 2004; Barnett, 2003) and it is called a fitness cloud.

Since high-fitness points tend to be much more important than low-fitness ones in determining the behaviour of EAs, an alternative algorithm to generate fitness clouds was proposed in (Vanneschi et al., 2004). The main steps of this algorithm can be informally summarised as follows:

- Generate a set of individuals $\Gamma = \{\gamma_1, ..., \gamma_n\}$ by sampling the search space and let $f_i = f(\gamma_i)$, where $f(.)$ is the fitness function.

[1] These test functions include a set of hand-tailored theoretical functions, like Trap Functions, Royal Trees and the MAX problem and a set of well known GP benchmarks like the even parity, the multiplexer, the artificial ant on the Santa Fe trail, various instances of symbolic regression and the intertwined spirals problem.

- For each $\gamma_j \in \Gamma$ generate k neighbours, $v_1^j, \ldots, v_k^j$, by applying a genetic operator to γ_j and let $f'_j = \max_j f(v_j)$. This is equivalent to selecting one neighbor of individual γ_j by applying tournament selection of size k to its neighbors (Koza, 1992).
- Finally, take $C = \{(f_1, f'_1), \ldots, (f_n, f'_n)\}$ as the fitness cloud. This is the interpretation of fitness cloud used in this work.

The genetic operator used in this work to obtain the set of neighbors $\{v_1^j, \ldots, v_k^j\}$ of each individual $\gamma_j \in \Gamma$ is subtree mutation (Koza, 1992) in which the probability of selecting a node n in γ_j is in inverse proportion to the level of n in the tree: the leaves have the maximum probability of being chosen, while the root has a probability equal to zero[2]. Note how the previous algorithm essentially corresponds to the sampling produced by a set of n stochastic hill-climbers at their first iteration after initialisation. The fitness cloud can be of help in determining some characteristics of the fitness landscape related to evolvability and problem difficulty. But the mere observation of the scatterplot is not sufficient to quantify these features. The *nsc* has been defined to capture with a single number some interesting characteristics of fitness clouds. It can be calculated as follows: let us partition C into a certain number of separate ordered "bins" $C_1, \ldots, C_m$ such that $(f_a, f'_a) \in C_j$ and $(f_b, f'_b) \in C_k$ with $j < k$ implies $f_a < f_b$. Consider the averages fitnesses $\bar{f}_i = \frac{1}{|C_i|} \sum_{(f,f') \in C_i} f$ and $\bar{f}'_i = \frac{1}{|C_i|} \sum_{(f,f') \in C_i} f'$. The points $(\bar{f}_i, \bar{f}'_i)$ can be seen as the vertices of a polyline, which effectively represents the "skeleton" of the fitness cloud. For each of the segments of this we can define a *slope*, $S_i = (f'_{i+1} - f'_i)/(f_{i+1} - f_i)$. Finally, the negative slope coefficient is defined as:

$$nsc = \sum_{i=1}^{m-1} \min(0, S_i). \tag{7.1}$$

The hypothesis proposed in (Vanneschi, 2004; Vanneschi et al., 2004; Vanneschi et al., 2006) is that *ncs* should classify problems in the following way: if *nsc*$= 0$, the problem is easy; if *nsc*< 0 the problem is difficult and the value of *nsc* quantifies this difficulty: the smaller its value, the more difficult the problem. The informal justification for this hypothesis is that the presence of a segment with negative slope would indicate bad evolvability for individuals having fitness values contained in that segment as neighbours are, on average, worse than their parents in that segment. This intuition is the sole motivation that has been introduced until now for the use of the *nsc* as an hardness indicator, and it is surely not sufficient to justify its use in general. Nevertheless, in (Poli

[2]Otherwise the neighborhood of each individual would be the whole search space.

and Vanneschi, 2007) we tried to perform a first step towards a more formal justification of the *nsc* for GAs and a theoretical study of the same kind for GP is foreseen as one of our future reserach activities. Furthermore, the definition of the nsc, as given above, is very general and has many degrees of freedom. In particular, two questions must be answered to be able to calculate the *nsc*: (1) how should we sample the search space to generate the set of individuals $\Gamma = \{\gamma_1, ..., \gamma_n\}$? (2) how should we partition the abscissas of a fitness cloud into bins? The method used in this paper to sample the search space is based on the well known *Metropolis-Hastings* (Madras, 2002) algorithm, whose suitability has been discussed in (Vanneschi, 2004; Vanneschi et al., 2004), while the technique used to partition the abscissas of fitness clouds into bins is called *size driven bisection* and it has been presented and justified in (Vanneschi, 2004; Vanneschi et al., 2006). These two techniques will not be presented here for lack of space. The interested reader is referred to the references.

3. The Applications

Because of technical and scientific advances in combinatorial chemistry, high throughput screening (HTS), genomics, metabolomics and proteomics, pharmaceutical research is currently changing. In fact, in the traditional drug discovery process once a target protein has been identified and validated, the search of lead compounds begins with the design of a structural molecular fragment (scaffold) with therapeutic potency. Libraries of millions of chemical compounds built on the identified active fragment are then tested and ranked according to their specific biological activities. After these assays, some candidate drugs are selected from the library for more specific functional screenings (see Figure 7-1.a). Although proteomics research generated an impressive number of previously unknown target structures, their biological validation is still an hazardous task which can, as indeed has happened recently, lead to failures in drug development projects. It is interesting to remark that, both in 1991 (Kennedy, 1997) and in 2000 (Kola and Landis, 2004), a considerable fraction of attritions (i.e. failures of compounds development) in pharmacological development were generated at the level of pharmacokinetics and toxicology (see Figure 7-1.b). Good drugs in fact have not only to show good target binding, but must also follow a proper route into the human body without causing toxic effects. First of all they have to be absorbed from the gut wall and then to enter into hepatic circulation in the portal vein. Carried by the blood flux and possibly bound to plasma proteins, molecules arrive in the liver, where biochemical processes that try to destroy them take place. Only the fraction of drug initially administered that exits the liver and enters the systemic blood circulation will be available to produce some therapeutic effect.

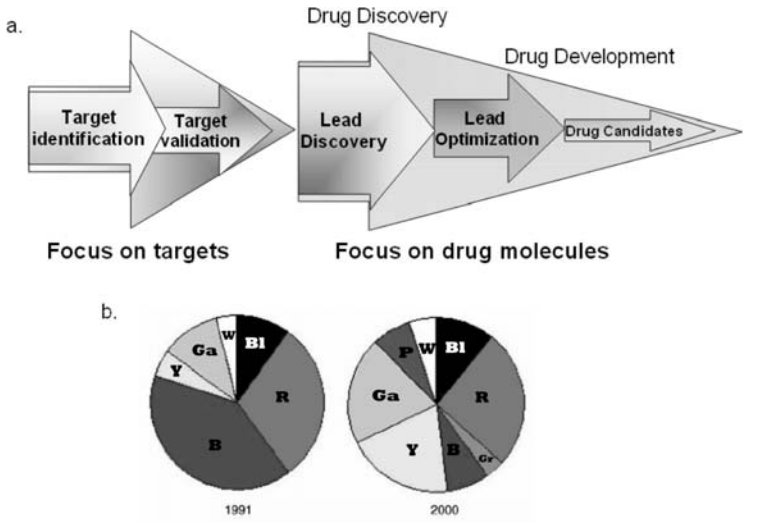


Figure 7-1. **a)** The process of drug discovery from target protein identification to candidate drugs: the identification and validation of the target are followed by lead discovery and optimization. **b)** Reasons for failure in drug development in 1991 and 2000: clinical safety (Bl), efficacy (R), formulation (Gr), PK/bioavailability (B), commercial (Y), toxicology (Ga), cost of goods (P) and others (W)

Although the development of assays evaluating pharmacokinetics parameters resulted in a reduction of attrition rate related to these properties, failures still produce an unacceptable burden on the budget of pharmaceutical companies. Thus, it is necessary to deeply characterize the behaviors of the pharmacological molecules in terms of adsorption, distribution, metabolism, excretion processes, collectively referred to as ADMET (J. P. Eddershaw and Bayliss, 2000). Automatic assessment of drug discovery by means of computer simulations is becoming a reality, and the development of computational tools applicable for ADMET profiling, also enabling the management of large and heterogeneous databases, would be of outmost relevance (van de Waterbeemd and Gifford, 2003; Norinder and Bergstrom, 2006). The availability of reliable pharmacokinetics prediction tools would permit to reduce the risk of late-stage research failures and would enable a decrease in the number of experiments and animal testing used in pharmacological research, by optimizing the screening assays. Furthermore, predictive ADMET models would be of critical relevance for preventing Adverse Drug Reactions (ADRs) like those involved in the recent Lipobay-Baycol (cerivastatin) toxicity (Tuffs, 2001), that can be very dangerous for patients. The potential of predictive modeling in terms of ADRs prediction is one more reason why computational ADMET can be considered an exciting research topic in medicine.

In this paper, we study the suitability of using GP for predicting the values of Human Oral Bioavailability, Median Oral Lethal Dose and Plasma Protein Binding levels.

Human oral bioavailability (indicated with %F from now on) is the parameter that measures the percentage of initial drug dose that effectively reaches the systemic blood circulation after the passage from the liver. This parameter is particularly relevant, because the oral assumption is usually the preferred way for supplying drugs to patients and because it is a representative measure of the quantity of active principle that effectively can actuate its therapeutic effect. Oral bioavailability is determined by two key ADMET processes: adsorption and metabolism.

The Median Oral Lethal Dose (indicated with LD50 from now on), is one of the parameters measuring the toxicity of a given compound. More precisely, LD50 refers to the amount of compound required to kill 50% of the test organisms (cavies). It is usually expressed as the number of milligrams of drug related to one kilogram of mass of the model organism (mg/kg). Depending on the specific organism (rat, mice, dog, monkey and rabbit usually), and on the precise way of supplying (intravenous, subcutaneous, intraperitoneal, oral generally) that are chosen in the experimental design, it is possible to define a spectrum of LD50 protocols. In this work the Median Lethal Dose measured in rats with the compound orally supplied is considered, which represents the most used protocol.

The Plasma Protein Binding level (indicated with %PPB from now on) corresponds to the percentage of the drug initial dose that reaches blood circulation and binds the proteins of plasma (Berezhkovskiy, 2006). This measure is fundamental for good pharmacokinetics, both because blood circulation is the major vehicle of drug distribution into human body and since only free (un-bound) drugs can permeate the membranes reaching their targets. Pharmacological molecules bind a variety of proteins in the blood, including albumin, α_1-acid glycoproteins, lipoproteins and α, β, γ-globulins, and each one of this proteins contribute in determining the binding level. In this paper, %PPB is defined as the binding levels between the drug and all the proteins contained in the plasma, avoiding the single specific drug-protein interactions.

4. Experimental Protocol Settings

In this section the procedures used for datasets collection and preparation are first introduced and then the two different GP variants studied in this work are described.

Dataset collecting and preparation

We have obtained a set of molecular structures and the corresponding %F, LD50 and %PPB values using the same data as in (Yoshida and Topliss, 2000) and a public database of food and drug Administration (FDA) approved drugs and drug-like compounds (Wishart et al., 2006). Chemical structures are all expressed as SMILES code (Simplified Molecular Input Line Entry Specification), i.e. strings codying the 2D molecular structure of a compound in an extremely concise form. The resulting libraries of molecules contained 260 (respectively 234 and 662) molecules with measured %F (respectively LD50 and %PPB) values. SMILES strings belonging to the %F dataset have been used to compute 241 bi-dimensional molecular descriptors using ADMET Predictor – a software produced by Simulation Plus Inc.(Inc, 2006). The features for molecules with known values of LD50 and %PPB have instead been calculated using the on-line DRAGON software (Tetko et al., 2005), which returned 626 bi-dimensional molecular descriptors. Thus, data have been gathered in matrices composed of 260 (respectively 234 and 662) rows and 242 (respectively 627 and 627) columns. Each row is a vector of molecular descriptors values identifying a drug; each column represents a molecular descriptor, except the last one, which contains the known values of %F (respectively LD50 and %PPB). These three datasets, comprehensive of SMILES data structures can be downloaded from the webpage: *http://www.life.disco.unimib.it/Ricerca.html*.

Genetic Programming Configurations

In (Archetti et al., 2006; Archetti et al., 2007a; Archetti et al., 2007b) various different GP configurations have been used to predict %F, LD50 and %PPB. The ones that have returned the best results, and thus the ones that will be studied in this paper, have been called *"canonic" or standard GP* (stdGP) and *Linear Scaling with 2 criteria and ephemeral random Costants GP* (LS2-C-GP). They are described in details and justified in the above references. A brief description is given below. Finally, a brief summary of the results obtained by these two GP versions in (Archetti et al., 2006; Archetti et al., 2007a; Archetti et al., 2007b) for %F, LD50 and %PPB prediction is presented.

"Canonic" (or standard) GP. The first GP setting that will be studied (called *canonic* or *standard* GP and indicated as stdGP), is a deliberately simple version of standard tree-based GP (Koza, 1992). In particular, we used the parameter setting, sets of functions and terminal symbols as similar as possible to the ones originally adopted in (Koza, 1992) for symbolic regression problems. Each molecular feature has been represented as a floating point number. Potential solutions (GP individuals) have been built by means of the set of functions $F = \{+, *, -, \%\}$ (where $\%$ is the protected division, i.e. it returns 1 if the

denominator is zero), and the set of terminals T composed by n floating point variables (where n is the number of columns in the training sets, i.e. the number of molecular descriptors of each compound). The fitness of each individual has been defined as the root mean squared error ($RMSE$) between expected outputs and the results returned by the GP candidate solution.

LS2-C-GP. The second version of GP differs from the previous one for the fitness function and the set of terminal symbols employed. Fitness is a weighted average between the $RMSE$ with linear scaling – a detailed description of this method can be found in (Keijzer, 2003) – and the *correlation coefficient* (*CC*) between expected outputs and the results returned by the GP candidate solution. Weights equal to 0.4 and 0.6 have been respectively assigned to $RMSE$ with linear scaling and *CC*, after that both of them have been normalized into the range $[0, 1]$, thus giving slightly higher importance to *CC*. These values have been empirically chosen through a simple experimentation phase (whose results are not shown here). The idea behind this weighted sum is that the *CC* between outputs and targets is a very important measure for results accuracy and thus it deserves to be used as an optimization criterium. The reason why we use GP with a different fitness function is – as it has been shown in (Vanneschi et al., 2007) – that using more different criteria for evaluating regression models is often beneficial for generalization.

Furthermore, in LS2-C-GP a set of ephemeral random constants ($ERCs$) is added to the set of terminal symbols to code GP expressions. These $ERCs$ are generated uniformly at random from the range $[m, M]$, where m and M are the minimum and the maximum target values in the dataset respectively. In the experiments presented in this paper, a number of $ERCs$ equal to the number of floating point variables has been used. This choice has been empirically confirmed to be suitable by a set of GP runs in which different numbers of $ERCs$ extracted from different ranges have been used (the results of these experiments are not shown here).

Previous Results

In (Archetti et al., 2006; Archetti et al., 2007a; Archetti et al., 2007b), performances of stdGP and LS2-C-GP to approximate %F, LD50 and %PPB values have been compared between them and with a set of other well known Machine Learning techniques including Artificial Neural Networks, Support Vector Machines and Linear and Least Square Regressions. The results obtained in that work may be summarized as follows: both GP methods outperform the other Machine Learning strategies. LS2-C-GP outperforms stdGP for %F and %PPB approximations, while it is slightly outperformed by stdGP for the LD50 approximation – see (Archetti et al., 2007b) for the experimental results and a detailed discussion of them. The reason for this behavior is probably due to the

fact that the relationship between molecular descriptors and %F and %PPB in our datasets can be reliably approximated by a linear function, while it is not the case for LD50. Thus, the linear scaling and the correlation coefficient are not helpful for optimizing the LD50 dataset.

5. Experimental Results

Once a measure of hardness, like the *nsc*, and the way to compute it have been chosen, the problem remains of finding a means to validate the prediction of the measure with respect to the problem instance and the algorithm. The easiest way is to use a *performance* measure (Naudts and Kallel, 2000). For the present work, performance is defined as being the proportion of the runs for which a satisfactory solution has been found in less than 500 generations over 100 runs. In this paper, a solution x is considered satisfactory if its normalized RMSE is larger than 0.99 (all fitness criteria reported below, included the RMSE, have been normalized into the range $[0, 1]$ in such a way that the best possible fitness value is equal to 1, and the worst is 0). Even if this performance definition is informal and prone to criticism, good or bad performance values correspond to our intuition of what "easy" or "hard" means in practice. In all GP runs executed to calculate performance values the same set of GP parameters have been used both for stdGP and LS2-C-GP: population size of 500 individuals; ramped half-and-half initialization; tournament selection of size 10; maximum tree depth equal to 10; standard subtree mutation used as the sole genetic operator and applied with a rate of 0.95; elitism, i.e. unchanged copy of the best individual in the population at each generation. The other parameters (i.e. the set of terminal symbols and the fitness function) change from a GP version to the other, and they have separately been discussed in Section 4 for each GP configuration. To obtain the fitness clouds reported below, samples of 40000 GP individuals have been generated with the Metropolis-Hastings algorithm and, for each one of these individuals, one neighbor has been chosen applying tournament selection of size 10 to its neighborhood.

Empirical results obtained for the %F approximation are summarized by Figure 7-2 and Table 7-1. Figure 7-2(a) as well as the first row in Table 7-1

Table 7-1. Bioavailability (%F) symbolic regression. Indicators related to scatterplots of Figure 7-2.

scatterplot	algorithm	p	nsc
Fig. 7-2(a)	stdGP	0.27	-1.26
Fig. 7-2(b)	LS2-C-GP	0.39	-0.47

concern the results obtained using stdGP. Performance (p in the table) of stdGP

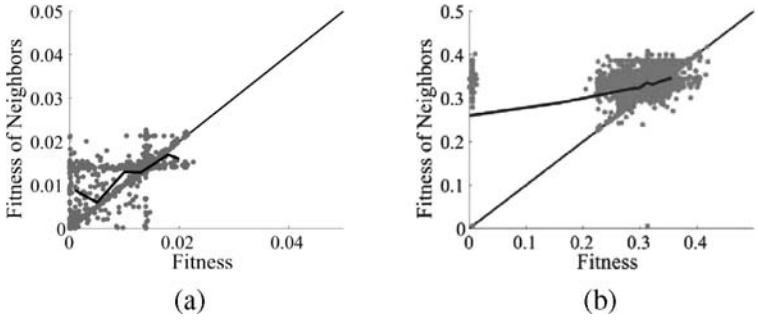


Figure 7-2. Bioavailability (%F) symbolic regression. Fitness clouds and segments. (a) Standard GP. (b) LS2-C-GP.

is equal to 0.27 (which means that a satisfactory solution has been found in 27 runs over 100 before generation 500). Given that p is smaller than 0.5, this problem could be broadly classified as a "difficult" one (the same thing holds for all the other ones considered in this work). The value of the *nsc* is equal to -1.26, which means that also according to the *nsc* measure the problem is difficult to solve. Thus, the *nsc* gives the correct indication about the hardness of this problem. Results obtained using LS2-C-GP are summarized by Figure 7-2(b) as well as the second row in Table 7-1. The first thing that we remark is that when LS-2-GP is used, the points composing the the fitness cloud are not as close to each other as they are when stdGP is used: the fitness cloud appears to be partitioned into four separated sub-clouds and one of those sub-clouds (the one that contains the larger number of points) is located at high fitness values, while the points reported in Figure 7-2(a) seem to be clustered around bad fitness values. In other words, if LS2-C-GP is used, we are able to sample a large number of individuals with better fitness. The main reason is probably the fact that linear scaling is used for calculating fitness in LS2-C-GP; in fact, expressions with a similar "shape" to the target function, but with different position and variance are bad GP individuals if only the *RMSE* is used to evaluate fitness (as it is the case for stdGP), while they have a good quality if linear scaling is used. Also the fact that *CC* is included in the fitness calculation of LS2-C-GP probably contributes to a general fitness improvement. Performance obtained by LS2-C-GP is equal to 0.39 and thus, although the problem remains a difficult one, it is clearly less difficult than for stdGP, since satisfactory solutions are found more frequently. This is also confirmed by the *nsc* value. In fact, the *nsc* is equal to -0.47, i.e. even though the problem is predicted a difficult one ($nsc < 0$), it is also predicted less difficult than for stdGP ($nsc > -1.26$). For all the *nsc* values reported in this work, standard deviations of the average points of the polylines have been calculated (results not shown here for lack of space) and all the differences in the *nsc* values

between the different GP versions seem to be statistically significant. These results suggest that the *nsc*, besides being a reliable hardness indicator, may also be used for choosing the more suitable GP configuration (fitness function, representation, parameter setting) among a set of given ones. This could be very useful for practitioners, since it would be sufficient to look for the configuration that maximizes the *nsc*, instead of executing many time consuming simulations.

The same considerations also qualitatively hold for the %PPB approximation, whose results are shown in Figure 7-3 and Table 7-2. In fact, also in this case,

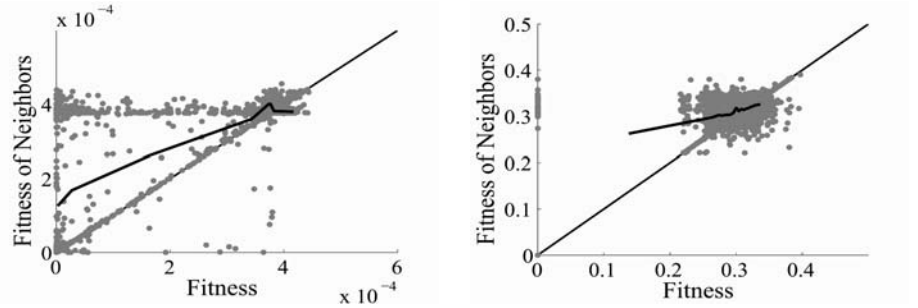


Figure 7-3. Plasma Protein Binding level (%PPB) symbolic regression. Fitness clouds and segments. (a) Standard GP. (b) LS2-C-GP.

Table 7-2. Plasma Protein Binding level (%PPB) symbolic regression. Indicators related to scatterplots of Figure 7-3.

scatterplot	algorithm	p	nsc
Fig. 7-3(a)	stdGP	0.18	-3.09
Fig. 7-3(b)	LS2-C-GP	0.24	-2.26

the problem can be broadly classified as "difficult" both if stdGP and LS2-C-GP are used (performance is smaller than 0.5 in both cases), but it is less difficult when LS2-C-GP is used (24 satisfactory solutions found by LS2-C-GP before generation 500 over 100 runs, against 18 for stdGP). Both the position of the cloud and the *nsc* values confirm this trend.

Finally, results obtained on the LD50 dataset are shown in Figure 7-4 and in Table 7-3. Contrarily to what happens for %F and %PPB, in this case stdGP slightly outperforms LS2-C-GP (13 satisfactory solutions found by stdGP over 100 runs, against 9 ones found by LS2-C-GP), even though the problem is rather difficult independently from the GP version used. This fact is confirmed by the *nsc*, that has rather low values in both cases but, despite the position of the

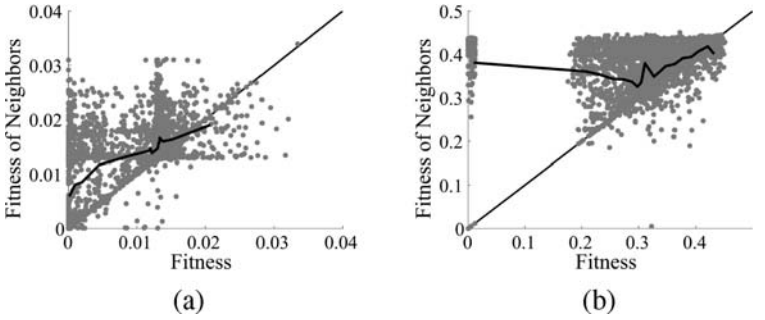


Figure 7-4. Median Oral Lethal Dose (LD50) symbolic regression. Fitness clouds and segments. (a) Standard GP. (b) LS2-C-GP.

Table 7-3. Median Oral Lethal Dose (LD50) symbolic regression. Indicators related to scatterplots of figure 7-4.

scatterplot	algorithm	p	nsc
Fig. 7-4(a)	stdGP	0.13	-5.45
Fig. 7-4(b)	LS2-C-GP	0.09	-6.13

cloud, is slightly larger for stdGP. The conclusion is that, also in this case, the *nsc* is larger for the parameter setting and the fitness function that have returned the best results. Once again, maximizing the *nsc* seems to be a reasonable way to choose the right GP configuration.

6. Discussion

For the results reported in the previous section, both the GP simulations and the *nsc* calculations have been done using all the data in our datasets. In other words, this chapter does not deal with the problem of generalization. Can the *nsc* be used to test the generalization ability of GP? This is a very ambitious question. As already mentioned above in this chapter, the *nsc* as defined until now is still in many senses "empirical". For instance, the minimum number of points that a bin can contain and the maximum admissible size (in abscissa) of the bins have been chosen in an empirical way. Furthermore, a formal justification for the *nsc* approach is still missing, even though a significant first step has been done for AGs in (Poli and Vanneschi, 2007). All this considered, the fact that the *nsc* has revealed a reliable hardness indicator on the same data as the ones used to calculate fitness for so many test problems of so many different difficulties – see (Vanneschi, 2004; Vanneschi et al., 2004; Vanneschi et al., 2006) besides this chapter – has already to be considered a surprisingly

good result. Nevertheless, one may imagine to take the same individuals as the ones used to calculate the fitness cloud of a training set and to calculate their fitness on a test set, generating in this way a new fitness cloud. This new fitness cloud may then be used as an indicator of the generalization ability of GP. This approach will be investigated in the future.

Besides the fact that it is still "empirical" from many points of view, one of the major limitations of the *nsc* in its current definition is that it only takes into account mutation, ignoring crossover which is typically the most used genetic operator in GP. How could we build a fitness cloud considering neighborhoods induced by crossover? Crossover can, in some senses, be though of as a form of mutation, where the probability distribution that allows to build the replacing subtree is not given *a priori*, but is, in some senses, determined by the evolution dynamics. If one gets to know this probability distribution, then fitness clouds may be built as it has been done until now, just using this probability distribution to build the neighbors of the sampled individuals. On the other hand, assuming that this probability is unknown, the easiest way to obtain such a fitness cloud could probably be given by the following algorithm:

- Generate two samples of individuals S_1 and S_2 and repeat until both S_1 and S_2 are empty:
  - Take one individual i_1 from S_1, one individual i_2 from S_2, perform the crossover between i_1 and i_2, let j_1 and j_2 be the offspring and let j be the individual with better fitness among j_1 and j_2;
  - Plot a point (i_1, i_2, j) on a 3D plane;
  - Eliminate i_1 from S_1 and i_2 from S_2;

this would generate a tridimensional scatterplot. Successively one may study some techniques to partition it into bins and joining the centroids of these bins would allow one to calculate a *nsc*. Although very challenging, this research activity will also be investigated in the future.

7. Conclusions

The negative slope coefficient (*nsc*) is a predictive indicator of problem difficulty for GP that has proven rather reliable on many hand-tailored functions and standard GP benchmarks (Vanneschi, 2004; Vanneschi et al., 2004; Vanneschi et al., 2006). It is based on the concept of fitness cloud, first introduced in (Vérel et al., 2003), which is a plot of the fitness values of individuals against the fitness of their neighbours. In this paper, the *nsc* has been applied for the first time to three important pharmaceutical applications, to quantify their difficulty for GP and to choose the right GP configuration (parameter setting, fitness function and representation) from a set of given ones. The goal of these applications is to

automatically assess a mathematical relationship between a set of molecular descriptors from a given dataset to approximate the values of Oral Bioavailability (%F), Median Oral Lethal Dose (*LD50*) and Plasma Protein Binding (*%PPB*) levels of drugs. The availability of good prediction tools for pharmacokinetics parameters like the ones studied in this paper is critical for optimizing the efficiency of therapies, maximizing medical success rate and minimizing toxic effects. Furthermore, computational ADMET predictive tools development has been encouraged in UE community by the REACH (REACH, 2006) proposal, whose aim is to improve the protection of human health through the better and earlier identification of the toxicity of chemical substances. Our experimental results have shown that, in all the three applications considered, the GP configuration that returns the best results is also the one that maximizes the *nsc* value. These results are encouraging and pave the way to a wider application of the *nsc* as a predictive tool for real-life GP applications.

Nevertheless, this work leaves some open questions about the *nsc*: first of all, it is based on statistical samplings of the search space and thus counterexamples can surely be built for this measure. Secondly, and even more importantly, no technique has been found yet to normalize *nsc* values into a given range, in order to enable comparisons between the difficulties of two or more problems of different nature. Only the hardness of different instances of the same problem can be calculated using *nsc*, as it has been defined until now – a deep discussion of this *nsc* drawback is contained in (Vanneschi, 2004). Third, no theoretical foundation, nor formal justification for the use of the *nsc* as a hardness indicator has ever been given to date. The only justification put forward for the use of the *nsc* is that the presence of a segment with negative slope in the polyline which represents the skeleton of a fitness cloud indicates a bad evolvability for individuals having fitness values contained in that segment as neighbours are, on average, worse than their parents in that segment (Vanneschi, 2004). This justification clearly lacks formality and is too weak to justify the use of the *nsc* as a predictive tool of applications hardness on a large industrial scale. Thus, future research includes attempts to find a new and more formal definition and theoretical justification of the *nsc*. One significant first step in this direction has recently appeared in (Poli and Vanneschi, 2007). Furthermore, in order to deeply characterize the ability of GP in the ADMET in silico arena, we are planning to test our methodologies on other datasets, for example for the prediction of Blood Brain Barrier Permeability or Cytochrome P450 interactions.

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In Kinnear, K., editor, *Advances in Genetic Programming*, pages 47–74, Cambridge, MA. The MIT Press.

Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007a). Genetic programming and other machine learning approaches to predict median oral lethal dose (LD50) and plasma protein binding levels (of drugs. In *et al.*, E. Marchiori, editor, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Proceedings of the Fifth European Conference, EvoBIO 2007*, Lecture Notes in Computer Science, LNCS 4447, pages 11–23. Springer, Berlin, Heidelberg, New York.

Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007b). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines, special issue on Medical Applications*. To appear. Submitted on November 18, 2006.

Archetti, Francesco, Lanzeni, Stefano, Messina, Enza, and Vanneschi, Leonardo (2006). Genetic programming for human oral bioavailability of drugs. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 255–262, Seattle, Washington, USA. ACM Press.

Barnett, L. (2003). *Evolutionary Search on Fitness Landscapes with Neutral Networks*. PhD thesis, University of Sussex.

Berezhkovskiy, L. M. (2006). Determination of drug binding to plasma proteins using competitive equilibrium binding to dextran-coated charcoal. *Journal of Pharmacokinetics and Pharmacodynamics*, 33(5):920–937.

Collard, P., Verel, S., and Clergue, M. (2004). Local search heuristics: Fitness cloud versus fitness landscape. In Mántaras, R. L. De and Saitta, L., editors, *2004 European Conference on Artificial Intelligence (ECAI04)*, pages 973–974, Valence, Spain. IOS Press.

Horn, J. and Goldberg, D. E. (1995). Genetic algorithm difficulty and the modality of the fitness landscapes. In Whitley, D. and Vose, M., editors, *FOGA-3*, pages 243–269. Morgan Kaufmann.

Inc, Simulation Plus (2006). a company that use both statistical methods and differential equations based simulations for adme parameter estimation. See www.simulationsplus.com.

J. P. Eddershaw, A. P. Beresford and Bayliss, M. K. (2000). Adme/pk as part of a rational approach to drug discovery. *Drug Discovery Today*, 9:409–414.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque.

Keijzer, Maarten (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, Conor, Soule, Terence, Keijzer, Maarten, Tsang, Edward, Poli, Riccardo, and Costa, Ernesto, editors, *Genetic Pro-*

gramming, Proceedings of EuroGP'2003, volume 2610 of *LNCS*, pages 70–82, Essex. Springer-Verlag.

Kennedy, T. (1997). Managing the drug discovery/development interface. *Drug Discovery Today*, 2:436–444.

Kinnear, Jr., Kenneth E. (1994). Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, Orlando, Florida, USA. IEEE Press.

Kola, I. and Landis, J. (2004). Can the pharmaceutical industry reduce attrition rates? *Nature Reviews Dug Discovery*, 3:711–716.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R. and Poli, Riccardo (2003). A genetic programming tutorial. www.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Madras, N. (2002). *Lectures on Monte Carlo Methods*. American Mathematical Society, Providence, Rhode Island.

Manderick, B., de Weger, M., and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150. Morgan Kaufmann.

Naudts, B. and Kallel, L. (2000). A comparison of predictive measures of problem difficulty in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):1–15.

Norinder, U. and Bergstrom, C. A. S. (2006). Prediction of admet properties. *ChemMedChem*, 1:920–937.

Poli, R. and Vanneschi, L. (2007). Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms. In *Proceedings of the 9th annual conference on Genetic and Evolutionary Computation, GECCO 2007*, London, UK. To appear. Nominated for the *best paper award* of the Genetic Algorithms track.

REACH (2006). Registration, evaluation and authorisation of chemicals.

Stadler, P. F. (2002). Fitness landscapes. In Lassig, M. and Valleriani, A., editors, *Biological Evolution and Statistical Physics*, volume 585 of *Lecture Notes Physics*, pages 187–207. Springer, Berlin, Heidelberg, New York.

Tetko, I. V., Gasteiger, J., Todeschini, R., Mauri, A., Livingstone, D., Palyulin, P. Ertland V.A., Radchenko, E.V., Zefirov, N.S., Makarenko, A.S., Tanchuk, V.Y., and Prokopenko, V.V. (2005). Virtual computational chemistry laboratory - design and description. *Journal of Computer Aided Molecular Design*, 19:453–63. see www.vcclab.org.

Tomassini, M., Vanneschi, L., Collard, P., and Clergue, M. (2005). A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 13(2):213–239.

Tuffs, A. (2001). Bayer faces shake up after lipobay withdrawn. *British Medical Journal*, 23(317):828.

van de Waterbeemd, H. and Gifford, E. (2003). Admet in silico modeling: towards prediction paradise? *Nature Reviews Drug Discovery*, 2:192–204.

Vanneschi, L., Rochat, D., and Tomassini, M. (2007). Multi-optimization improves genetic programming generalization ability. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007*. ACM Press. To appear.

Vanneschi, Leonardo (2004). *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Sciences, University of Lausanne, Switzerland.

Vanneschi, Leonardo, Clergue, Manuel, Collard, Philippe, Tomassini, Marco, and Vérel, Sébastien (2004). Fitness clouds and problem hardness in genetic programming. In Deb, Kalyanmoy, Poli, Riccardo, Banzhaf, Wolfgang, Beyer, Hans-Georg, Burke, Edmund, Darwen, Paul, Dasgupta, Dipankar, Floreano, Dario, Foster, James, Harman, Mark, Holland, Owen, Lanzi, Pier Luca, Spector, Lee, Tettamanzi, Andrea, Thierens, Dirk, and Tyrrell, Andy, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 690–701, Seattle, WA, USA. Springer-Verlag.

Vanneschi, Leonardo, Tomassini, Marco, Collard, Philippe, and Vérel, Sébastien (2006). Negative slope coefficient. A measure to characterize genetic programming. In Collet, Pierre, Tomassini, Marco, Ebner, Marc, Gustafson, Steven, and Ekárt, Anikó, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 178–189, Budapest, Hungary. Springer.

Vérel, S., Collard, P., and Clergue, M. (2003). Where are bottleneck in nk-fitness landscapes? In *CEC 2003: IEEE International Congress on Evolutionary Computation. Canberra, Australia*, pages 273–280. IEEE Press, Piscataway, NJ.

Weinberger, E. D. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biol. Cybern.*, 63:325–336.

Wishart, D.S., Knox, C., Guo, A. C., Shrivastava, S., Hassanali, M., Stothard, P., Chang, Z., and Woolsey, J. (2006). Drugbank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research*, 34. doi:10.1093/nar/gkj067.

Yoshida, F. and Topliss, J. G. (2000). Qsar model for drug human oral bioavailability. *Journal of Medicinal Chemistry*, 43:2575–2585.

Chapter 8

IMPROVING THE SCALABILITY OF GENERATIVE REPRESENTATIONS FOR OPEN-ENDED DESIGN

Gregory S. Hornby[1]

[1] *U. C. Santa Cruz, Mail Stop 269-3, NASA Ames Research Center, Moffett Field, CA, 94035*

Abstract With the recent examples of the human-competitiveness of evolutionary design systems, it is not of interest to scale them up to produce more sophisticated designs. Here we argue that for computer-automated design systems to scale to producing more sophisticated results they must be able to produce designs with greater structure and organization. By "structure and organization" we mean the characteristics of modularity, reuse and hierarchy (MR&H), characteristics that are found both in man-made and natural designs. We claim that these characteristics are enabled by implementing the attributes of combination, control-flow and abstraction in the representation, and define metrics for measuring MR&H and define two measures of overall structure and organization by combining the measures of MR&H. To demonstrate the merit of our complexity measures, we use an evolutionary algorithm to evolve solutions to different sizes for a table design problem, and compare the structure and organization scores of the best tables against existing complexity measures. We find that our measures better correlate with the complexity of good designs than do others, which supports our claim that MR&H are important components of complexity. We also compare evolution using five representations with different combinations of MR&H, and find that the best designs are achieved when all three of these attributes are present. The results of this second set of experiments demonstrate that implementing representations with MR&H can greatly improve search performance.

Keywords: evolutionary design, scalability, representations, complexity

1. Introduction

With improvements in software modeling packages and increases in computational power, there is growing interest in using artificial intelligence techniques to automate some of the design process. Automated design systems based on evolutionary algorithms (EAs) have been used to create interesting designs in a variety of different domains (Bentley, 1999; Bentley and Corne, 2001). Of interest is understanding how to improve existing computer-automated design systems so that they scale from designing merely a single component of a design to creating a large, complex design and all of its custom parts. In order to improve the ability of evolving design systmes (EDSs) to scale up for producing sophisticated designs, we need: a better understanding of scalability, metrics to measure designs scalability, and understanding of how computer-automated design systems enable scalability.Already various metrics exist for measuring what has been loosely defined as *complexity*, such as Algorithmic Information Content (AIC) (Chaitin, 1966; Kolmogorov, 1965; Solomonoff, 1964), Logical Depth (Bennett, 1986), and Sophistication (Koppel, 1987). These metrics vary in their degree of intuitiveness in measuring complexity. For example, the AIC of a random string will score higher than a string of the same length with hierarchies of regularities, whereas we are inclined to think that a string with the patterns is more complex. More importantly, existing complexity measures are not based on measuring characteristics of good design. Thus, rather than using these existing measures, a more useful approach may be to set them aside and develop new metrics that explicitly measure those characteristics that have been demonstrated to be useful for improving scalability.

In engineering and software development sophisticated artifacts are achieved by exploiting the principles of modularity, reuse, and hierarchy (MR&H) (Huang and Kusiak, 1998; Meyer, 1988; Ulrich and Tung, 1991), and these characteristics can also be seen in the artifacts of the natural world. Assuming that the principles of MR&H are necessary for achieving scalability, then by constructing an EDS capable of producing designs with these characteristics we can hope to achieve more scalable computer-automated design. Breaking down an EDS into its separate modules yields the representation for encoding designs, the search algorithm for exploring the space of designs that can be represented, and the fitness function for scoring the goodness of a particular design. Ideally, the ability of an EDS to create designs with hierarchies of reused modules should be independent of how designs are scored. In addition, the EA for exploring the space of designs can only find designs that can be expressed by the chosen representation. Thus for an EDS to achieve MR&H it must use a representation capable of encoding designs with these characteristics.

To be able to develop representations which can encode designs with MR&H we need to understand the fundamental attributes of design representations. One

way to analyze representations is to consider them as a kind of programming language and, using this metaphor, the properties of programming languages (combination, control-flow and abstraction (Abelson et al., 1996)) can be used to formally classify representations.

Here we show how the different properties of programming languages enable MR&H, and develop metrics for these three characteristics. In addition, we present two measures of complexity, which we call measures of *structure and organization*, and show that they better correlate with the complexity of an object than do existing measures. To support these claims we present results using different representations with different combinations of MR&H enabled on different sizes of a scalable design problem.

The rest of this chapter is organized as follows: First we describe what we mean by modularity, reuse and hierarchy, show how these characteristics are enabled by different properties of representations and give metrics for measuring them (Section 2). This is followed by a description of the other complexity measures that we compare against and our two measures of structure and organization, which are composite functions of the MR&H measures (Section 3). We then describe an experimental setup for comparing the different measures and representations on different sizes of a design problem (Section 4). Next we present our first set of results in which we show that the measures of MR&H and structure and organization better correlate with complexity with other existing measures (Section 5), and then we present our second set of results which demonstrates that enabling more of MR&H in the representation improves evolutionary performance and scalability (Section 6). Finally, we close with a summary of our work.

2. Modularity, Reuse and Hierarchy

While various metrics already exist for measuring complexity, none of them provide useful guidance on how to build better EDSs which can evolve more complex designs. To improve the sophistication of what can be evolved, we need definitions and metrics for the types of characteristics that are useful for improving scalability. Here we claim that the ability to produce designs with good structure and organization–that is, designs with the characteristics of modularity, reuse and hierachy–is the way to improving the scalability of an EDS. A useful way of measuring complexity in a design is with metrics of MR&H. Before creating metrics for measuring MR&H, we first give an overview of what we mean by these terms, and then show how they are enabled.

We define *modularity* as an encapsulated group of elements which can be manipulated as a unit. This form of modularity is related to the building block hypothesis of genetic algorithms (GAs) (Holland, 1975), which states that GAs work by testing groups of basic components and combining them to form highly

fit solutions. Modularity also helps enable both reuse and hierarchy. *Reuse* is a repetition of elements in generating a design. *Hierarchy* is the number of layers of encapsulated modules in the structure of a design. Each of these three characteristics is enabled by one or more of the fundamental properties of generative representations.

Previously we have claimed that, because generative representations are a type of algorithm for specifying a design (Hornby, 2003; Hornby, 2004; Hornby and Pollack, 2002), the fundamental properties of generative representations are the same as those for computer programming languages: combination, control-flow and abstraction (Abelson et al., 1996). Combination is the ability to hierarchically create more powerful expressions from simpler ones; abstraction is the ability to name compound elements, along with formal parameters, and manipulate them as units; and control-flow are those operators, such as conditionals and iterative constructs, which control the flow of execution. Using these definitions we can now define metrics for MR&H and show how they are enabled by the fundamental properties of programming languages.

Modularity: The modularity value of a design is a count of the number of structural modules in it, which we define as an encapsulated group of elements in the design encoding that can be manipulated as a unit. Since the lable of a procedure can be manipulated as a unit, each procedure in the design encoding adds one point to the encoded modularity value. In addition, the ability to change the iteration counter means that the group of encoded elements inside an iterative block also constitute a module; hence each iterative block is one module in the encoding. Thus, modularity is enabled by *abstraction* and *iteration*. As well as counting modules in the encoded design (which we label $\mathtt{M}_p$, for modules in the program) we can also count the number of occurrences of modules in the design itself, $\mathtt{M}_d$. In this case each procedure call counts as one toward the design modularity value and each iteration of an iterative block adds one to the modularity value of the design.

Reuse: is a measure of the average number of times parts of the design program are used to create the resulting design. Here we measure three types of reuse. The first, overall reuse, $\mathtt{R}_a$, is calculated by dividing the number of symbols in an object's assembly procedure by the number of symbols in program that generates it. Second, reuse of build symbols, $\mathtt{R}_b$, is the average number of times a design constructing operator – as opposed to an operator that is a conditional, iterative statement or procedure call – is used. Third, reuse of modules, $\mathtt{R}_m$, is the average number of times modules are reused in the design and is calculated as $\mathtt{M}_d$ divided by $\mathtt{M}_p$.

Hierarchy: The hierarchy of a design is a measure of the number of nested layers of modules, such as through iteration or abstraction. A design encoding with no modules has a hierarchy of zero. Each nested module, whether a successful call to a labeled procedure or a non-empty iterative block, increases

the hierarchy value by one. This is similar to measuring the depth of an object's assembly sequence (Goldwasser et al., 1996), but whereas with that the measure is of basic steps in constructing an object, here we are measuring levels of modules.

As defined, these measures of MR&H apply to any programming language, and are thus comparable on the same systems as existing complexity measures, such as AIC, Logical Depth and Sophistication. These measures can also be generalized to any representation with a hierarchical graph structure, such as the set of parts used to describe a composite assembly in a CAD/CAM package, and any system that can be described as a hierarchical graph structure, such as a regular expression. Not as obvious is how to apply these measures to non-procedural representations such as DNA and artificial genetic regulatory networks, for which the challenge is mainly the identification of modules.

In the rest of this chapter we use "MRH" to refer to these metrics of structure and organization and "MR&H" to refer to the characteristics of modularity, reuse and hierarchy.

3. Measures of Complexity

To demonstrate that the MRH metrics of structure and organization are meaningful we compare them against existing complexity metrics. For this comparison we selected those metrics which are relatively straightforward to compute or approximate, and which we thought had a reasonable chance at being relevant. Examples of measures we left out are: Arithmetic Complexity, Cognitive Complexity, Dimension of Attractor, Ease of Decomposition, Logical Complexity, Number of States in a Finite Automata, and Thermodynamic Depth. All of these measures, as well as many others, are reviewed in Edmunds' dissertation (Edmunds, 1999). In addition to the MRH metrics of structure and organization and several existing complexity metrics found in the literature, we also include some additional measures to serve as a kind of control variables. We now review the different complexity metrics against which we compare our measures of structure and organization.

Algorithmic Information Content (AIC) is one of the best known and influential complexity metrics, having been used as a starting point for many others (Chaitin, 1966; Kolmogorov, 1965; Solomonoff, 1964). The AIC of a given string is the length, in number of symbols, of the shortest program that produces that string. For this work we estimate the AIC by calculating the number of symbols in the design program, since this is the evolved genotype that defines the object. While it is likely that some of the evolved genotypes could be compressed, using their actual size is a simple upper bound on AIC and is a correct measure of the size of the program that was evolved. This

measure is very similar to counting the number of lines in a computer program, which is another measure of complexity (Edmunds, 1999).

Design size (DS) is a measure of the size of what is encoded by the design program (genotype), and here we measure this by counting the number of symbols in the assembly procedure. This contrasts with AIC, which counts the number of symbols in the program that generates the assembly procedure.

Logical Depth is a measure of the value of information and, for a given string, it is the minimum running time of a near-incompressible program that produces the string (Bennett, 1986). In this case we use the evolved design program as the near-incompressible program, and calculate the running time of this program as the number of symbols that are processed in generating the assembly procedure. This can also be considered computational complexity, in that it is a measure of the amount of computational time that is spent to compute the assembly procedure.

Sophistication is a measure of the structure of a string by counting the number of *control* symbols in the design program used to generate it (Koppel, 1987). In trying to measure the structure of a string, the goal for this measure is similar to the goal of the `MRH` metrics. Here we calculate the sophistication of a design by counting the number of control symbols – that is, procedure symbols, loop symbols, conditionals – in the design program that is used to generate it.

Number of Build Symbols: Whereas Sophistication is a measure of structure by counting the number of control symbols, we propose a counter measure which is a count of the number of non-control symbols in the program that is used to generate the assembly procedure. In our system, these non-control symbols are the operators that are used by the design-constructing interpreter and we call them *build* symbols, since they are used to generate a design.

Grammar Size: Any string that has a pattern can be expressed as being generated by a grammar. Simple strings with simple patterns generally have a simple grammar, thus the size of the grammar needed to produce a string serves as a measure of complexity (Edmunds, 1999). The representation used here can be thought of as a kind of grammar, with different procedures being different grammar rules. Thus to calculate the grammar size of an assembly procedure we use the program that produces it as the grammar, and count the number of production-rules in the program.

Connectivity: More complex systems have greater inter-connectedness between components, and thus the connectivity of a system can be used as a complexity measure (Edmunds, 1999). For a graph-structure, its connectivity is the maximum number of edges that can be removed before it is split into two non-connected graphs. To calculate the connectivity of a design we use the connectivity of the design program (in graph form) that is used to generate it.

Number of Branches: Related to the previous measure of complexity, this is another measure of the structure of a graph is a count of number of nodes

which are branch nodes – nodes which have two or more children. Strings have a very simple structure with no branching nodes, whereas a fully balanced binary tree will have roughly *lg(n)* branch nodes. For this measure, we apply it to the assembly procedure that is generated by the evolved design program.

Height: is the maximum number of edges that can be traversed in going from the root of the tree to a leaf node. Unlike other complexity metrics which are based on strings, this measure is for trees. This measure of complexity is related to work in formal language theory in which ease of comprehension is measured by depth of postponed symbols (Yngve, 1960) or depth and nesting, called Syntactic Depth (Rosen, 1974). As with the previous measure, we apply this to the assembly procedure that is generated by the evolved design program.

In addition, we define two overall measures of structure and organization (SO), which are products of the MRH metrics:

$$SO_1 = \frac{M_p \times R_m \times H}{AIC} \tag{8.1}$$

$$SO_2 = \frac{M_p \times R_m \times H}{DesignSize} \tag{8.2}$$

The first one, SO_1, is normalized for size by dividing by the amount of information in the design, and the second one, SO_2, is normalized by dividing by the size of the design.

4. Experimental Setup

To compare the different metrics of complexity and structure and organization we use an evolutionary algorithm to evolve designs for different sizes of a design problem and then apply the different measures to the best evolved designs of each size. We now describe the test problem and the evolutionary design system, GENRE, used for these experiments.

Test Problem

For the experiments in this chapter, the design problem we use is that of producing a 3D table out of cubes, for which the fitness function for scoring tables is a function of their height, surface structure, stability and the number of excess cubes used (Hornby, 2003; Hornby, 2004). Height is the number of cubes above the ground. Surface structure is the number of cubes at the maximum height. Stability is a function of the volume of the table, and is calculated by summing the area at each layer of the table. Maximizing height, surface structure and stability typically results in table designs that are solid volumes, thus a measure of excess cubes is used to reward designs that use

fewer cubes.

$$
\begin{aligned}
f_{height} &= \text{the height of the highest cube, } Y_{max} && (8.3)\\
f_{surface} &= \text{the number of cubes at } Y_{max} && (8.4)\\
f_{stability} &= \sum_{y=0}^{Y_{max}} f_{area}(y) && (8.5)\\
f_{area}(y) &= \text{area in the convex hull at height y} && (8.6)\\
f_{excess} &= \text{number of cubes not on the surface} && (8.7)
\end{aligned}
$$

To produce a single fitness score for a design, these five criteria are combined together:

$$\text{fitness} = f_{height} \times f_{surface} \times f_{stability} / f_{excess} \quad (8.8)$$

This problem can be scaled by varying the size of the grid. In our experiments we perform runs with sizes from $20 \times 20 \times 20$ to $80 \times 80 \times 80$.

The design constructor for making table designs starts with a single cube in an otherwise empty 3D grid and then executes the assembly procedure that was produced from executing the design program. Cubes are added to this design with the operators `forward()` and `backward()`. The current state, consisting of location and orientation, is maintained with the addition of cubes resulting in a change in the current location, and the three `rotate-[x|y|z]()` operators change the current orientation. A branching in the assembly procedure results in a split in the construction process with construction continuing with each child subtree working with its own copy of the construction state.

Representation

To encode tables, the generative representation used by GENRE is a kind of program which specifies how to construct a table. This program consists of a forest of tree-structured procedures in which each node in the tree is an operator, and operators can be procedure calls, control-flow operators, or design construction operators. Designs are created by compiling a design program into an assembly procedure of construction operators and then executing this assembly procedure to generate the artifact.

The following example of a design encoded with GENRE's generative representation consists of two labeled procedures, `Proc_0` and `Proc_1`, each with

two parameters, and the initial call to the program, `Proc_0(4.0,2.0)`:

Start with: $Proc\_0(4.0, 2.0)$

$Proc\_0(n_0, n_1)$:
$n_0 > 3.0 \rightarrow$ rotate-z(1) [Proc_0(1.0,2.0) repeat(2) [forward(n_1/2) [repeat-end [Proc_1(n_0+2.0,2.0) [forward(1)]] [] []]]]
$true \rightarrow$ rotate-z(1) [repeat(4) [rotate-y(1) [forward(n_1+1.0) repeat-end [rotate-x(1)]]] []]

$Proc\_1(n_0, n_1)$:
$n_0 > 1.0 \rightarrow$ forward(2) [Proc_1(1.0,n_1+1.0) [forward(1)] rotate-y(2) [[] Proc_1(1.0,n_1+1.0) [forward(1)]] Proc_1(n_0-2.0,n_1-1.0) [end-proc]]
$n_0 > 0.0 \rightarrow$ rotate-y(1) [[] backward(n_1) [end-proc []]]

To generate the assembly procedure, this design program is executed, starting with the statement `Proc_0(4.0,2.0)`. This results in the following assembly procedure:

```
rotate-z(1) [ rotate-z(1) [ rotate-y(1) [ forward(3)
rotate-y(1) [ forward(3) rotate-y(1) [ forward(3)
rotate-y(1) [ forward(3) rotate-x(1) ] ] ] ] [] ]
forward(1) [ forward(1) [ forward(2) [ rotate-y(1)
[ [] backward(3) [ forward(1) [] ] ] rotate-y(2)
[ [] rotate-y(1) [ [] backward(3) [ forward(1)
[] ] ] ] forward(2) [ rotate-y(1) [ [] backward(2)
[ forward(1) [] ] ] rotate-y(2) [ [] rotate-y(1) [
[] backward(2) [ forward(1) [] ] ] ] forward(2) [
rotate-y(1) [ [] backward(1) [ forward(1) [] ] ]
rotate-y(2) [ [] rotate-y(1) [ [] backward(1) [
forward(1) [] ] ] ] forward(1) ] ] ] [] [] ] [] [] ] ]
```

A graphical version of this design program is shown in Figure 8-1a, along with the corresponding assembly tree of design-construction operators, Figure 8-1b, and the resulting design, Figure 8-1c. In the images of the design program and assembly procedure, cubes represent labeled procedures and the calls to them, pyramids represent control-flow operators, and construction operators are represented by spheres.

This example design can be analyzed using the metrics of MRH and the various complexity measures. The program has six modules which are used a total of 17 times giving a modularity value of 6 for the program (M_p), and a modularity value of 17 for the design (M_d). The size of the program is 30 symbols and the

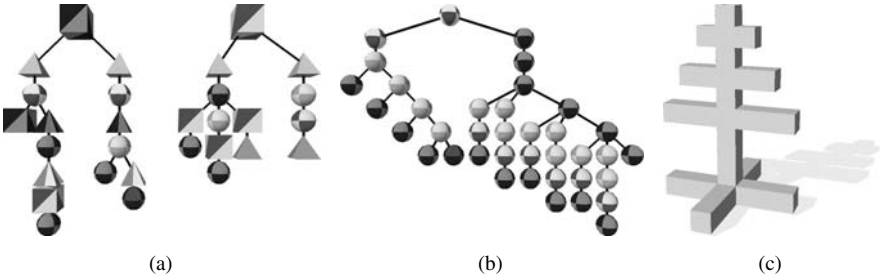


Figure 8-1. This figure contains: (a) a graphical version of an example design encoding; (b) the assembly procedure it produces; and (c) the resulting design.

size of the final assembly procedure is 38 symbols giving an overall reuse ($\mathtt{R}_a$) value of 1.27, reuse of build symbols ($\mathtt{R}_b$) of 2.92, reuse of modules ($\mathtt{R}_m$) of 2.832, and it has five levels of nested modules which gives a hierarchy value (H) of 5. Its structure and organization scores are: SO_1 is 0.78, and SO_2 is 0.62. The scores on the other complexity measures are: an AIC of 30; a Design size of 38; a Logical Depth of 124; a Sophistication of 21; 13 build symbols; a grammar size of 2; a connectivity of 5; 8 branches, and a height of 10.

Representations with Different Combinations of MR&H

In the experiments presented later in this chapter we use five representations with different combinations of MR&H enabled. These combinations are:

None: No features are enabled. In this case there are no forms of control-flow or abstraction and the program being evolved is a single procedural rule with a single body in which the condition always succeeds. None of modularity, reuse or hierarchy are enabled with this representation and the body has an upper limit of 10000 symbols.

M: Labeled procedures are enabled, but not iteration and only the first procedure, `Proc_0`, can call any other procedures. With this representation *modularity* is enabled through abstraction, but reuse is not enabled since there is neither iteration or recursion (the first procedure is not allowed to call itself) and hierarchy is limited to at most two levels. For this representation at most 25 procedures can be used, with each procedure having three conditionals, and of those subtrees having a maximum size of 1000 symbols.

MH: This representation has labeled procedures but no iteration. The labeled procedures can call each other, but a procedure can be called at most once. Here *modularity* is enabled through abstraction and *hierarchy* is enabled through the use of nested modules but reuse is not allowed. With this representation at most 25 procedures can be used, with each procedure having 3 conditionals, and each with a subtree of at most 1000 symbols.

MR: Iteration and labeled procedures are used but only the first procedure, `Proc_0`, is able to call other procedures and have iterative loops. Through these features *modularity* and *reuse* are enabled. Hierarchy is limited to two levels by only allowing iterative loops and procedure calls in the first procedure, `Proc_0`, which is not allowed to call itself. As with MH, at most 25 procedures can be used with each procedure having three conditionals and again each of these three subtrees has a maximum size of 1000 symbols.

MRH: This representation has all of the features allowed – control-flow, iteration, and labeled procedures with parameters that are able to call themselves recursively. Consequently all three of *modularity*, *reuse* and *hierarchy* are enabled. The number and size of the procedures is configured the same as with MR: 25 productions, with 3 conditional subtrees, each with a maximum size of 1000 symbols.

Evolutionary Algorithm

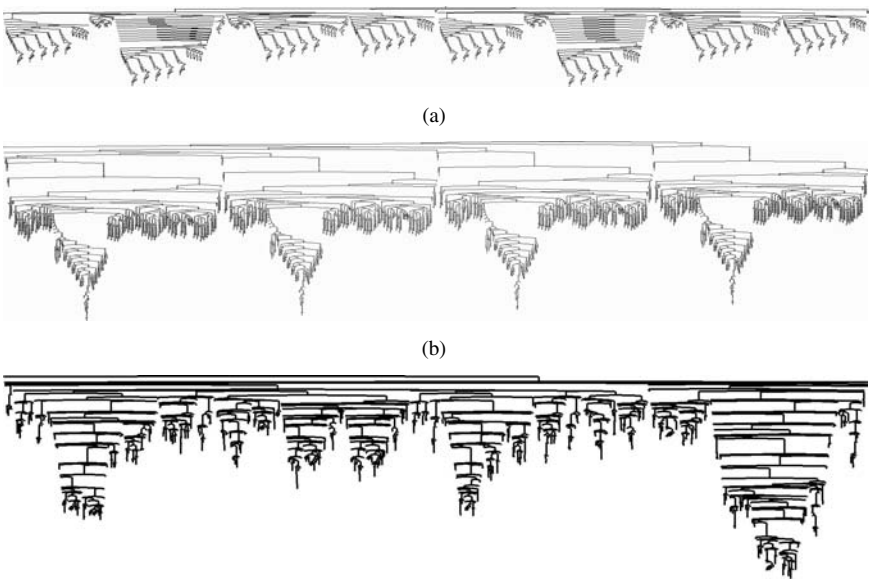
The EA used for the experiments is the Age-Layered Population Structure (ALPS) (Hornby, 2006). Unlike a traditional EA, ALPS maintains several layers of individuals of different age levels and continuously introduces new, randomly generated individuals into the first layer. It has been shown to work better than the canonical EA by better avoiding premature convergence. The setup we use consists of 10 layers, each with 40 individuals. In each layer the best 2 individuals from the previous generation are copied to the current generation and then new individuals are created with a 40% chance of mutation and 60% chance of recombination. Mutations to the genotype can be either the insertion or deletion of nodes, or a change to the parameters of a node. Recombination is a kind of GP-style swapping of subtrees from procedures in two individuals. Tournament selection with a tournament size of 5 is used to select parents. In our experiments we ran 15 trials with each configuration and each trial is run for one million evaluations.

5. Comparing Metrics

In this first set of experiments we empirically demonstrate that the characteristics of modularity, reuse and hierarchy are postively correlated with "complex" designs, and that our `MRH` metrics are as good as, or better than, other measures of complexity. Here we are working with the assumption that a more "complex" design is needed to produce good designs for a larger design space, and so we are looking for complexity metrics whose values increase along with the increase in the design problem. For these experiments we used the representation with all three of the characteristics MR&H enabled and perform 15 evolutionary runs, each on different sizes of the table design problem.

(a) (b) (c)

Figure 8-2. Two of the best, and most structurally organized, of the evolved tables using the representation `MRH`: (a) was evolved in the $20 \times 20 \times 20$ design space; (b) was evolved in the $80 \times 80 \times 80$ design space; and (c) a table evolved with representation `None` in the $80 \times 80 \times 80$ design space.



(a)

(b)

(c)

Figure 8-3. A graphical rendition of the assembly procedures for constructing the tables in Figure 8-2. The assembly procedure in (a) produces a table for the 20x20x20 design space, and the assembly procedure in (b) produces a table for the 80x80x80 design space. Both of these assembly procedures were created from compiling a design program evolved using representation `MRH`. The assembly program in (c) is the evolved genotype using representation `None`.

Figure 8-2 contains images of two of the best and most structurally organized tables that were evolved. The smaller table, Figure 8-2a, was evolved in the $20 \times 20 \times 20$ design space and has a fitness of 582221 and the following scores: AIC of 913; Design Size of 8007; Logical Depth of 10311; Sophistication of

Table 8-1. A comparison of the different complexity measures on the best tables evolved for different sizes of the design problem using representation MRH. Results are the average over 15 trials.

Measure	Problem Size			
	20^3	40^3	60^3	80^3
Fitness ($\times 10^6$)	0.56	18.1	123	440
AIC	719	768	680	775
Design Size	6769	9499	9739	9944
Logical Depth	9541	13421	14376	18011
Sophistication	79.9	70.53	74.0	85.4
Number of Build Symbols	626	684	593	676
Grammar Size	13.5	13.2	12.5	13.5
Connectivity	33.7	25.2	26.4	37.3
Number of Branch Nodes	1653	2087	1905	1825
Height	118	145	276	220
Modularity ($\mathtt{M}_p$)	27.5	26.1	30.8	31.1
Modularity in Design ($\mathtt{M}_d$)	377	547	1133	1329
Reuse ($\mathtt{R}_a$)	12.1	14.0	16.6	15.7
Reuse of Build Symbols ($\mathtt{R}_b$)	15.2	16.2	19.6	18.5
Reuse of Modules ($\mathtt{R}_m$)	15.2	21.8	37.4	50.1
Hierarchy (H)	7.53	7.7	8.0	8.6
SO_1 ($\mathtt{M}\times\mathtt{R}_m\times\mathtt{H}$ / AIC)	4.59	6.87	15.4	19.3
SO_2 ($\frac{\mathtt{M}\times\mathtt{R}_m\times\mathtt{H}}{DesignSize}$)	0.42	0.46	0.89	1.13

89; 811 build symbols; a Grammar Size of 13; a Connectivity of 34; 1595 branches; a height of 155. Its MRH scores are: $\mathtt{M}_p$ is 34, $\mathtt{M}_d$ is 431; $\mathtt{R}_a$ is 8.8; $\mathtt{R}_b$ is 9.9; $\mathtt{R}_m$ is 12.7 and it has an H of 8. The larger table, Figure 8-2b, was evolved in the $80 \times 80 \times 80$ design space and has a fitness of 600324286 and the following scores: AIC of 630; Design Size of 9753; Logical Depth of 14365; Sophistication of 90; 529 build symbols; a Grammar Size of 11; a Connectivity of 58; 1668 branches; and a height of 168. Its MRH scores are: $\mathtt{M}_p$ is 20, $\mathtt{M}_d$ is 2202; $\mathtt{R}_a$ is 15.5; $\mathtt{R}_b$ is 18.4; $\mathtt{R}_m$ is 110.1 and it has an H of 9. While these scores give examples of the differences that can arise, a better overall picture is gained from looking at the average values from a number of evolutionary runs on different sizes of the design problem.

Table 8-1 lists the average values over 15 trials of the various measures as applied to the best tables evolved on different sizes of the design problem ($20 \times 20 \times 20$, $40 \times 40 \times 40$, $60 \times 60 \times 60$, and $80 \times 80 \times 80$). Most of the existing measures of complexity and "control" measures – Algorithmic In-

formation Content, Sophistication, Number of Build Symbols, Grammar Size, Connectivity, Number of Branch Nodes and Height – do not scale monotonically as the design problem is scaled up. Aside from the MRH measures, the two complexity measures that do increase along wth the scaling of the design problem are Design Size and Logical Depth. While this positive correlation suggests that Design Size and Logical Depth are good measures of complexity, the guidance they provide is not very useful for creating a better evolutionary design system: add more parts/processing to a design.

Of the MRH and structure and organization measures, $\mathtt{M}_d$, $\mathtt{R}_m$, H and both measures of structure and organization scale up along with the scaling of the design problem. Since only modular reuse ($\mathtt{R}_m$) scales along with the size of the design space, this suggests that the type of reuse that is useful in evolutionary design is not overall reuse ($\mathtt{R}_a$) or reuse of build symbols ($\mathtt{R}_b$), but the reuse of modules. By extension, this also suggests that those design representations which do not have the ability to hierarchically assemble and reuse modules will not scale as well.

Each of the MRH metrics measure different aspects of the structure and organization of an object. Of interest is combining the scores of these three metrics into a measure of structure and organization with a single value, for which we have proposed two measures. Both measures are products of the three MRH measures, with SO_1 normalized for size by dividing by the amount of information in the object and SO_2 normalized for size by dividing by the size of the object. Both measures scale up with the increase in the size of the problem so both are reasonable measures of structure and organization. We leave it to future work to determine which of these two measures is better.

6. Advantages of Enabling MR&H

Having identified the characteristics of MR&H as being well correlated with more sophisticated designs, we now demonstrate that evolution using representations with MR&H will scale better, and find higher fitness designs than evolution using representations without MR&H.

Table 8-2 contains the results of 15 trials with each of the five representations (none, M, MH, MR and MRH) on four different sizes of the table design problem. Each entry in the table shows the average of these 15 trials of the best individual found after one million evaluations. Using a two-tailed Mann-Whitney test, the differences in performance are highly significant, $P < 0.001$, for problem sizes 40x40x40 and larger – with the exception that there is no significant difference between *none* and *M* ($P > 0.05$) on the 40x40x40 sized problem. The best performance is achieved in a representation with more of the features of MR&H enabled: representation MR&H is always best, representation MR is always second best, and representation M always outperforms *none*.

Table 8-2. Averaged best fitness, after 15 trials, of the five different representations on different sizes of the table design problem.

problem size	(fitness $\times 10^5$) none	M	MH	MR	MRH
20x20x20	3.08	3.06	3.84	5.35	5.64
40x40x40	59.7	58.6	79.6	131	181
60x60x60	222	167	366	762	1227
80x80x80	413	266	773	3082	4402

These results support the claim that enabling MR&H improves the performance and scalability of evolutionary design systems.

An intuitive understanding of the differences between evolution with MR&H enabled and not enabled can be gained by viewing the structure of the assembly procedures that are generated by the evolved design programs and the resulting tables. The evolved design program (genotype) and the table it constructs for the best individual evolved with representation `None`, are shown in Figure 8-3c and Figure 8-2c. This table has randomly scattered holes on its surface and none of the legs on its corners go down more than a few levels. Since none of the characteristics of MR&H are used in the design encoding, its modularity, reuse and hierarchy values are all 1.0 and this lack of structure and organization can be seen in the randomness of its assembly procedure–which is the same as its genotype. In contrast, the table evolved with representation `MRH` has a fully filled surface, and the structure and organization of its encoding can be seen in the complex, multi-level patterns in the assembly procedure generated by its evolved design program, shown in Figure 8-3b.

Enabling all three components of MR&H not only results in assembly procedures and designs with more structure and organization, but also enables the search algorithm to reach parts of the design space that would not otherwise be explored. Figure 8-4 contains plots of size (number of nodes) versus depth of the assembly procedures of individuals evolved with the different representations. For these plots the best individual from each layer for every five generations of all the evolutionary runs was used. Included with these graphs are lines indicating the minimum and maximum number of nodes that can be in a tree of a given depth.

Starting with representation `M`, as more elements of MR&H are added to the representation, the plots in Figure 8-4 show that a greater size–depth spread is achieved. This increase in size–depth spread as more of the elements of MR&H are enabled is matched by better evolutionary performance. Similarly, the size–depth coverage of representation `None` is larger than that of `M`, and this corresponds to better evolutionary performance with representation `None`.

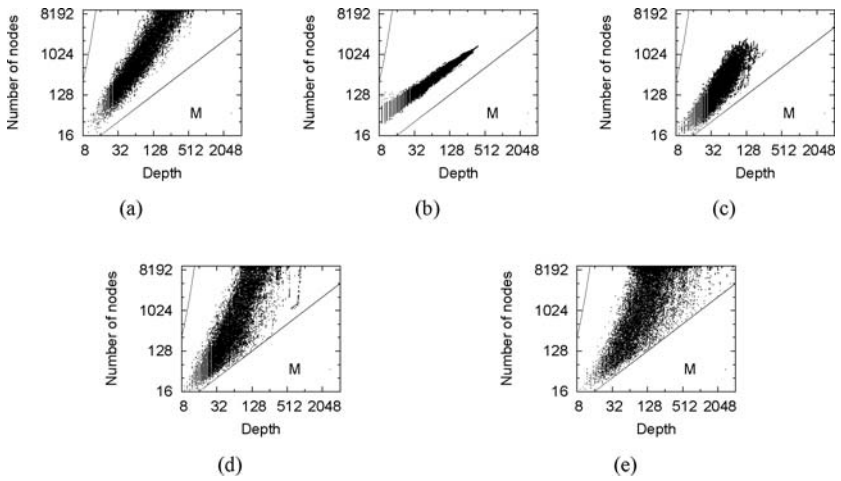


Figure 8-4. This figure contains size-depth plots for the best individuals produced in the evolutionary runs with representation: (a) `None`; (b) `M`; (c) `MH`; (d) `MR`; (e) `MRH`.

One anomaly is that the size–depth coverage is greater with `None` than with `MH`, yet evolution with `MH` has better fitness scores. An explanation for this may be related to differences in the types of assembly procedures that are produced with these two representations: individuals evolved with `MH` tend to have fewer nodes and also tend to be deeper for a given number of nodes than those evolved with `None`.

7. Conclusion

The designs that we can achieve are limited only by our imagination and the tools we use. Similarly the designs that evolutionary design systems can achieve are limited only by the representations with which they operate. In this chapter we have argued that to improve the scalability of evolutionary design systems we need modularity, reuse and hierarchy (MR&H) – characteristics found in both man-made and natural designs. Furthermore, these three characteristics are enabled not by the fitness function or the search algorithm, but by attributes of the representational language of the design encoding. We borrowed from the field of computer programming languages to identify three attributes of design representations: combination, control-flow and abstraction – and claimed that these attributes enable MR&H in evolved designs and used them to define metrics of `MRH`.

To support our argument that MR&H are the design characteristics that must be enabled to improve evolutionary performance and scalability we performed two sets of experiments. First, we first compared our metrics of `MRH` against

existing complexity measures on a scalable design problem, and found that the measures of modularity in the design ($\mathtt{M}_d$), reuse of modules ($\mathtt{R}_m$), hierarchy ($\mathtt{H}$), and our two measures of structure and organization to be more useful measures of design complexity than the others. Next, we compared five representations with different combinations of the characteristics of MR&H enabled on different sizes of our design problem and found that performance and scalability improved with more of MR&H enabled.

To summarize: while existing complexity measures may measure different aspects of a design, it is not so much the amount of information or processing that is important in determining complexity, rather it is how this information is structured and organized. By implementing increasingly more powerful representations which hierarchically encode reusable modules, future evolutionary systems will be better able to produce ever more sophisticated designs.

References

Abelson, H., Sussman, G. J., and Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. McGraw-Hill, second edition.

Bennett, C. H. (1986). On the nature and origin of complexity in discrete, homogenous, locally-interacting systems. *Foundations of Physics*, 16:585–592.

Bentley, P. J. (1999). *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco.

Bentley, Peter J. and Corne, David W. (2001). An introduction to creative evolutionary systems. In Bentley, Peter J. and Corne, David W., editors, *Creative Evolutionary Systems*, pages 1–75. Morgan Kaufmann.

Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *Journal of the Association of Computing Machinery*, 13:547–569.

Edmunds, B. (1999). *Syntactic Measures of Complexity*. PhD thesis, Dept. of Philosophy, University of Manchester.

Goldwasser, M., Latombe, J.-C., and Motwani, R. (1996). Complexity measures for assembly sequences. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 1581–1587, Minneapolis, MN.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Hornby, G. S. (2006). ALPS: The age-layered population structure for reducing the problem of premature convergence. In et al., M. Keijzer, editor, *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2006*, pages 815–822, New York, NY. ACM Press.

Hornby, Gregory S. (2004). Functional scalability through generative representations: the evolution of table designs. *Environment and Planning B: Planning and Design*, 31(4):569–587.

Hornby, Gregory S. and Pollack, Jordan B. (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8:223–246.

Hornby, Gregory Scott (2003). *Generative Representations for Evolutionary Design Automation*. PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA.

Huang, C. C. and Kusiak, A. (1998). Modularity in design of products and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 28(1):66–77.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–17.

Koppel, M. (1987). Complexity, depth and sophistication. *Complex Systems*, 1:1087–1091.

Meyer, B. (1988). *Object-oriented Software Construction*. Prentice Hall, New York.

Rosen, B. K. (1974). Syntactic complexity. *Information and Control*, 24:305–335.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control*, 7:1–22,224–254.

Ulrich, K. and Tung, K. (1991). Fundamentals of product modularity. In *Proc. of ASME Winter Annual Meeting Symposium on Design and Manufacturing Integration*, pages 73–79.

Yngve, V. H. (1960). A model and an hypothesis for language structure. In *Proceedings of the American Philosophical Society*, pages 444–466.

Chapter 9

PROGRAM STRUCTURE-FITNESS DISCONNECT AND ITS IMPACT ON EVOLUTION IN GENETIC PROGRAMMING

A.A. Almal[1], C.D. MacLean[1] and W.P. Worzel[1]
[1]*Genetics Squared Inc., Ann Arbor.*

Abstract Simple Genetic Programming (GP) is generally considered to lack the strong separation between genotype and phenotype found in natural evolution. In many cases, the genotype and the phenotype are considered identical in GP since the program representation does not undergo any modification prior to its encounter with "environment" in the form of inputs and a fitness function. However, this view overlooks a key fact: fitness in GP is determined without reference to the makeup of the individual programs but evolutionary changes occur in the structure and content of the individual without reference to its fitness. This creates a disconnect between "genetic recombination" and fitness similar to that in nature that can create unexpected effects during the evolution of a population and suggests an important dynamic that has not been thoroughly considered by the GP community. This paper describes some of the observed effects of this disconnect and studies some approaches for the estimating diversity of a population which could lead to a new way of modeling the dynamics of GP. We also speculate on the similarity of these effects and some recently studied aspects of natural evolution.

Keywords: phenotype, genotype, evolutionary dynamics, GP structure, GP content, speciation, population, fitness

1. Introduction

In comparison to natural genetics and evolution, genetic programming (GP) is a crude mechanism. In a simple GP system, the phenotype is seemingly the same as the genotype with the individual program applied to a problem being identical to the entity that is involved in such operations as crossover and mutation. There are no equivalents to the natural mechanisms of transcription or translation,

let alone such complexities as embryonic development and maturation of the phenotype.

Moreover, in most GP applications, fitness is determined in a static environment as represented by a single, unchanging fitness function, and the data it learns from is often retrospective and static. Finally, the genetic operation of crossover is quite disruptive, having no concept of particulate genes that are exchanged intact. Crossover in GP is more like a genetic translocation in nature wherein a piece of genetic material is randomly swapped between the strands of DNA in the potential zygote, which almost inevitably leads to death of the embryo.

In addition, the conventional view of evolution in genetic programming populations is that a superior individual propagates fairly quickly throughout a population, creating–after a period of diffusion–a much less diverse population (Poli and Langdon, 1997). Although such mechanisms as demes, tournament selection and geographies slow this progression, as long as there is free communication between all members of the population, the overall population is expected to reach a relatively stable state where the comparative diversity is relatively small, until the next major mutational event creates a notable change in the fitness of members of the population.

In this paper we discuss the nature of genotypes and phenotypes in GP and describe a measure of informational entropy for following diversity in the evolution of a population. We then discuss a way to display the content and fitness of GP individuals in order to investigate how sub-populations may arise in GP. Finally, we consider the differences and similarities of evolution in GP to natural evolution. In this paper we will discuss the nature of genotype and phenotype in GP and discuss GP crossover and mutation effects.

2. Background

(Daida, 2003) and (Almal, 2005) showed that the structure and content of an entire population could be imaged creating a dynamic animation of its progression through many generations. Daida used this to show that most genotypic changes occur toward the leaves of the program tree and that this creates structural limitations in the space that could be effectively searched, even within the space of all possible structures. Figure 9-1 shows one of the structural visualizations of a population from (Daida, 2003) where the larger the number of individuals that share a structure within a program tree, the darker the branch. The reader is urged to review this work and, if possible, the animations that show the evolution of simple GP systems, as they reveal much about the mechanisms of GP.

(Almal, 2005) showed that the content of program trees could be could be visualized, along with the structure, by using a technique developed for

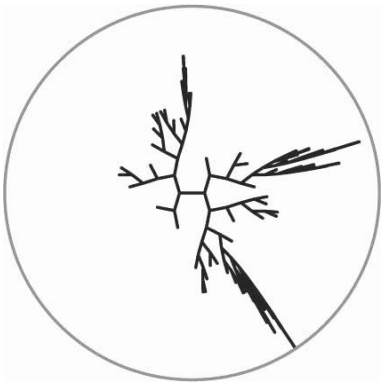


Figure 9-1. Representation of structures used in a population from a overlay of tree structures created by Daida from (Daida, 2003).

visualizing DNA sequences. The method of visualizing DNA is done by plotting a sequence of bases in order from the center of a square to the corner of the squared labeled with the next DNA base in the sequence where each successive line segment is plotted from the end of the previous segment and travels $1/(n+1)$ of the distance where n is the number of steps taken since starting at the beginning of a sequence (Jeffrey, 1990).

(Almal, 2005) modified this by plotting all possible variables and operators used in a GP tree on a circle and then used the same mechanism of plotting line segments as the program tree is traversed. Each branch of the tree that resolved to a terminal is plotted separately, creating a set of endpoints for each branch. This appears as a set of ovals around a common center. Figure 9-2 shows an example of such a population representation.

Each cluster of endpoints may be viewed as a different path through structure-content space and collectively they may be viewed as a population in the statistical sense, so that statistical metrics of the dispersion of the endpoints can be used to evaluate the diversity of the population. Using this idea, we have begun to calculate the informational entropy of GP populations. Conceptually, the smaller the distribution of the endpoints, the more uniform the population and the less informational entropy there is in the population. However, because individuals are not represented by a single point (due to the various branches in the program tree), we must first find the "centroid" or a similar representation for each program, and demonstrate that different trees cannot have the same centroid, before we can look at the distribution of endpoints in a population. We are exploring this, and other measures of structural diversity within a population, as a means to assess the convergence and possibly control the evolutionary process in GP.

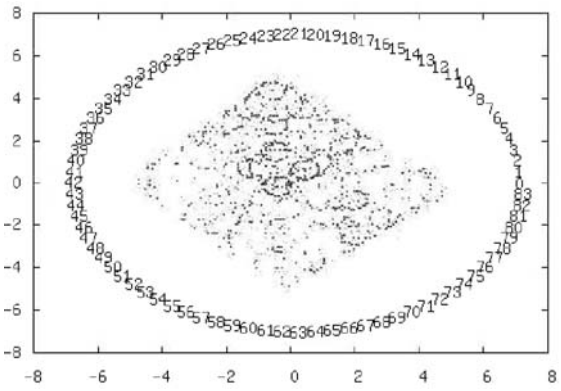


Figure 9-2. Plot of structure and content using modified chaos game method to represent an entire population from (Almal, 2005).

For the moment, we have confined ourselves to calculating the "fitness diversity" over time within a population. The measure of diversity that we use is based on entropy. Entropy can be used very succinctly in estimating the phenotypic diversity (Rosca, 1995); however there have been studies using entropy based structural diversity measures (Ekárt and Németh, 2002).

In a GP paradigm the phenotype is represented by the fitness, and so we use the fitness entropy for each generation to ascertain the diversity in the population along the evolutionary timeline.

The entropy H_p in a population P, where f_j is the fraction $\frac{n_j}{N}$ of individuals, P having fitness j and N is the number of fitness values shown in Equation 9.1. This is equivalent to the fitness diversity in a GP population.

$$H_p(P) = -\sum_{j=1}^{N} f_j log(f_j) \tag{9.1}$$

Entropy provides us with a quantitative entropy measure that facilitates studying the phenotypic diversity during the evolutionary process. Entropy represents the amount of disorder of the population (Rosca, 1995), thus low entropy means low diversity. However, since the phenotypic measure compares the number of different fitness values, it could be interpreted as the number of groups having the same fitness value. Thus high entropy could be considered as the presence in the population of a high number of small groups of individuals, each group having the same fitness value, while low entropy would mean a low number of large groups of individuals. Of course it is also possible that there are a number of different individuals with the same fitness.

It is interesting to follow the entropy plots along the evolutionary timeline. In most plots we can observe the entropy in general increases in the system followed by a gradual reduction. This seems to concur with the belief that in a GP paradigm, evolution can be separated in two different phases: exploratory (indicated by an increase in entropy) and what might be called "selectionary" (indicated by a decrease in entropy) where the GP system works to exploit a small number of individuals by varying their structure. Figure 9-3 is a plot of the fitness diversity calculated using Equation 9.1 for a single GP run over a relatively small number of generations.

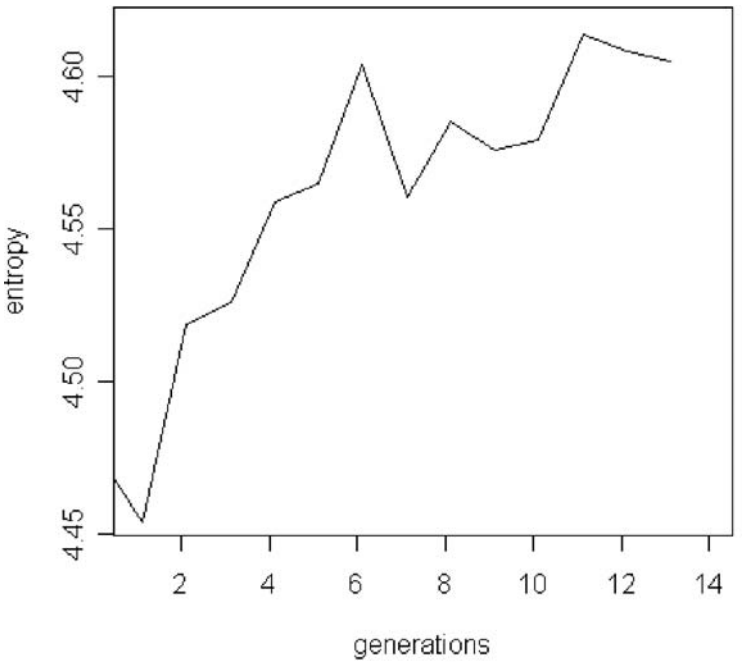


Figure 9-3. Fitness entropy over time as calculated using Equation 9.1.

This figure clearly shows a significant variation in diversity as evolution continues during this run. It is interesting that at the end of the run, the fitness entropy, and presumably the diversity, has continued to rise after falling at generation 7. This suggests it has not converged on a single set of features and structure for the problem at hand.

Figures 9-4 to Figure 9-8 show heat maps of the structure-content plots of the population.[1]. The increasing entropy plot in Figure 9-3 shows that there were a

[1]The fitness is normally displayed by coloring each of the structure-content endpoints, according to fitness where the better fitness individuals have a "hotter" color than others.

greater number of solutions explored progressively over these generations. The structure-content-fitness heat map plots show that there was quite a bit of difference in these individuals with the population settling in the upper left corner of the plots as being the most fit region for study. The transition from generation 7, shown in Figure 9-6, where the fitness diversity has dropped, to generation 11, shown in Figure 9-7, where the fitness diversity has increased, shows an apparent increase in the structure-content diversity based on comparing the distribution of endpoints in the heat map in Figure 9-6 with those in Figure 9-7 and a corresponding increase in the number of neighborhoods with high fitnesses. This suggests that fitness diversity could be used as a good approximation of structural diversity. We are working on a measure for quantifying structural diversity, using information theoretic approaches.

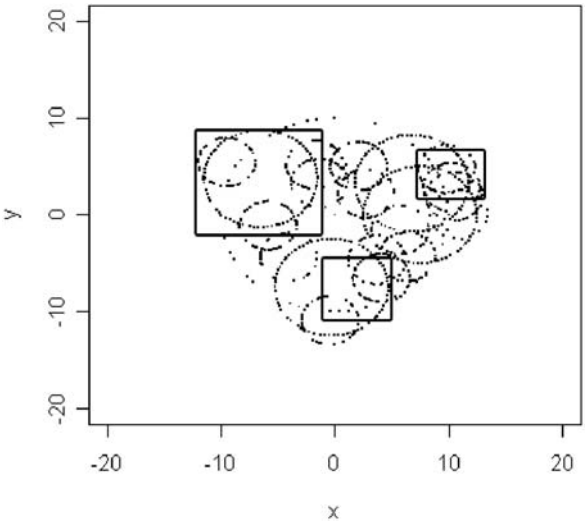


Figure 9-4. Generation1 structure-content-fitness heat map.

Our intention is to create data on runs with more generations than these early examples as the evolutionary process has not progressed yet to a point where a clear focus has emerged. We also plan to explore structure-content entropy measures further to see what correspondence there is with fitness entropy.

3. The Nature of Genotype and Phenotype in GP

Programs or "genotypes" in GP are usually created by assembling a random selection of possible variables and operators. There is sometimes bias for, or against, certain terminal elements, but in general a "generation 0" population is the most diverse population of an entire GP run because there has been no selection of individuals to remove combinations from the population.

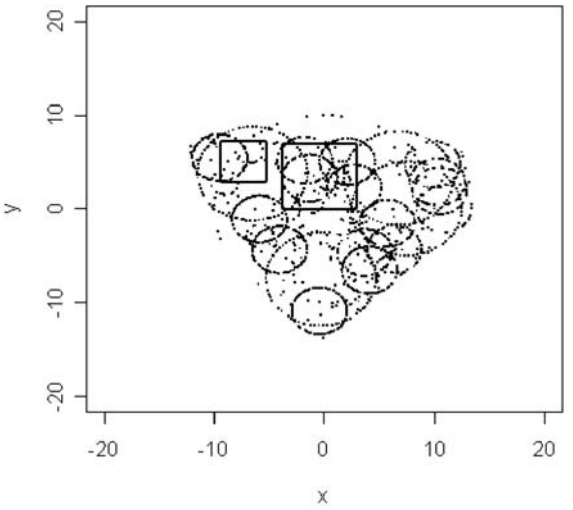


Figure 9-5. Generation6 structure-content-fitness heat map.

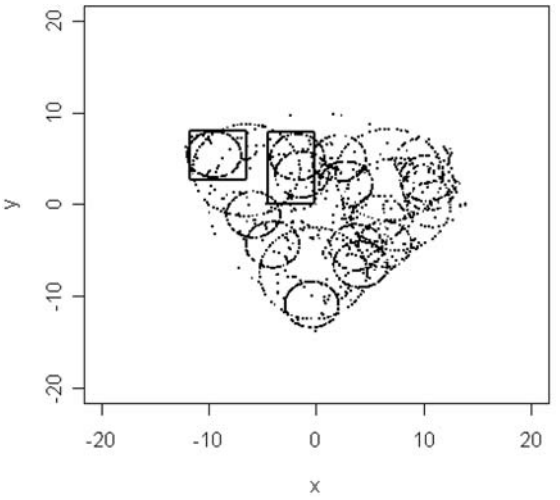


Figure 9-6. Generation7 structure-content-fitness heat map.

However, it is important to remember that not all possible combinations create viable individuals. For instance, an individual that contains combinations such as "+ * *" is not viable and will either be prevented or removed by syntax driven selection, repair, or in the worst case, negative selection by assignment of a poor fitness. (Yu and Bentley, 1998) outlines all the methods used to remove

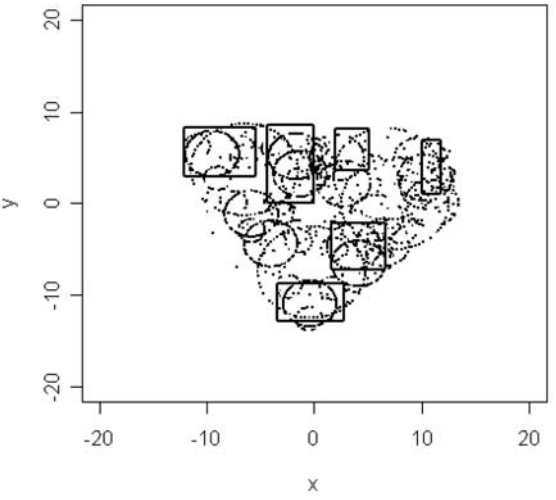


Figure 9-7. Generation11 structure-content-fitness heat map.

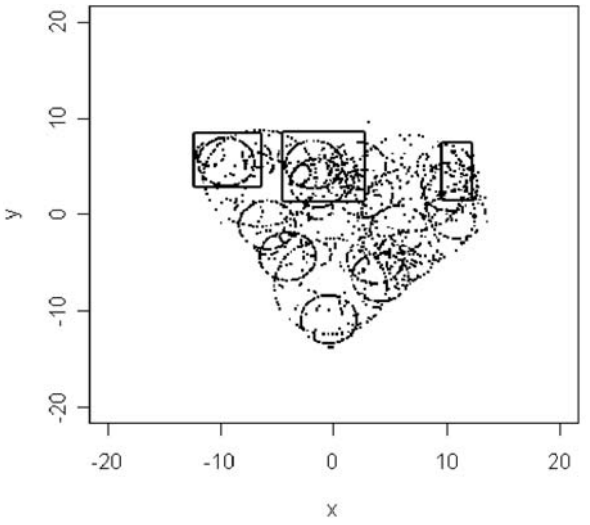


Figure 9-8. Generation13 structure-content-fitness heat map.

deleterious individuals due to bad composition. The fact that individuals are removed at some point, or even several points in the process of creation, as well as during crossover and mutation, has a correspondence to natural processes such as selection during embryonic development, DNA repair and even human

interventions such as gene therapy. What this suggests is that in genetic programming, as in nature, not all genetic paths are allowable–and that this affects the probabilities associated with evolutionary pathways.

Moreover, as Daida has suggested in (Daida, 2003), even within the set of allowable evolutionary paths, not all are equally likely to be selected because of the tendency of GP to weight crossover and mutation toward points lower in the program trees, leaving the roots mostly untouched beyond the early stages of evolution.

Taken together, these factors suggest that not all evolutionary paths are equally likely, and that there some paths are unlikely to be retraced. For example if, through chance, crossover occurs high in a program tree, it is unlikely that the inverse crossover will occur as the joint probability of two high-tree crossovers occurring is increasingly small as the typical tree size grows.

All of the GP operations involving population creation, crossover, and mutation occur without reference to fitness–they involve only structural changes to the program tree. This corresponds loosely to genetic alterations of an individual genotype in nature, which suggests that even without conversion of GP "DNA" to "proteins" through transcription and translation, there is still a de facto phenotype even though the phenotype is the same "physical" entity as the genotype.

What, then, is a phenotype in GP? On paper it is the same entity as the genotype: the program representation. However, in general, an individual's fitness is not determined by structure or content–it is determined by evaluating its fitness in some way using a particular set of inputs or running the program in a specific environment. In other words, it is the application of a genetically derived program representation to an environment in the form of a set of input data or its injection into a simulation, from which a fitness is calculated. Even though the fitness measure and environment used to test the system may be different for different problems, the phenotype is always defined by the individual's encounter with these factors. One way of describing a GP phenotype, conceptually, is that it is an attempt to exploit the resources available to it, mostly in the form of "consuming" data to "produce" a fitness which leads to survival of the individual.

Whether a GP individual has an embryonic stage or not, whether it "grows" after it is "born" (as has been done in various applications, particularly in real world engineering designs), the characteristics of structure and content manipulation define the GP genotype, while test data and fitness calculation define the phenotype.

4. Implications of GP Genotype–Phenotype Definition

While constraints on the structure and content of a GP genotype control its path in structure–content space, the selection of individuals for recombination is controlled by fitness. Importantly, the fitness is not directly connected to changes to structure-content made by the genetic operations of crossover and mutation. The impact of structural changes on fitness are not known at the time the structure and content changes are made, while the impact of fitness is only related to which individuals are combined during crossover. Changes to the genotype are not directly driven by the fitness of the individual, instead, genotypic changes are driven indirectly by later comparison of the fitness of a new individual created during crossover, when compared with some or all of the other members in the population. In other words, a GP system does not make decisions on crossover or mutation based on the fitness of the individual, and similarly, fitness is not determined by the history of genotypic alterations but by the testing of the individual against an environment. This highlights the difference between the GP genotype and the GP phenotype even though, unlike many biological systems, there is no visible difference between the two. Thus the path through fitness space is not explicitly related to the mechanism of change in the genotype.

It is worth noting that the only exception to this is when the fitness is explicitly connected to the size, shape or content of the individuals in a GP population. For example, many parsimony rules reduce the fitness of large, complex individuals. Similarly, it is not unusual to bias the fitness of an individual if there are certain content elements that are more or less desirable than others. There are even fitness measures in extreme cases, such as the LiD function described in (Daida, 2004), where fitness is determined by the size and shape of the structure in order to show how structures evolve, but these cases only influence the selection of individuals by changing their fitness according to their structure, not by the mechanisms of recombination and mutation.

Given that there is a clear separation of mechanisms, what are the implications of this separation in a population as it evolves? By using the structure-content plots, and adding a heat map for fitness, we can examine the evolution of different structures with roughly similar fitness in a single population. We have begun to search for this behavior within populations and have seen some early cases where similar fitness occurs in very different parts of the structure-content space. We have yet to determine the longevity of these occurrences and must still show that the distance between the endpoints in Figure 9-2 are truly definitive in characterizing differences between individuals, but we believe, based on preliminary analysis, that it is possible for distinct sub-populations to emerge and to coexist within a population over a long (in generational terms) periods of time, effectively exploring different parts of the structure–content–

fitness space. This is contrary to the previously stated belief that diversity within populations tends to narrow irrevocably over time.

Because the crossover operator in GP tends to operate as a macro-mutation operator, it can create significant changes in the structure and content of an individual program in a GP population. This can lead to the sudden appearance of two distinct sub-populations. Occasionally, whether by accident (Angeline, 1997) or design (Ryan et al., 2004), the average fitness of these populations may be roughly equivalent, so that they endure despite their differences. If a distinctly different individual appears because of a crossover event that occurs high up in a tree, then when this individual is crossed with another individual in the population, at least one of offspring will tend to resemble the different individual more than it does the other individual's immediate ancestors in the sense that the offspring with tend to be located closer to the distinct individual in structure-content space than do the other members of the population. Given roughly equivalent fitness in these two groups, this new population will not disappear back into the main population as the change that created it is unlikely to be reversed since the probability of having a second crossover high in the tree is very small. The only way such offspring are likely to disappear is if the structure-content neighborhood they have moved to is not as "fitness rich" as the neighborhood they came from. One approach under consideration is to create a test case where there are two or more distinctly different solutions to a problem. Repeated GP runs where the structure-content-fitness heat maps are created should demonstrate clearly whether distinct sub-populations are created and endure over generations.

Population mechanisms such as tournament selection (Koza, 1992), demes (Iwashita and Iba, 2002), and "trivial" geography (Spector and Klein, 2005) will increase this tendency by allowing a pattern to be fixed in a sub-population and reducing the chance of dilution that occurs in a larger population, but such sub-populations may not be necessary for a sub-population to be created and once created, to persist. Once a distinctly different individual occurs through a high-tree crossover event, it begins to take on some of the attributes of species in a natural environment.

If two distinct sub-populations with similar fitnesses among their respective individuals occur, then they can get into a arms-race where one sub-population's size increases slightly, reducing the "resources" of available places in the overall population. This is then followed by the other sub-population gaining an advantage by additional refinement of its fitness and wins back the space in the population. Eventually however, this behavior can collapse as one sub-population gains a clear upper hand and drives the other sub-population out of existence, leading to a newly stable system. This is again reminiscent of natural species that are in competition with one another.

However, it remains to be seen whether such differences in structure and content, as mapped by our method, signifies a true difference in individuals or whether variations in structure that lead to apparently different structures and content are truly different or not. For instance, the program A+ (B + C) is identical in fitness to C + (A + B) though it maps to a distinctly different location. Do apparent sub-populations actually map to the same result? Similarly, even though they have similar but different content, is A + (B + C + D) really different when compared to A + (B + C) if D is 0 or very small?

5. Is there a relationship between GP behavior and natural selection?

Walter Fontana, in "The Topology of the Possible" (Fontana, 2003) described the exploration of different phenotypes, as represented by transcribed RNA structures, and structural changes at key decision points in the variation of DNA sequences. He focuses on the mechanisms of change within a genotype that lead to a differentiation in the corresponding phenotype. Fontana studied whether all DNA sequences that lead to a transcription coding for the same protein are co-equal in evolutionary terms and concludes that, in fact, from an evolutionary perspective, they are quite different, as they may be on very different paths that are more or less likely to lead to structural changes in a phenotype. He makes the argument that some sequence changes can lead in directions where change is more likely in one direction than it is in the other and that this creates a gradient such that evolution is more likely in one direction than in the other.

GP is not quite the same, as a macro-mutation from crossover with a significant variance in structure is unlikely to occur. But if a significant structural change does occur, and if it has equivalent fitness to the individuals the original population, it is unlikely that it will be go back to the structure as the original population. The point is that significant changes that occur near the root node are very unlikely to recur and even changes that are not near the root but are noticeably higher than the terminal leaf nodes may survive long enough to create a fixed sub-population.

Fontana also describes protein folding constraints in RNA that mediate exploration of natural systems, while in GP (Daida, 2003) and (Almal, 2005) suggest that exploration occurs at certain points as constrained by both syntactic and structural behaviors of crossover in GP. Simply put, in both systems some combinations are very unlikely or unreachable while others are much easier to reach. The difference is that, in RNA, there is an unequal gradient to movement in both directions while in GP, movement shares an equal probability.

At a macrobiology level, the disconnect between genotype and phenotype suggests that if significant structural changes are possible in the genotype, then

evolutionary changes are possible without an environmental change. There can be changes that simply occur through random exploration that provide an equivalent fitness without competing with the original phenotypes. A possible example of this is the recently discovered genotypic and phenotypic emergence of a new sub-species of the Darwin finch on the Galapagos Islands within the same range as the original species (Huber, 2006) without any obvious ecological change that would put pressure on the Darwin finches to change. The changes to beak structure in the new sub-species allow them to feed on different plant seeds and therefore do not put them into direct competition for resources with the original species. Indeed, the original species remains and flourishes while the emergent sub-species grow in numbers. This phenotypic change correlates with a specific variation in the DNA of the finches. One could consider such a change to be setting the stage for speciation if there was a future environmental change that led to further changes in the new sub-species away from the originating species. It is a marvelous twist of history that such a case should emerge in Darwin finches, one of the key species Charles Darwin used to make his case for natural selection (Darwin, 1859).

This leads to further questions about the similarities and differences in natural and artificial evolutionary systems. For example, given the energy cost of creating individuals that do not develop successfully through the embryonic stage, is selection more conservative when compared to the relatively trivial cost of evaluating a new phenotype in an artificial evolutionary system? If so, this would suggest that as artificially evolutionary systems are applied to more complex problems, where the costs to evaluate fitness increases, a more conservative mechanism for genotypic change may be called for, perhaps making use of phenotypic plasticity and related processes to generate novel but non-lethal offspring (Kirschner and Gerhart, 2005).

Conversely, the comparative cheapness of fitness evaluation in simpler problems may suggest that the greater degree of exploration allowed in artificial evolutionary systems is effort well spent. MacArthur and Wilson's discussion (MacArthur and Wilson, 1967) of r-selection and K-selection, though originally defined in terms of ecosystems, may also be informative of the differences between structural exploration and developmental costs.

Similarly, does diploidy encourage structural change "testing" at a lower evolutionary cost in nature since some individuals with a bad allele on one strand of DNA can survive while others with a double allele cannot? There are many cases suggestive of this such as sickle cell anemia, cystic fibrosis and Tay-Sachs disease where a single allele confers an evolutionary advantage in the form of resistance to a disease, while the double allele is deadly. One could imagine a structural change in DNA leading to this sort of occurrence with the diploidal relationship conferring some protection from dangerous mutations. Monoploidy would also have the advantage of allowing easier testing

of mutations in a highly competitive environment with an immediate payoff for success.

Similarly prokaryotes, with their lower cost of testing structural changes (from an evolutionary perspective) may evolve more aggressively since populations are large, generations are short, and there is no embryo or separate egg-sperm development to support.

Finally, can the lessons learned from watching the dynamics of GP behavior using the modified Chaos Game approach described in (Almal, 2005) be turned back to the study of structural differences in DNA in nature? Rapid whole genome mapping is becoming feasible (Wicker et al., 2006) and it may be possible, to watch structural changes in the DNA of creatures with short generational timeframes, to appear and disappear or become fixed in the genome.

6. Summary and Future Directions

Our notions of the evolutionary mechanisms of even "simple" GP may need to be reconsidered. The Building Block Hypothesis and the Schema Theorem are just the start to understanding a complex dynamic. More investigation needs to be done on the use of tool such as Daida's population structure displays and the modified Chaos Game structure-content-fitness heat maps described here. In particular more classes of problems need to be studied using these techniques, and the effect of changing genetic operators and parameters on the structure and content of populations should be studied.

We expect that it may be possible to "tune" the genetic parameters based on this view of population dynamics where it is possible to characterize the cost of exploration in terms of fitness changes within a population and to visualize the effect of such changes on both a population's diversity and the overall fitness of the individuals in it. If, for example, an increased crossover rate leads to greater structural diversity but without a corresponding increase in fitness, it may be that there's a need to decrease the crossover rate to exploit the local search characteristics of mutation. Conversely, a relatively uniform population in terms of fitness and structural diversity may suggest that the problem may require an increase in the crossover rate.

Finally, while it has been suggested that GP and similar, simple evolutionary systems do not really reveal much about natural selection, the convergence of our ideas with Fontana's, despite our being unaware of his work at the time, suggests that that even a simple evolutionary system can have some of the same behavioral complexities as a natural system at the genotypic level. From an evolutionary perspective, a phenotype, no matter how abstracted, is what interacts with the environment.

7. Acknowledgements

The authors would like to thank Rick Riolo for his trenchant comments on evolutionary algorithms, biology and ecology and would like to thank Jason Daida for permission to reproduce one of the images from his 2003 paper on structural development in GP populations.

References

Angeline, Peter J. (1997). Subtree crossover: Building block engine or macromutation? In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA. Morgan Kaufmann.

Daida, Jason M. (2003). What makes a problem GP-hard? A look at how structure affects content. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice*, chapter 7, pages 99–118. Kluwer.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Racs in the Struggle for Life*. Murray, London, UK.

Ekárt, Anikó and Németh, Sandor Zoltan (2002). Maintaining the diversity of genetic programs. In Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 162–171, Kinsale, Ireland. Springer-Verlag.

Fontana, W. (2003). The topology of the possible. Working paper 03-03-017, The Santa Fe Institute, Santa Fe.

Huber, S.K. (2006). Premating isolation of sympatric morphs in a population of drawin's finches (geospiza fortis). In *Animal Behavior Society 43rd Annual Meeting*, Snowbird, Utah.

Iwashita, Makoto and Iba, Hitoshi (2002). Island model GP with immigrants aging and depth-dependent crossover. In Fogel, David B., El-Sharkawi, Mohamed A., Yao, Xin, Greenwood, Garry, Iba, Hitoshi, Marrow, Paul, and Shackleton, Mark, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 267–272. IEEE Press.

Jeffrey, H.J. (1990). Choas game representation of gene structure. *Nucleic Acids Res*, 18(8):2163–70.

Kirschner, Marc W. and Gerhart, John C. (2005). *The Plausibility of Life: Resolving Darwin's Dilemma*. Yale University Press.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

MacArthur, R. and Wilson, E.O. (1967). *The Theory of Island Biogeography*. Princeton University Press.

Poli, Riccardo and Langdon, W. B. (1997). An experimental analysis of schema creation, propagation and disruption in genetic programming. In Back, Thomas, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 18–25, Michigan State University, East Lansing, MI, USA. Morgan Kaufmann.

Rosca, Justinian P. (1995). Entropy-driven adaptive representation. In Rosca, Justinian P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, Tahoe City, California, USA.

Ryan, Conor, Majeed, Hammad, and Azad, Atif (2004). A competitive building block hypothesis. In Deb, Kalyanmoy, Poli, Riccardo, Banzhaf, Wolfgang, Beyer, Hans-Georg, Burke, Edmund, Darwen, Paul, Dasgupta, Dipankar, Floreano, Dario, Foster, James, Harman, Mark, Holland, Owen, Lanzi, Pier Luca, Spector, Lee, Tettamanzi, Andrea, Thierens, Dirk, and Tyrrell, Andy, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 654–665, Seattle, WA, USA. Springer-Verlag.

Spector, Lee and Klein, Jon (2005). Trivial geography in genetic programming. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 8, pages 109–123. Springer, Ann Arbor.

Wicker, Thomas, Schlagenhauf, Edith, Graner, Andreas, Close, Timothy, Keller, Beat, and Stein, Nils (2006). Published online.

Yu, Tina and Bentley, Peter (1998). Methods to evolve legal phenotypes. In Eiben, Agoston E., Back, Thomas, Schoenauer, Marc, and Schwefel, Hans-Paul, editors, *Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 280–291, Amsterdam. Springer-Verlag.

Chapter 10

GENETIC PROGRAMMING WITH REUSE OF KNOWN DESIGNS FOR INDUSTRIALLY SCALABLE, NOVEL CIRCUIT DESIGN

Trent McConaghy[1], Pieter Palmers[1], Georges Gielen[1], and Michiel Steyaert [1]
[1] *Katholieke Universiteit Leuven, Leuven, Belgium*

Abstract This paper shows how aggressive reuse of known designs brings orders-of-magnitude reduction in computational effort, and simultaneously resolves trust issues for synthesized designs, for genetic programming applied to automated structural design. Furthermore, it uses trustworthiness tradeoffs to handle addition of novelty in a trackable fashion. It uses a multi-objective algorithm with an age-layered population structure to avoid premature convergence. While the application here is analog circuit design , the methodology is general enough for many other problem domains.

Keywords: synthesis, industrial, analog, integrated circuits, CAD

1. Introduction

Background: GP for Automated Structural Design

A core reason that genetic programming (GP) (Koza, 1992) is interesting is its natural ability to handle search spaces with tree-like and graph-like structures (topologies), which makes it a natural fit for automated invention of structures. One focus has been design of analog circuit topologies, such as those in (Koza et al., 1999; Koza et al., 2003a; Koza et al., 2004; Hu and Goodman, 2004; Lohn and Colombano, 1998; Shibata et al., 2002; Sripramong and Toumazou, 2002; Dastidar and Chakrabarti, 2005). In this domain, GP has evolved several patent-quality circuits (Koza et al., 2003a) essentially "from scratch", which is a remarkable success by almost any measure. It is an especially notable accomplishment from an artificial intelligence perspective because "patent-worthiness" is a good measure of success for testing techniques in automated "creative" design.

GP has been used for structural design in other fields as well: in (Lohn et al., 2004), Hornby and Lohn evolved an antenna design for NASA which was successfully deployed in space. In several works including (Spector, 2004), Spector has evolved quantum circuits. Several groups have used GP as a means to suggest a "design" in the form of a mathematical equation. These designs get manually filtered and tweaked, then deployed in the field, such as: chemical sensors (Castillo et al., 2004), geological exploration (Yu et al., 2006), and financial markets (Becker et al., 2006; Korns, 2006). Unlike the other domains mentioned, GP for circuit design has never been deployed in industry.

GP has not been deployed for circuit design in industry because (a) new designs cost millions of dollars to fabricate and test, and (b) GP-synthesized designs so far have not had the combination of sufficient complexity and trustworthiness to make the cost worth it. If the design fails, then there is not only a new fabrication needed for the revised design ("re-spin"), there is lost time to market. A new analog topology has higher chance of failure due to lack of experience with that topology; it is risky coming from an experienced designer and even more risky coming from an untrusted black box. New topologies only come about if there is no other way – if idea has possible orders of magnitude payoff that it's worth the money to try, or if there is some way to make trying it zero risk. It gets worse: addressing even just robustness (a subset of the trustworthiness issue) on a sufficiently complex problem would take 150 years on a thousand-CPU 1-GHz cluster; faster CPUs with Moore's Law (ITRS, 2007) can't help because the problem becomes more difficult as Moore's Law progresses (McConaghy and Gielen, 2005). Aerospace design has similar resistance to new structural ideas, except there if the new design fails it means that the plane or rocket crashes. Is there a path out?

Background: The Power of Domain Knowledge

Domain knowledge, if applied in the right places, can bring about orders of magnitude reduction in size of the search space, improvement in runtime, or improvement in quality of results. If we are interested in industrial applications then speed and quality of results are of utmost importance, and embedding domain knowledge can be well worth it. Domain knowledge can be applied at multiple levels of generality. We now give some illustrative examples from both evolutionary computation (EC) and other fields. In EC, each of these brought one or more orders of magnitude speedup or improvement in result quality: generative representations and modularity in general, e.g. (Hornby, 2003); permutation design via floating point representations (Rothlauf, 2006); avoiding "danglers" in circuit topology design e.g. (Koza et al., 2003a), machine-code symbolic regression (Nordin, 1994), machine-code digital logic design (Poli and Langdon, 1999), avoiding the need for learning the linear weights in symbolic

regression (Keijzer, 2004); thorough exploration of smaller building blocks, e.g. one variable at a time in symbolic regression (Korns, 2006); and more. It has been shown that GP can learn about the structure of the domain in one run to help subsequent runs (Keijzer, 2005). Some interesting examples outside of EC include: in splines, 10x-1000x or more speedup in regression by iteratively updating the least-squares learning matrix rather than doing a full update (Friedman, 1991); 1,000,000x speedup when building behavioral models of circuits, by using knowledge of its connectivity (Phillips, 1998), 1000x by exploiting sparsity in matrices (Lai and Roychowdhury, 2006); 100x space reduction via cheap-to-compute "device operating constraints" in circuits (Ding and Vemuri, 2005), 1,000,000x space reduction by reformulating the independent design variables of a design problem to more natural variables (Bernardinis et al., 2005); and more. For non-trivial practical applications, domain knowledge is key.

Reuse of Structural Domain Knowledge

In (Koza et al., 2003b), Koza et. al note: "Anyone who has ever looked at a blueprint for a building, an electrical circuit, a corporate organizational chart, a musical score, a city map, or a computer program will be struck by the ubiquitous reuse of certain basic substructures within the overall structure...Reuse can accelerate automated learning by avoiding 'reinventing the wheel' on each occasion requiring a particular sequence of already-learned steps. We believe that reuse is the cornerstone of meaningful machine intelligence." All scientific and engineering fields accumulate knowledge of useful structures over time; added new structures are literally advances in the field. For mathematics, this includes new theorems and proofs; for computer science, algorithms; for software engineering, design patterns, and libraries of code; for biology, new theories and models; for analog circuit design, taken to mean new circuit topologies.

Interestingly, "reuse" in GP systems has been reuse of structures that were found by GP during the run, or in a previous run, and not reuse of structural domain knowledge. For automotive design, GP would literally have to reinvent the wheel–and the piston, crankshaft, transmission, etc. Issues which emerge are: reinvention takes a huge amount of computational effort, if it is even tractable at all; there is no guarantee that the functionality will be hit; and because GP does not distinguish the known structures from novel structures, final designs can look very odd and therefore are less trusted.

This paper shows how reuse of structural domain knowledge simultaneously solves the GP issues of computational efficiency and of trust, for those problems which have a sufficient amount of accumulated structural domain knowledge. Figure 10-1 illustrates the general approach to such problems. We demonstrate

the approach to analog circuit design, which has accumulated a large amount of structural knowledge over the decades (Razavi, 2000; Sansen, 2006).

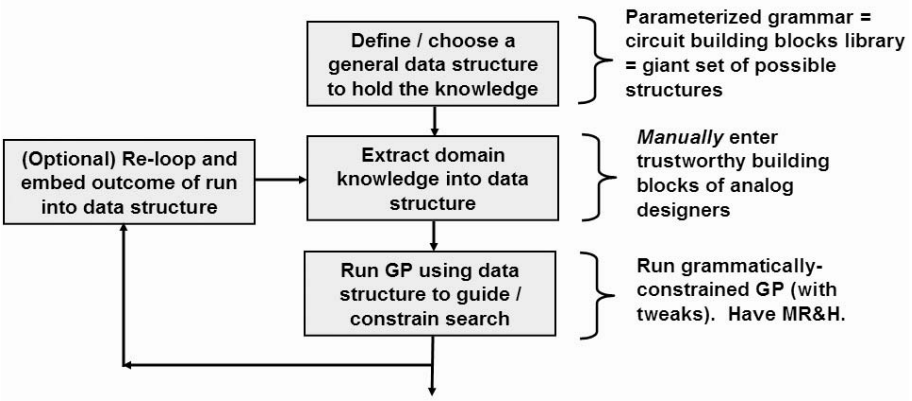


Figure 10-1. A general framework to leverage domain-specific structural knowledge with GP. Our instantiation of the framework for analog circuit design is described with the text on the right.

A Path to Practical Automated Structural Design

We now discuss various approaches to design of structures ("topologies"). The status quo approach to GP for structural design is shown in Figure 10-2 left. Figure 10-2 middle gives a flow that focuses on optimizing a fixed structure (what circuit designers currently do).

We specify our goals for a structural (topology) design tool. If a topology that is known to be 100% trustworthy will meet design goals, then the tool should return that topology. It should strive to keep the inputs and outputs as close as possible to existing techniques. It should draw on as much prior structural design knowledge as possible, so long as that knowledge is convenient to the user, it doesn't have to be convenient to the tool developer. Only if no existing known topology can meet its goals should the tool resort to adding novelty–to do so otherwise would introduce unnecessary risk. If it does add novelty, it should be easy to track where and how that novelty is added, and what the payoff is.

We now classify "automated topology design" into the following sub-categories, and discuss which of them a designer would want:

1. **Lightweight multi-topology sizing:** Search across predefined, 100% trusted topology space, but the topologies have to be input by designers. The trustworthiness is useful because it means that there is less reliance

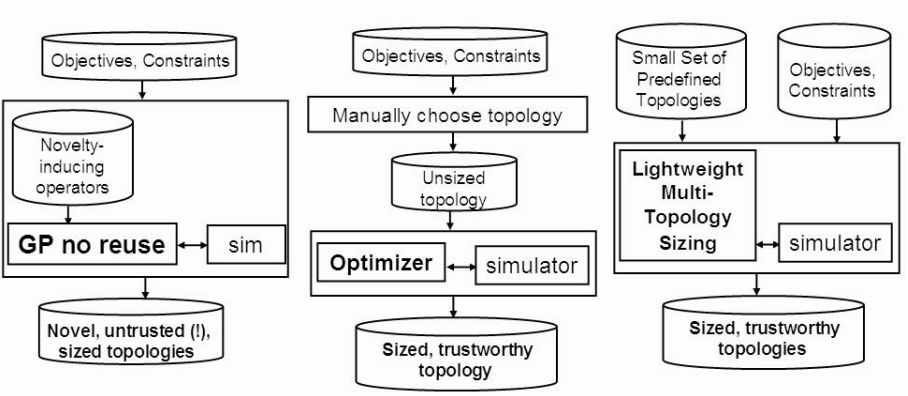


Figure 10-2. Current approaches to get sized topologies. Left: Status quo GP flow having no structural reuse – painful because topologies are untrustworthy, and huge computational burden. Middle: Current industrial flow using optimization (sizing) – painful because topology selection is manual. Right: Earlier approaches to multi-topology sizing – painful because the libraries are small and inflexible, and therefore required designer setup and intervention.

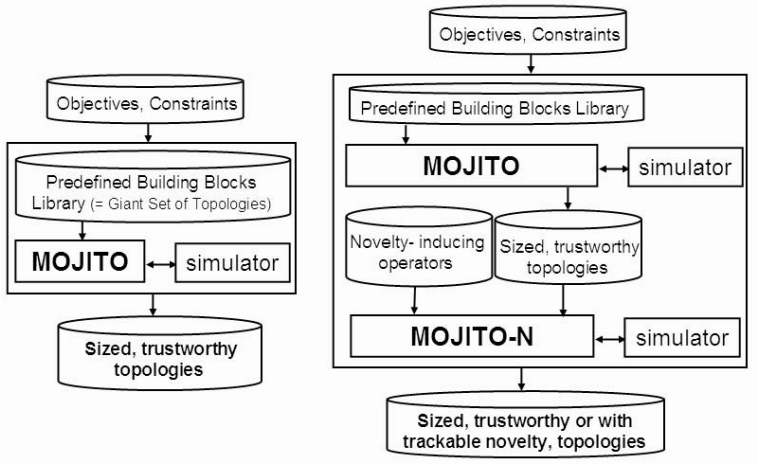


Figure 10-3. Proposed approaches to get sized topologies. Left: MOJITO: Multi-topology sizing – specs-in, sized-circuit out; gives 100% trustworthy results, but not novel designs. Right: MOJITO-N: Multi-topology sizing with novelty – gives trustworthy results *and* designs with measurable novelty.

on detailed measures to guarantee robustness and manufacturability, but it is too tedious to expect a designer to enter more than a few topologies. Even if the topology space is parameterized, it is hard to get beyond a few dozen possible topologies. Figure 10-2 right, illustrates this.

2. **Multi-topology sizing:** Search across predefined, 100% trusted topology space, where the number of topologies is sufficiently rich that the designer can consider it "complete enough" to not have to intervene in a typical design problem (i.e. hidden from view from the perspective of the designer). This is of great interest to them, because it means that it uses the same inputs and outputs as their existing tools, yet they don't have to take the time to select a topology. It is simply "specs in, sized topology out". Interestingly, if one does a (long) multi-topology sizing run with a huge number of goals set as objectives, the result itself is effectively a library of sized results; future queries for sized topologies of certain specifications are a computationally cheap lookup; i.e. it is "specs in, sized topology out, *immediately*." Figure 10-3, left, illustrates.

3. **Multi-topology sizing with innovation:** Search across 100% trusted topology space, and add novelty if there is a performance payoff. That is, "innovate" as needed. This would be of great interest for designers who are searching for new design ideas, if that is what is truly desired or needed. It is especially useful if there is a mechanism to track novelty, and therefore assess how much trust designers have in the design. Figure 10-3, right, illustrates.

4. **Topology invention from scratch:** No structural information is input (status quo GP). That is everything is "invented" (or reinvented) from scratch. Designers would question why this would ever be needed, if (3) exists. After all, why ever reinvent known structures? And they have no idea where the novelty may lie; it may be near-impossible to untangle the circuit to understand it. If they wanted extreme novelty, they would just let (3) run longer. Incidentally, because such a methodology would require a tedious iterative looping of plugging "holes in goals" for each new problem, that makes it more "hands-on" than an approach that has structural reuse. Figure 10-2, left, illustrates.

In this paper, we demonstrate how GP can be used to build the industrially interesting categories (2) and (3). The key to (2) is to aggressively reuse existing structural knowledge. The key to (3) is trustworthiness tradeoffs to ensure that only novel designs that actually give a payoff are rewarded. One might be concerned that the problems (2) and (3) are trivially easy compared to (4). Our responses are that (4) is pointlessly hard, and that one should strive to "trivialize a problem" as much as possible to help ensure its use. And we will see that problems (2) and (3) are challenging in their own right, by no means trivial to solve effectively.

2. MOJITO for Multi-Topology Sizing

MOJITO is a system for multi-objective and topology sizing. Its flow from a user perspective is shown in Figure 3 left (the diagram on the right is for novelty, described later). It actually follows a generally applicable framework for GP in structural design, as Figure 1 describes. This section describes the instantiation of the framework, specifically: how the library of structural design knowledge is defined, the GP search algorithm, and experimental results. Note: some parts of this section were originally reported in (McConaghy et al., 2007).

Background on Multi-Topology Sizing

This section reviews other approaches to multi-topology sizing in the literature. Multi-topology sizing is not a new idea, but it has never been applied to giant topology spaces, nor with SPICE in the loop (both of which greatly increase the difficulty of the problem). The work typically comes out of the analog CAD field (as opposed to an EA field). BLADES (E1-Turky and Nordin, 1986), OASYS (Harjani et al., 1992), and others (Berkcan et al., 1988; Koh et al., 1990; Toumazou et al., 1990; Swings et al., 1991; Ning et al., 1991; Antao and Brodersen, 1995; Kampe, 2000; Doboli and Vemuri, 2003; Martens and Gielen, 2006) depend on rule-based reasoning or abstract models having transforms to structural descriptions, and therefore have an undesirable amount of up-front setup effort. DARWIN (Kruiskamp and Leenaerts, 1995) and others (Maulik et al., 1995; Tang and Doboli, 2006) only require structural information, but rely on a sneaky definition of a flat combinatorial search space to define possible topologies; they do not show a clear path to generalize and are restricted to a few hundred topologies at most.

MOJITO Inputs and Outputs

The core philosophy is to use the inputs and outputs that are acceptable for industrial single-topology multi-objective sizing tools, such as (Synopsys, 2007); but to add the smallest possible amount of extra information in order to enable multi-topology sizing. Instead of a single topology, the tool takes in a set of hierarchically organized building blocks. Just like a single topology, these building blocks can be specified in an industrial circuit schematic editor. Getting such inputs is not unreasonable: such blocks do not appear as anything special to the designer, as they are merely based on well-known building blocks that one can find in any analog design textbook (Razavi, 2000; Sansen, 2006). And in fact, since we have already designed an example library (see following sections), the designers can use that. This makes it is straightforward to switch technologies, or add new building blocks.

MOJITO uses off-the-shelf simulators (e.g. SPICE) rather than specially designed performance estimators. Its output is a tradeoff of sized circuits, for selection by a designer or within a hierarchical methodology like MOBU (Eeckelaert et al., 2007). MOJITO can be seen as a pragmatic fusion of knowledge-based and optimization-based analog CAD (Rutenbar et al., 2002).

Search Space Framework

This section describes a topology space that is specified by structural information only, searchable, trustworthy, and flexible. Its flexibility is due to an intrinsically hierarchical nature which includes parameter mappings; the parameter mappings can choose sub-block implementations. It could be summarized as a parameterized grammar with a generative-representation twist.

Creating a representation for circuits is a design challenge in its own right. We choose to adopt a strongly hierarchical approach, because a flat representation is not conducive to the construction of a library or to larger designs. Analog circuit hierarchies analog be represented by analog hardware description languages (HDLs) (Ashenden et al., 2002; Kundert and Zinke, 2004), analog circuit database representations, even grammars (Ressler, 1984; Tanaka, 1993). With these options already existing in the analog domain, why not just use one of them? The problem is that if a designer makes a small conceptual change to a circuit that corresponds to a small change in performance, there may be a drastic change in the netlist. While this complicates the design of an appropriate search representation, it is needed for changes like folding an input or flipping all NMOS transistors to PMOS. Myriad examples can be found in any analog design textbook. The structural-only op amp approaches (Kruiskamp and Leenaerts, 1995; Maulik et al., 1995) do cover some of these examples, but are designed into a flat space, need special heuristics just to work in their small spaces, and do not readily generalize. The existing grammatical approaches did not provide enough flexibility.

The generative representation GENRE (Hornby, 2003) provided inspiration. A generative representation transforms a genotype to phenotype by executing the genotype commands as if they were a program. Unfortunately, GENRE does not readily allow one to embed known trusted building blocks, and is too flexible in allowing the addition and removal of ports on substructures during search. The MOJITO representation removes some flexibility in order to allow easier embedding of domain knowledge; it has an associated drawing style that both analog designers and computer scientists will understand. It is composed of three simply-defined "Part" types, which we now describe.

Let us define a "Part" as merely a circuit block at any level of the hierarchy. It has a fixed set of arguments in its interface: "port arguments" (nodes available

to the outside world) and "number arguments" (parameters which affect its behavior, e.g. a device size). Arguments to a Part's embedded Parts are a function of arguments above. To fully netlist a given Part, the only extra information needed is values for the arguments to that Part. Direct-representation Part types are:

- **Atomic Part Type.** Parts of this type are the leaf nodes in the hierarchy (tree) of Parts. They do not contain any embedded parts. Figure 10-4 gives examples.

- **Compound Part Type.** These have one or more sub-Parts embedded. Sub-parts can have internal connections among themselves and to the Part's external ports. All sub-parts get netlisted. Figure 10-5 gives examples.

We add the following generative Part type. It netlists by executing the Part as a function of a third type of argument in its interface: "topological arguments":

- **Flexible Part Type.** These have the topological argument "choice_index", which during netlisting is used to select one of several candidate embedded parts and respective wirings. The argument values going into the chosen sub-part can be very particular to that sub-part if the mapping function has cases for different choice_index values. Example: a current mirror which may be simple or cascode (choice_index = 0 or 1). Figure 6 gives an example.

Despite the simplicity of Part types, the interactions allow a capture of essential structural domain knowledge of analog building blocks. The generative Parts, especially Flexible Parts, are what turn a Part into its own IC *library of possibilities* rather than merely a representation of a single circuit design. Traversing the topology space merely means changing one or more of the "topological argument" input values.

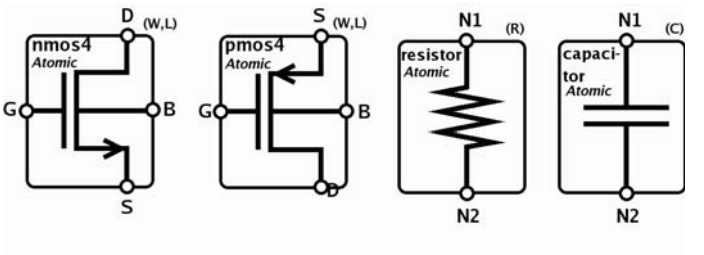


Figure 10-4. Example Atomic Parts: nmos4 transistor, pmos4 transistor, resistor, capacitor.

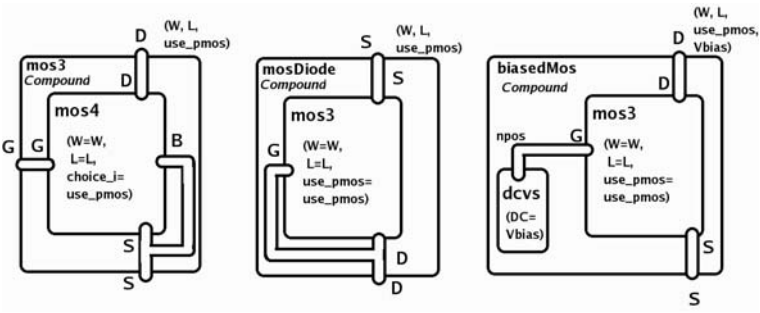


Figure 10-5. Example Compound parts. mos3 is a wrapper for mos4, so that the mos4's 'B' node is not seen at higher levels. mosDiode ties together two internal ports to only present two external ports. biasedMos uses a 1-port dcvs (dc-controlled voltage source) part to set its gate bias internally.

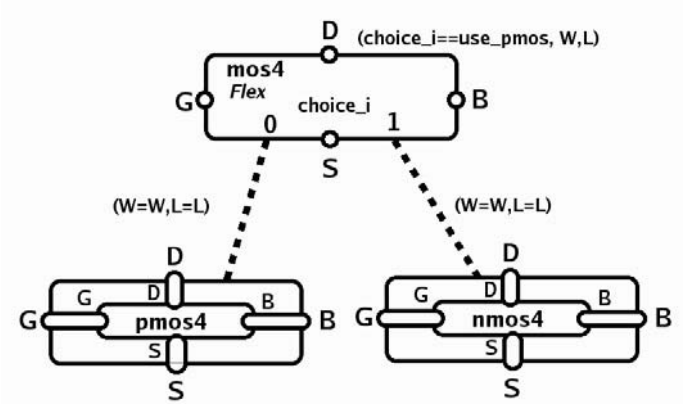


Figure 10-6. Example Flex part: mos4 turns the choice of NMOS vs. PMOS into a parameter "*choice_index*". Note how parameters get assigned from mos4 to either of its sub-blocks. In this case both sub-blocks use the mos4's W and L parameters as their own W and L values.

Remember that for all these subblocks, instantiation into sets of nmos vs. pmos devices is deferred until the very leaf block, based on the parameters that flow through the circuit. This sort of flexibility allows for a large number of topologies at the top level, without having an excess number of building blocks. It also means that many parameters are shared in the conversion from one block to subblocks, which keeps the overall variable count lower than it might have been; this is crucial to the locality of the space and thus the ultimate success of the search algorithm. Figure 7 gives an example of a circuit "sentence" instantiated in the parameterized grammar of MOJITO; this sentence will also be a GP individual tree.

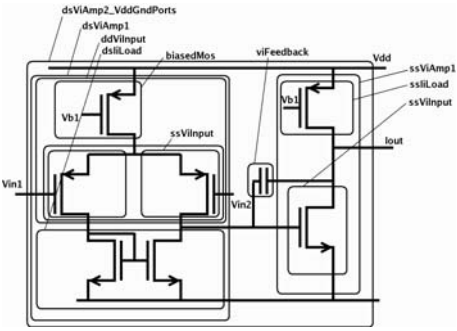


Figure 10-7. Example of MOJITO Building Blocks on a PMOS-input Miller OTA.

3. MOJITO Search Algorithm

We now proceed to describe an algorithm that traverses the MOJITO search space to produce a set of topologies that collectively trade off performances.

The search space is gigantic and diverse, because there can be thousands of possible topologies plus associated sizings. This means a mix of hierarchy and parameters which can be continuous-, discrete-, or integer-valued. SPICE-accurate performance estimation adds computational demand too, compared to the simplified performance estimators that most previous multi-topology sizing approaches used.

- An evolutionary search algorithm that balances exploration with exploitation by grouping individuals by genetic age (ALPS (Hornby, 2006)), and at a nested level achieves multiobjective search by grouping individuals by degree of nondomination (NSGA-II (Deb et al., 2002)).

- Special operators that are designed to exploit the nature of the search space. The crossover operator respects the parameters that should be held together within building blocks, yet still allows sibling building blocks to share parameters (i.e. a mix between vector and tree search spaces). The mutation operator has tactics to avoid stealth mutations (Rothlauf, 2006) on "turned-off" building blocks.

Structure of Search Space from Search Algorithm's Perspective

Each building block has its own parameters, which fully describe how to implement it and its sub-blocks. As we build up the hierarchy of building blocks, we eventually reach the level of the block we want to search for, such as the amplifier block. Thus, the search space for the circuit type (e.g. fully differential amplifier) is merely the possible values that each of the block's parameters can

take. Since these parameters can be continuous, discrete, or integer-valued, one could view the problem as a mixed-integer nonlinear programming problem, which one could solve with an off-the-shelf algorithm whether it be a classical MINLP solver or an evolutionary algorithm (EA) operating on vectors. But a vector-oriented view does not recognize the hierarchy, and so operations on it may have issues. One issue is that a change to variable(s) may not change the resulting netlist at all, because those variables are in sub-blocks that are turned off. From the perspective of a search algorithm, this means that there are vast regions of neutrality (Huynen et al., 1996); or alternatively the representation is non-uniformly redundant and runs the risk of stealth mutations (Rothlauf, 2006). For EAs, another issue is that an n-point or uniform crossover operator could readily disrupt the values of the building blocks in the hierarchy, e.g. the sizes of some sub-blocks' transistors change while others stay the same, thereby hurting the resulting topology's likelihood of having decent behavior. From an EA perspective this means that the "building block mixing" is poor (Goldberg, 2002).

What if we reconcile the hierarchy? We cannot apply a hierarchical design methodology such as (Chang, 1997; Eeckelaert et al., 2005), because there are no goals on the sub-blocks, just the highest-level blocks (we could, however, still apply hierarchal methodology to the results). Neither can we treat it completely as a tree induction problem (to be solved, for example, by grammar-based genetic programming (Whigham, 1995)) because some sibling sub-blocks share the same parent blocks' parameters.

So the search algorithm's perspective of the space has both tree-based and vector-based aspects. We design novel operators that reconcile both aspects, for use within an EA. First, we have a mutation operator which chooses one or more parameters to mutate. Continuous-valued parameters follow Cauchy mutation (Yao et al., 1999) which allows for both tuning and exploration. Integer-valued "part choice" parameters follow a discrete uniform distribution. Other integer and discrete parameters follow discretized Cauchy mutations. To avoid stealth mutations on "turned-off" building blocks, mutations are only kept if the netlist changes; mutation attempts are repeated until this happens. Though "neutral wanderings" of the space has been shown to help exploration in some applications (Vassilev and Miller, 2000; McConaghy et al., 2005), results are mixed and in general make performance more unpredictable (Rothlauf, 2006). We prefer predictability, and rely on ALPS to enhance exploration.

The second operator is crossover. It works as follows: given two parent individuals, randomly choose a sub-block in parent A, identify all the parameters associated with that sub-block, and swap those parameters between parent A and parent B. This will preserve the parameters in the sub-blocks. There will still be some crosstalk because sibling blocks may use those parameters as well, but the crosstalk is relatively small compared to the 100% crosstalk that we'd

have if we used standard vector-based crossover. This effectively makes the search a hybrid between tree-based and string-based search (i.e. a cross between a GA and GP).

To generate random individuals, we merely randomly choose a value for each parameter using a uniform distribution.

The Search Algorithm

Even with a search space and operators that are as well-behaved as possible, there is a need for a highly competent search algorithm because the space is so large (there is such a large set of possible topologies and associated sizings), and the performance estimation time for an individual can be on the order of minutes (using SPICE to maintain industrial relevance). We also need multi-objective results. The blow is softened a bit because some degree of parallel computing is allowed (industrial setups for automated sizing typically have 5-30 CPUs).

A popular, competent EA for multiobjective is NSGA-II (Deb et al., 2002), which sorts individuals by nondomination layer. NSGA-II provides a reasonable starting point for us in the design of our multiobjective EA. We use the constraint-handling approach of NSGA-II as well: a feasible individual always dominates an infeasible one, and for two infeasible individuals the one that dominates is the one with the least total constraint violation (a sum across all constraints' violations).

One key issue with NSGA-II, and most EAs, is that they can converge prematurely. To fix this, one needs to ensure an adequate supply of building blocks (Goldberg, 2002). Tactics include massive population sizes (Koza et al., 2003a), restarting, time-varying population sizes, or diversity measures such as crowding. All tactics are all either inadequate or highly sensitive to parameter settings. Random injection of individuals for fresh new building blocks might help, except they get killed off too quickly during selection. To fix that, HFC (Hu et al., 2005) segregates individuals into similar fitness layers, and restricts competition to within layers, which gives random individuals a reasonable chance. Unfortunately, the choice of fitness thresholds is complicated in practice, and near-stagnation may occur at some fitness levels because the best individuals per level have no competition. The age-layered population structure, ALPS (Hornby, 2006) builds on HFC, but rather than segregate individuals by fitness it segregates by genetic age levels. The age distinction overcomes the issues of HFC. For example, age level 0 might allow individuals with age 0-19, level 1 allows age 0-39, level 2 allows age 0-59, and so on until the top level (e.g. level 9) which allows individuals of any age. Genetic age is the number of generations of an individual's oldest genetic material: the age of a randomly generated individual is 0; the age of a child is the maximum of its parents' ages; age is incremented by 1 each generation. If an individual gets too old

for a fitness level, it gets kicked out of that level and given one last chance to compete at the next higher level. Selection at one age level uses the individuals at that level and at one level below as candidates.

Only a single-objective, single-CPU ALPS exists in the literature. In this paper, we make it multi-objective for the first time. There are many conceivable ways to make ALPS multi-objective. We chose a pragmatic approach which is shown in Figure 10-8. There is canonical NSGA-II evolution at each age level, with one difference: for selection at a level l, the individuals at level l and level $l-1$ are candidates (rather than just at level l). In this fashion, younger high-fitness individuals can propagate to higher levels.

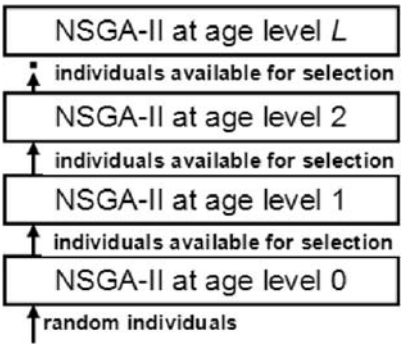


Figure 10-8. Multi-objective ALPS has NSGA-II at each age level.

4. MOJITO Multi-Topology Sizing: Experimental Results

This section describes application of MOJITO to two multi-objective multi-op-amp topology sizing problems.

Problem Setup

The problems were set up as follows. The search space had 50 variables (topology selection variables and sizing variables). EA settings were: 100 individuals per age layer; 10 age layers, maximum age per layer: 9, 19...79, 89, infinity. Each run took approximately 150 hours on a single-core 2.0 GHz Linux machine, covering 100,000 individuals. Search objectives: maximize GBW, minimize power, maximize DC Gain (Experiment Set 2). Constraints: phase margin $> 65°$, all DOCs, DC Gain $>$ 30dB (Experiment Set 1). Simulator was HSPICE. Technology was 0.18μ CMOS; supply voltage 1.8V; load capacitance 1pF.

Experiment Set 1

These runs were to verify the algorithm's ability to traverse the search space and select different topologies. The problem was set up such that the optimization end result was known a priori. Three GP runs were done, with problem setups such that specific output topologies were expected. To summarize results for non-circuit people: it achieved the structures which were expected. The rest of this paragraph gives circuit-specific details. The only difference between the 3 runs is the common mode voltage ($V_{cmm,in}$) at the input. We know that for $V_{cmm,in}$ = 1.5V, topologies must have an NMOS input pair. For $V_{cmm,in} = 0.3V$, topologies must have PMOS inputs. At $V_{cmm,in} = 0.9V$, there is no restriction between NMOS and PMOS inputs. Figure 10-4 illustrates the outcome of the experiments. It contains the combined results of three optimization runs. Result (a) has $V_{cmm,in} = 1.5V$, and has only topologies with NMOS inputs. It chose to use 1-stage and 2-stage amplifiers, depending on the power-GBW tradeoff. Result (b) has $V_{cmm,in} = 0.3V$, and MOJITO only returns PMOS input pairs. Note that result (a) is a result before convergence in order to retain the 2-stage amplifier in the result set. Older generations eliminate the 2-stage amplifier in favor of the folded cascode amplifier, as in result (b). For result (c) a $V_{cmm,in} = 0.9V$ has been specified. Though both NMOS and PMOS input pairs might have arisen, the optimization preferred NMOS inputs. The curve clearly shows the switch in topology around GBW=1.9GHz, moving from a folded cascode input to a simple current-mirror amp. Interestingly, the search retained a stacked current-mirror load for about 250MHz GBW.

Experiment Set 2

In second experiment, one GP run was done, to verify that MOJITO could get interesting groups of topologies in a tradeoff of three objectives. The motivation is as follows: whereas a single-objective multi-topology optimization can only return one topology, the more objectives that one has in a multi-topology search, the more opportunity there is for many topologies to be returned, because different topologies naturally lie in different regions of performance space. Results are shown in Figure 5. We can see that MOJITO found rich and diverse structures as expected. The rest of this paragraph has circuit-specific details. It determined: a folded-cascode op amps gave high gain-bandwidth but with high area, 2-stage amps give high gain but at the cost of high area, the low-voltage current mirror load is a 1-stage with high gain, and there are many other 1 stage topologies which give a broad performance tradeoff. These are all results that a circuit designer would expect.

Incidentally, problems of comparative complexity took status quo GP (i.e. no reuse) 100 million or more individuals (Koza et al., 2003a; Koza et al., 2003b), and the results were not trustworthy; it was estimated that to get to

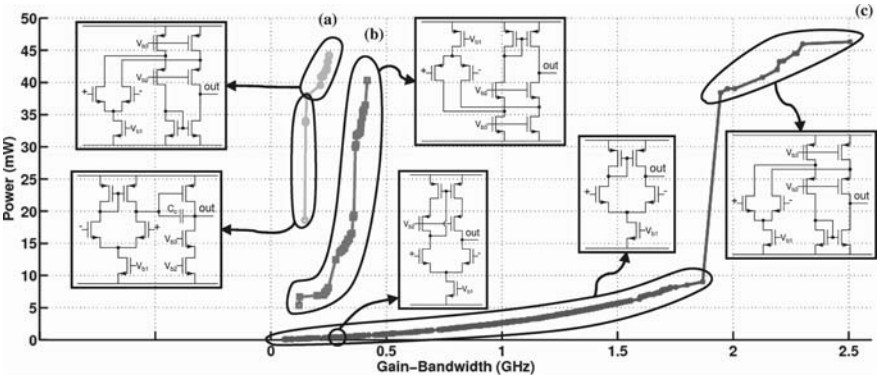


Figure 10-9. Pareto fronts for 3 GP runs a/b/c which had different input settings. The y-axis is an objective to minimize, and the x-axis is an objective to maximize; each point is an individual, which has an associated structure (topology) and parameters (sizings). Some of the specific topologies found are shown; these are the expected topologies.

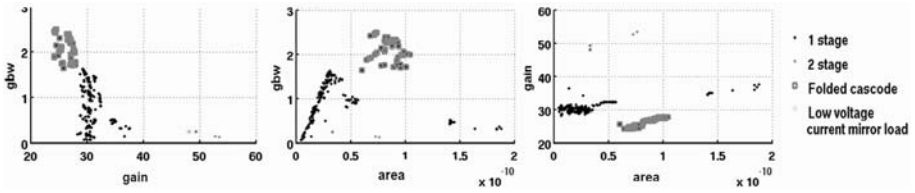


Figure 10-10. Pareto front for a GP run on 3 objectives (maximize gbw, maximize gain, minimize area). Individuals are grouped according to some of their structural characteristics (e.g. 1 stage vs. 2 stage) to illustrate their diversity.

get a reasonable degree of robustness would take 150 years on a 1,000 node 1-Ghz cluster (McConaghy and Gielen, 2005). That is, it would have taken ((150 years * 365 days / year * 24 hours / day) * 1000 CPUS * 1Ghz) / ((150 hours) * 1 CPU * 2 Ghz) = 4.4 million times more computational effort than MOJITO to get comparable results. There's a lot to be said for topology reuse.

5. How Far can Reuse-Only Go? (With No Novelty)

This section describes how huge a fully trustworthy (reuse-only, no novelty) space can become.

The first major question of this subsection is: Can the number of possible topologies be sufficiently rich so that the designer can consider it "complete enough" to not have to intervene in a typical design problem? We calculate the size as follows. The count for an atomic block is one; for a flexible block,

it's the sum of the counts of each choice block; for a compound block, it's the product of the counts of each of its sub-blocks–but there are subtleties. Subtlety: for a given choice of flexible block, other choice parameters at that level may not matter. Subtlety: one higher-level choice might govern > 1 lower-level choices, so don't overcount. Table 1 shows that MOJITO increases the op amp count by 50x compared to the other reuse-only techniques.

Table 10-1. Size of Op Amp Topology Spaces.

Technique	**# topologies**	**Trustworthy?**
GP without reuse, e.g. (Koza et al., 2003a)	billions	NO
DARWIN (Kruiskamp and Leenaerts, 1995)	24	YES
MINLP (Maulik et al., 1995)	64	YES
GP with reuse: MOJITO (this work)	**3528**	**YES**

The second major question of this subsection is: How big can the space of possible trustworthy topologies for an industrially relevant application get? Compared to what we have just established, we can make the space even larger in many ways, using new techniques, recursion, and system-level design:

- **Add more design techniques.** The field of analog design is a research field in its own right, with its own conferences, journals, etc. Core advances in that field are precisely: new topologies and techniques. One can think of that design effort as (manual) co-evolution of building block topologies. Design opportunities and challenges arise due to new applications, different target specifications, and the steady advance of Moore's Law (ITRS, 2007). Each design technique advance would increase the size of the space by at least 2x, so if we merely took the top 10 advances in op amp design, we would increase the space by at least $2^{10} = 10^3$, bringing the count to $3.5 * 10^6$. And that is a lowball estimate: more realistically one would consider dozens or hundreds of advances, and some advances could be used in multiple places in the design; if we had 10 advances which doubled, 10 which tripled, and 10 which quadrupled, then the space increases by $2 * 10 * 3^{10} * 4^{10} = 6 * 10^{13}$, to total $2 * 10^{17}$ trustworthy op amp designs.

- **Recursion.** Circuits' designs can recurse. For op amps, this is via "gain boosting." One level of recursion brings the count to $(2 * 10^{17})^2 = 4 * 10^{34}$, and two levels of recursion (i.e. gain boosted amps using gain boosted amps) brings the count to $(4 * 10^{34})^2 = 1.6x10^{69}$ trustworthy op amp designs. Yes, designers in industry do actually use two levels of gain boosting, in combination with the latest design techniques.

- **System-level design.** So far we have just talked about an op amp space which is a circuit at lowest level of the design hierarchy (cell level), but

higher levels exist too. The next-highest level includes circuits such as data converters (A/Ds, D/As), active filters, and more. These circuits use lower-level blocks like op amps. The level above that is typically the whole analog system, e.g. a radio transceiver like a Bluetooth or Wi-Fi implementation. The level above that would typically combine the analog and digital parts into a mixed-signal system. Each level can have many sub-blocks, and each of those sub-blocks can be any of its combinations. E.g. an A/D might have 8 different op amps. If each op amp had $1.6 * 10^{69}$ possible topologies and even if there was no other topological variation at the A/D level, it means $(1.6 * 10^{69})^8 = 4.2{*}10^{553}$ possible A/D topologies. Let's say the system at one level higher up had an A/D, a D/A, and a couple other parts all with about the same number of topologies; then its size would be $(4.2 * 10^{553})^4 = 3.1x10^{2214}$ possible topologies. (For reference, if just 3528 designs at the cell level, that leads to $((3528)^8)^4 = 10^{113}$ designs).

Combinatorial explosion is a good thing: the more possibilities available for *any* part type, the more possible trustworthy designs you can have. (1) If one can decompose their design into sub-problems (where each sub-problem has its own goals), (2) if one has a competent hierarchical design methodology, (3) if the problem of "massively multi-topology" cell-level sizing design can be cracked, then one can ultimately do system-level 100% trustworthy topology design in spaces with 10^{113} designs, 10^{832} designs, or more.

We *can* do (1) because the decomposition is obvious in circuit design, and the names of sub-blocks are well-established (op amps, bias generators, A/Ds, D/As, filters, phase-locked loops, etc) (Razavi, 2000; Sansen, 2006). We can do (2) because competent hierarchical design methodologies have been demonstrated; and recently it has been demonstrated that they can choose from among different candidate topologies (Eeckelaert et al., 2007). This paper has demonstrated (3).

6. Multi-Topology Sizing with Novelty

Because of the costs of fabricating a design, the motivation for a new topology has to be strong. New topologies only come about if there is no other way, if idea has possible orders of magnitude payoff that it's worth the money to try, or if there is some way to make trying it zero risk. That said, sometimes these motivations exist, and therefore it is of interest to see what sort of effective algorithms can be created. This section describes MOJITO-N, a system for multi-objective and topology sizing, that adds novelty as needed, with the flow of Figure 10-3, right.

The Search Algorithm

The specifications for such a system, above and beyond (non-novelty) MOJITO, are:

- If a topology that is known to be 100% trustworthy will meet their goals, then the tool should return that.
- Only if no existing known topology can meet their goals should the tool resort to adding novelty.
- If it does add novelty, it should be easy to track where and how that novelty is added, and what the payoff is.

These specifications are resolved in MOJITO-N as follows:

- Use trustworthy designs as the structural starting points. In fact, do a long 100% trustworthy run first; then add novelty in a follow-on run.
- Create novel designs by: copying an existing part within the parts library, mutating the copy, and then getting a new individual to use that mutated copy. In order to track novelty, remember which parts and choices are novel, and what sort of novelty-mutating operator is used. These altered libraries can be subsequently reused in future runs, therefore closing the loop in the style of run-transferable-libraries (Keijzer, 2005).
- Have a multi-objective framework to manage trustworthiness tradeoffs: trust = −novelty, novelty = number of times that a novel part is used, and a novel part is one that has had random structural mutations. Therefore, if novelty does not actually help, it will not show up in the Pareto optimal front (but it will not necessarily be kicked out of the population; that is up to the multiobjective algorithm).
- A novel design will almost certainly be initially worse off than a non-novel design, until it has been sized well enough to be competitive. If not handled explicitly in the EA framework, the novel design will almost certainly die off before its benefit is discovered (if it has a benefit). So that novel designs have a fighting chance, only create novel designs for the easiest-competition age layer 0. Rather than randomly generating the whole individual from a uniform distribution, choose a parent from any age layer, and novelty-mutate it for placement in layer 0. (Note: a plethora of other possible schemes exist here too, but a key enabler is the ALPS structure).

Experiment

The experimental setup was the same as for the non-novelty MOJITO, except for the following differences. The 100 trustworthy results from the MOJITO "Experiment Set 2" run were used as the inputs to the MOJITO-N run. MOJITO-N was run for 15 more generations (15 * 10 * 100 = 15000 more individuals), which took about 25 hours. The novelty-mutating operators were: add two-port series, add two-port parallel, add n-port parallel. The two-port parts available for add were: capacitors, resistors, nmos/pmos diodes, and biased nmos/pmos devices (a biased mos is merely transistor with a pre-set voltage bias). One more search objective was added: minimize novelty.

With the results, we output the nondominated set, and first examined if any novel individuals existed. Some did. With each novel individual, we queried its data structure to find which parts were novel, and how they were than their original part. It turns out that so far in this run, they all had the same change: the feedback capacitor Cc had been mutated to include a resistor in series. Figure 9 illustrates. This is actually a well-known design technique that one can find in many analog design textbooks: what it does is increase the effective gain from feedback; it does not help the feedforward gain as much because the feedforward path does not get its gain amplified.

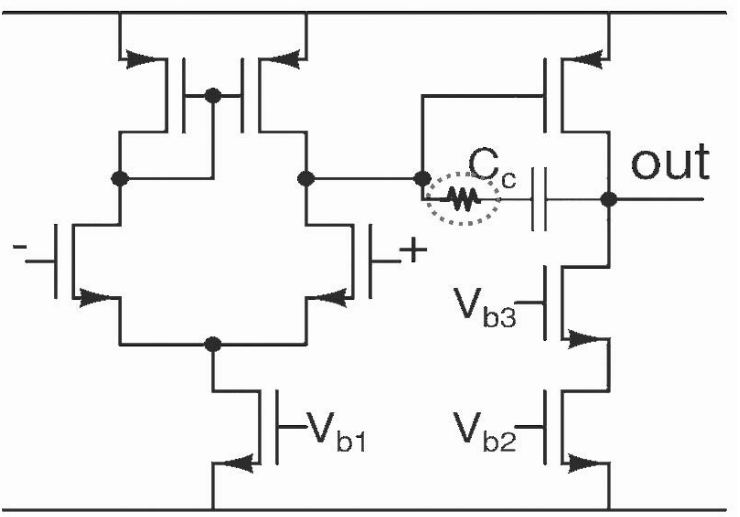


Figure 10-11. Circuit which MOJITO-N successfully re-invented. The circled resistor in the feedback path was not in the library; MOJITO-N added it; this is a well-known design technique.

7. Conclusion

This paper showed how aggressive reuse of known designs brings a vast reduction in computational effort in GP applied to automated structural design. It presented a complementary pair of approaches that incorporate reuse:

- MOJITO automatically designs 100% trustworthy structures of industrially relevant complexity, with commercially reasonable computational effort. MOJITO's effectiveness was demonstrated in two separate experiments, showing how it hit the target designs as expected, from a library of more than 3000 possible topologies.
- MOJITO-N adds novelty to the trustworthy designs, and returns circuits that trade off novelty with performance, also with commercially reasonable computational effort. The novelty is fully trackable, so all changes can be readily understood. MOJITO-N successfully re-invented a known design of industrially relevant complexity.

To properly capture the relevant knowledge to reuse, we designed a parameterized generative representation , and then used the representation to encode a library of building blocks for the specific problem (in our case, operational amplifier design). The key to manage trustworthiness in the presence of novelty was to add an extra objective of "minimize novelty" within a multi-objective optimization framework, which results in trustworthiness tradeoffs. "Novelty" is the number of structural mutation steps taken from a 100% trustworthy design. We view our novelty-approach as "automated innovation" rather than "automated invention" because it builds on existing knowledge – but note that patents are awarded for innovations too.

This work also used state-of-the-art ideas in EA design. It had a hybridized tree/vector view of the search space, implemented as operators having those two perspectives. It was guided by recent advances in theory of EA representations (Rothlauf, 2006). To avoid premature convergence and minimize sensitivity to population size setting, we employed the age-layered population structure (ALPS) (Hornby, 2006), and embedded NSGA-II (Deb et al., 2002) into each age layer of ALPS to make it multiobjective.

These techniques can be readily extended to other GP problem domains of interest, and are complementary with many other recent advances in GP.

References

Antao, B.A.A. and Brodersen, A.J. (1995). Archgen: Automated synthesis of analog systems. *IEEE Transactions on Very Large Scale Integrated Circuits*, 3(2):231–244.

Ashenden, Peter J., Peterson, Gregory D., and Teegarden, Darrell A. (2002). *The System Designer's Guide to VHDL-AMS*. Morgan Kaufmann.

Becker, Ying, Fei, Peng, and Lester, Anna M. (2006). Stock selection : An innovative application of genetic programming methodology. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages –. Springer, Ann Arbor.

Berkcan, E., d'Abreu, M., and Laughton, W. (1988). Analog compilation based on successive decompositions. In *Design Automation Conference*, pages 369–375.

Bernardinis, F. De, Nuzzo, P., and Sangiovanni-Vincentelli, A.L. (2005). Mixed signal design space exploration through analog platforms. In *Design Automation Conference*, pages 875–880.

Castillo, Flor, Kordon, Arthur, Sweeney, Jeff, and Zirk, Wayne (2004). Using genetic programming in industrial statistical model building. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 3, pages 31–48. Springer, Ann Arbor.

Chang, Henry (1997). *A Top Down, Constraint Driven Design Methodology for Analog Integrated Circuits*. Kluwer.

Dastidar, T.R. and Chakrabarti, P.P. (2005). A synthesis system for analog circuits based on evolutionary search and topological reuse. *IEEE Transactions on Evolutionary Computation*, 9(2):2005.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Ding, Mengmeng and Vemuri, Ranga (2005). A combined feasibility and performance macromodel for analog circuits. In *Design Automation Conference*, pages 63–68.

Doboli, Alex and Vemuri, Ranga (2003). Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. *IEEE Transactions on Computer-Aided Design*, 22(11).

E1-Turky, F.M. and Nordin, R.A. (1986). Blades: An expert system for analog circuit design. In *International Conference on Circuits and Systems*, pages 552–555.

Eeckelaert, Tom, McConaghy, Trent, and Gielen, Georges G. E. (2005). Efficient multiobjective synthesis of analog circuits using hierarchical paretoÂ–optimal performance hypersurfaces. In *Design Automation and Test Europe*.

Eeckelaert, Tom, Schoofs, Raf, Gielen, Georges G. E., and Steyaert, Michiel (2007). An efficient methodology for hierarchical synthesis of mixed-signal systems with fully integrated building block topology selection. In *Design Automation and Test Europe*.

Friedman, Jerome H. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, 19(1-141).

Goldberg, David E. (2002). *The Design of Innovation*. Springer.

Harjani, R., Rutenbar, R., and Carley, L. (1992). Oasys: A framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design*, 8(12):1247–1266.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Hornby, Gregory Scott (2003). *Generative Representations for Evolutionary Design Automation*. PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA.

Hu, Jianjun and Goodman, Erik (2004). Topological synthesis of robust dynamic systems by sustainable genetic programming. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 9, pages ??–157. Springer, Ann Arbor. pages missing.

Hu, Jianjun, Goodman, Erik, Seo, Kisung, Fan, Zhun, and Rosenberg, Rondal (2005). The hierarchical fair competition framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277.

Huynen, M.A., Stadler, P., and Fontana, W. (1996). Smoothness within ruggedness: The role of neutrality in adaptation. *National Academy of Sciences USA*, 93:397–401.

ITRS (2007). International technology roadmap for semiconductors.

Kampe, Jurgen (2000). A new approach for the structural synthesis of analog subsystems. In *International Workshop on Symbolic Methods and Applications in Circuit Design*, pages 33–38.

Keijzer, Maarten (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269.

Keijzer, Maarten (2005). Run transferable libraries. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*. Kluwer.

Koh, H.Y., Séquin, C.H., and Gray, Paul. R. (1990). Opasyn: A compiler for cmos operational amplifiers. *IEEE Transactions on Computer-Aided Design*, 9:113–125.

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Pro-*

gramming Theory and Practice IV, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

Koza, John R., Jones, Lee W., Keane, Martin A., and Streeter, Matthew J. (2004). Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 8, pages 120–?? Springer, Ann Arbor. pages missing?

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003a). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Koza, John R., Streeter, Matthew J., and Keane, Martin A. (2003b). Automated synthesis by means of genetic programming of complex structures incorporating reuse, hierarchies, development, and parameterized toplogies. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practise*, chapter 14, pages 221–237. Kluwer.

Kruiskamp, Wim and Leenaerts, Domine (1995). Darwin: Cmos opamp synthesis by means of a genetic algorithm. In *Design Automation Conference*.

Kundert, K. and Zinke, O. (2004). *The Designer's Guide to Verilog-AMS*. Kluwer.

Lai, X. and Roychowdhury, Jaijeet (2006). Macromodeling oscillators using krylov-subspace methods. In *Asia And South Pacific Design Automation Conference*.

Lohn, Jason, Hornby, Gregory, and Linden, Derek (2004). Evolutionary antenna design for a NASA spacecraft. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. Springer, Ann Arbor.

Lohn, Jason D. and Colombano, S.P. (1998). Automated analog circuit synthesis using a linear representation. In *International Conference on Evolvable Systems*, pages 125–133.

Martens, Ewout and Gielen, Georges G.E. (2006). Top-down heterogeneous synthesis of analog and mixed-signal systems. In *Design Automation and Test Europe*, pages 275–280.

Maulik, Peter C., Carley, L.R., and Rutenbar, R. (1995). Integer programming based topology selection of cell level analog circuits. *IEEE Transactions on Computer-Aided Design*, 14(4).

McConaghy, Trent, Eeckelaert, Tom, and Gielen, Georges (2005). CAFFEINE: Template-free symbolic model generation of analog circuits via canonical

form functions and genetic programming. In *Proceedings of the Design Automation and Test Europe (DATE) Conference*, volume 2, pages 1082–1087, Munich.

McConaghy, Trent and Gielen, Georges (2005). Genetic programming in industrial analog CAD: Applications and challenges. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 19, pages 291–306. Springer, Ann Arbor.

McConaghy, Trent, Palmers, Pieter, Gielen, Georges G.E., and Steyaert, Michiel (2007). Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *Design Automation Conference*.

Ning, Z., Mouthaan, A.J., and Wallinga, H. (1991). Seas: A simulated evolution approach for analog circuit synthesis. In *Custom Integrated Circuits Conference*.

Nordin, Peter (1994). A compiling genetic programming system that directly manipulates the machine code. In Kinnear, Jr., Kenneth E., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press.

Phillips, Joel R. (1998). Model reduction of time-varying linear systems using approximate multipoint krylov-subspace projectors. In *International Conference on Computer-Aided Design*, pages 96–102.

Poli, Riccardo and Langdon, William B. (1999). Sub-machine-code genetic programming. In Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter J., editors, *Advances in Genetic Programming 3*, chapter 13, pages 301–323. MIT Press, Cambridge, MA, USA.

Razavi, Behzad (2000). *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

Ressler, Andrew L. (1984). *A Circuit Grammar for Operational Amplifier Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Rothlauf, Franz (2006). *Representations for genetic and evolutionary algorithms*. Springer-Verlag, pub-SV:adr, second edition. First published 2002, 2nd edition available electronically.

Rutenbar, R.A., Gielen, Georges G.E., and Antao, B.A. (2002). *Computer-Aided Design of Analog Integrated Circuits and Systems*. IEEE Press, Piscataway, NJ, USA.

Sansen, Willy (2006). *Analog Design Essentials*. Springer.

Shibata, Hajime, Mori, Soji, and Fujii, Nobuo (2002). Automated design of analog circuits using cell-based structure. In *Nasa/DoD Conference on Evolvable Hardware*.

Spector, Lee (2004). *Automatic Quantum Computer Programming: A Genetic Programming Approach*, volume 7 of *Genetic Programming*. Kluwer Academic Publishers, Boston/Dordrecht/New York/London.

Sripramong, Thanwa and Toumazou, Christofer (2002). The invention of CMOS amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1237–1252.

Swings, K., Donnay, S., and Sansen, W. (1991). Hector: a hierarchical topology-construction program for analog circuits based on a declarative approach to circuit modeling. In *Custom Integrated Circuits Conference*.

Synopsys (2007). Circuit explorer product. *Website of Synopsys Inc.*

Tanaka, T. (1993). Parsing electronic circuits in a logic grammar. *IEEE Transactions Knowledge and Data Engineering*, 5(2):225–239.

Tang, H. and Doboli, A. (2006). High-level synthesis of delta-sigma modulator topologies optimized for complexity, sensitivity, and power consumption. *IEEE Transactions on Computer-Aided Design*, 25(3):597–607.

Toumazou, Chris, Makris, C.A., and Berrah, C.M. (1990). Isaid - a methodology for automated analog ic design. In *International Symposium on Circuits and Systems*, volume 1, pages 531–555.

Vassilev, Vesselin K. and Miller, Julian F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems*, pages 252–263. Springer-Verlag.

Whigham, P. A. (1995). Grammatically-based genetic programming. In Rosca, Justinian P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA.

Yao, Xin, Liu, Yong, and Lin, Guangming (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2).

Yu, Tina, Wilkinson, Dave, and Castellini, Alexandre (2006). Applying genetic programming to reservoir history matching problem. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 6, pages –. Springer, Ann Arbor.

Chapter 11

ROBUST ENGINEERING DESIGN OF ELECTRONIC CIRCUITS WITH ACTIVE COMPONENTS USING GENETIC PROGRAMMING AND BOND GRAPHS

Xiangdong Peng[1], Erik D. Goodman[1] and Ronald C. Rosenberg[2]

[1]*Gentic Algorithms Research and Applications Group (GARAGe), Dept. of Electrical and Computer Engineering, Michigan State University* [2]*Dept. of Mechanical Engineering, Michigan State University*

Abstract Genetic programming has been used by Koza and many others to design electrical, mechanical, and mechatronic systems, including systems with both active and passive components. This work has often required large population sizes (on the order of ten thousand) and millions of design evaluations to allow evolution of both the topology and parameters of interesting systems. For several years, the authors have studied the evolution of multi-domain engineering systems represented as bond graphs, a form that provides a unified representation of mechanical, electrical, hydraulic, pneumatic, thermal, and other systems in a unified representation. Using this approach, called the Genetic Programming/Bond Graph (GPBG) approach, they have tried to evolve systems with perhaps tens of components, but looking at only 100,000 or fewer design candidates. The GPBG system uses much smaller population sizes, but seeks to maintain diverse search by using "sustained" evolutionary search processes such as the Hierarchical Fair Competition principle and its derivatives. It uses stochastic setting of parameter values (resistances, capacitances, etc.) as a means of evolving more robust designs. However, in past work, the GPBG system was able to model and simulate only passive components and simple (voltage or current, in the case of electrical systems) sources, which severely restricted the domain of problems it could address. Thus, this paper reports the first steps in enhancing the system to include active components. To date, only three models of a transistor and one model of an operational amplifier (op amp) are analyzed and implemented as two-port bond graph components. The analysis method and design strategy can be easily extended to other models or other active components or even multi-port components. This chapter describes design of an active analog low-pass filter with fifth-order Bessel characteristics. A passive filter with the same characteristics is also evolved with GPBG. Then the best designs emerging from each of

these two procedures are compared. [The runs reported here are intended only to document that the analysis tools are working, and to begin study of the effects of stochasticity, but not to determine the power of the design procedure. The initial runs did not use HFC or structure fitness sharing, which will be included as soon as possible. Suitable problems will be tackled, and results with suitable numbers of replicates to allow drawing of statistically valid conclusions will be reported in this paper, to determine whether interesting circuits can be evolved more efficiently in this framework than using other GP approaches.]

Keywords: genetic programming, active component, transistor, bond graph, robust design strategy, Bessel analog filter design

1. Introduction

GP has been effectively applied to topologically open-ended computational synthesis (Koza et al., 2003). Though system performance is an important criterion, robustness, as the ability of a system to maintain its target performance even with changes in internal structure (including variations of parameters from their specified values) or external environment (Carlson and Doyle, 2002), is also critical to engineering design decisions. If the designed system is robust with respect to parameter values, it can probably still run well in a relatively harsh environment.

As there are many factors that affect system performance, it is difficult to take all system uncertainties or variability into consideration in robust engineering design. Two kinds of system robustness are often considered. One kind, widely investigated in robust engineering design (Du and Chen, 2000), is system robustness with respect to perturbation of component parameter values. Another kind is system robustness with respect to topology perturbation, such as component failure, short circuiting, etc. In this chapter, only robustness to parameter perturbation is considered.

This chapter considers evolution of a particular type of dynamic system, an analog low-pass filter with active components, as a design environment in which to do preliminary examination of some hypotheses about robust evolutionary design. The synthesis tool used throughout is called GPBG, a genetic programming (GP) system that uses trees to specify operations for construction of a bond graph (BG), which is a multi-energy-domain representation for dynamic systems. This GPBG system has earlier been used by the authors for automated design of a number of types of dynamic systems (Fan et al., 2001).

The chapter is organized as follows. Section 2 presents a short survey of robust design and introduction of evolutionary computation in this field. Section 3 discusses the GPBG methodology, which applies genetic programming and bond graphs for automated synthesis of dynamic systems. Bond graph modeling of common-emitter transistors and op amps is also discussed. Section 4

discusses topologically open-ended evolution and new active components and operators for robust design. Section 5 compares experimental results of these approaches. Conclusions and future research are discussed in Section 6.

2. Related Work

A method for robust design, called the Taguchi Method, pioneered by Dr. Genichi Taguchi, has greatly improved engineering productivity (Tay and Taguchi, 1993). After its introduction, it has been intensively studied in the community of engineering design (Zhu, 2001). In robust design, the control parameter settings are determined so the system produces the desired mean values for the performance, while at the same time minimizing the variance of the performance (Tay and Taguchi, 1993).

The most commonly applied system design methodology is the top-down procedure from system analysis, proceeding from functional design to detailed design. Within this methodology, robust design is most commonly treated during the detailed design phase. Design for robustness of system topology is normally not considered in this methodology. So the task of robust design is downgraded to parameter tuning and tolerance specification to maintain performance within acceptable limits. Topologically open-ended synthesis by genetic programming provides a way to move robust design forward to the conceptual/functional design stage and thus consider design for robustness from the very beginning, which will augment the current practice of design for robustness in practical design (Hu et al., 2005).

Application of evolutionary computation to robust design has been investigated since the early 1990s and can be classified into three categories (Forouraghi, 2000). The first type applies an evolutionary algorithm to parametric design for robustness. The second type focuses on evolving robust solutions in a noisy environment (Hammel and Back, 1994). A very active area of evolving robust systems is called evolvable hardware (Thompson, 1998). But most of these studies still separate the topology search and parameter tuning.

Two primary approaches to evolution of robust systems have been used by others: Robustness by Multiple Simulation ("RMS") and Robustness by Perturbed Evaluation ("RPE").

A common approach for evolving robust design is to use multiple Monte Carlo samplings with different environmental or system configurations (e.g., perturbation of parameter values of the system) to calculate a worst-case or an average fitness for a given candidate solution. This GP robust-by-multiple-simulation (RMS) method is used in (Branke, 2001), and in some of the experimental conditions reported here.

Another method is simply to add perturbations to the design variables before evaluation and evaluate a single, perturbed design. The perturbations, however,

are not incorporated into the genome, making it different from a "normal" parameter mutation operator or Lamarckian-style evolutionary algorithm. This robust-by-perturbed-evaluation (RPE) method is used in (Tsutsui and Ghosh, 1997) and is suggested to be more efficient by (Jin and Sendhoff, 2003). It is attractive because it uses only a single simulation run for evaluation of each design, relying on the fact that the design will persist and be re-evaluated again in future generations if it is a good one. However, it is not used in the study reported here.

3. Analog Filter Synthesis Using Bond Graphics and Genetic Programming

Bond Graphs

The bond graph is a modeling tool that provides a unified approach to the modeling and analysis of dynamic systems, especially hybrid multi-domain systems including mechanical, electrical, pneumatic, and hydraulic components (Karnopp et al., 2000). The explicit representation of model topology used in bond graphs makes them particularly good candidates for use in open-ended design search using genetic programming – for example, both series and parallel connections appear graphically as trees, unlike their conventional circuit-diagram representation. Complex electrical circuits and mechanical systems, or a synthesis of both, can be modeled as a tree structure using a bond graph, which is easy to evolve with one genetic programming tree. Bond graphs have four embedded strengths for evolutionary design application – namely, the wide scope of systems that can be created because of the multi- and inter-domain nature of bond graphs, the efficiency of evaluation of design alternatives, the natural combinatorial features of bond and node components for generation of design alternatives, and the ease of mapping to the engineering design process. Notation details and methods of system analysis related to the bond graph representation can be found in (Karnopp et al., 2000).

Bond Graph Modeling of Two-Port Active Components

In this section, a common-emitter transistor and operational amplifier are discussed. To represent the operational amplifier and transistor, we will extend the normal bond graph notation, as is frequently done (Karnopp et al., 2000), to include a double arrow to indicate a signal flow, rather than a bond. This allows the definition of controlled sources of effort, which are still one-port components from the bond graph perspective. The GPBG system is able to process the formulation of state equations using signal flows (forcing one of the associated variables to zero) as well as normal bonds.

The state models and parameters of the transistor and op amp components are fairly similar, and they are expected to play very similar roles in a GPBG-modeled circuit, so the initial experiments have been done with either common-emitter transistors or op amps in the designs, but not both. Figure 11-1 is the AC-equivalent-circuit model for the common-emitter transistor, and we can simplify this model to Figure 11-2.

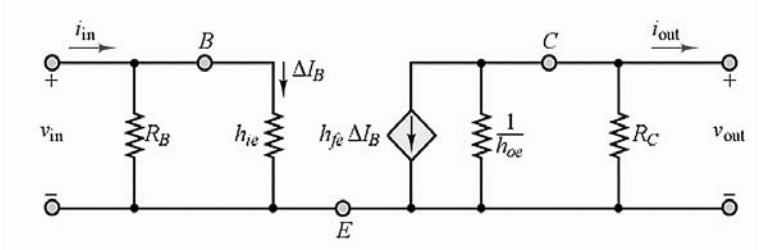


Figure 11-1. AC equivalent-circuit model for the common-emitter transistor

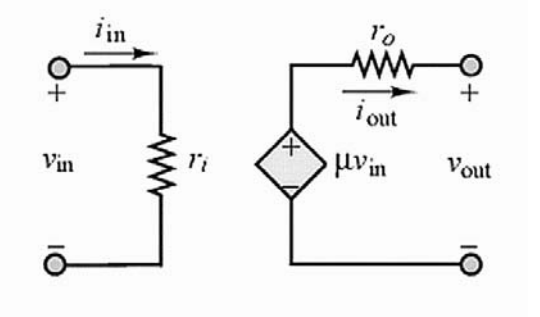


Figure 11-2. Simplified equivalent circuits for the common emitter amplifier

From this simplified circuit of a common emitter amplifier, we can derive, in Figure 11-3, the equivalent bond graph of the circuit of Figure 11-2.

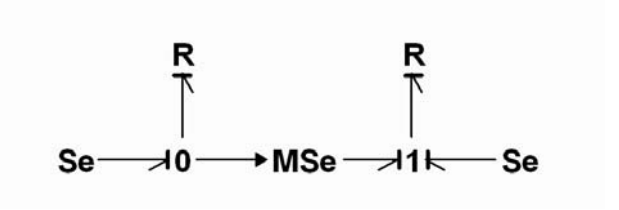


Figure 11-3. Equivalent Bond Graph of Common Emitter Transistor

Again using the signal-flow convention to define a controlled source of effort, we can draw in Figure 11-5 the bond graph equivalent of the circuit of Figure 11-4.

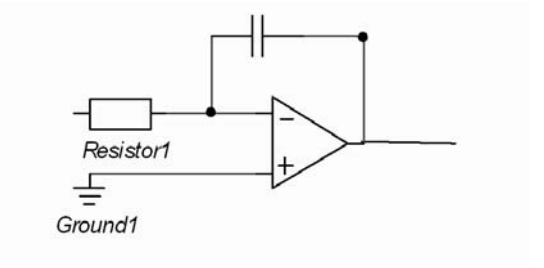


Figure 11-4. Electric Circuits with Operational Amplifier

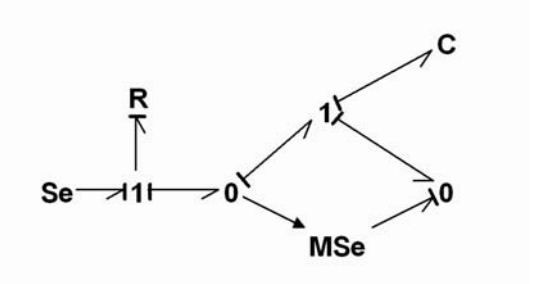


Figure 11-5. Bond Graph Model of Operational Amplifier

In Figure 11-3 and Figure 11-5, the models have a controlled source of effort and we can simplify the models' implementations in GPBG.

Standard components of bond graphs for design of active systems are the inductor (I), resistor (R), capacitor (C), transformer (TF), gyrator (GY), 0-junction (J0), 1-junction (J1), source of effort (SE), source of flow (SF), transistor (TR), and OPAMP. In the electrical context, a source of effort corresponds to a voltage source, and a source of flow, to a current source. For the use of op amps and transistors, we have also included the capability to process signals (represented with full arrowheads) as well as bonds. In this chapter, we concentrate our discussion on active analog filter design, and the resulting bond graphs will be composed of only I, R, C, SE, SF, TR, OPAMP components; however, in this initial study, we will not use both the transistor and operational amplifier in the same bond graph, as the particular (two-port) bond graph transistor model implemented here does not provide capabilities beyond that of the operational amplifier.

Combining Bond Graphs and Genetic Programming

The problem of automated synthesis of bond graphs involves two basic searches: the search for a good topology and the search for good parameters for each topology, in order to be able to evaluate its performance. Building upon Koza's work on automated synthesis of electronic circuits, we created a developmental GP system for open-ended synthesis of mechatronic systems represented as bond graphs. It includes the following major components: (1) an embryo bond graph with modifiable sites at which further topological operations can be applied to grow the embryo into a functional system, (2) a GP function set, composed of a set of topology manipulation and other primitive instructions which will be assembled into a GP tree by the evolutionary process. Execution of this GP program leads to topological and parametric manipulation of the developing embryo bond graph, yielding a final bond graph, and 3) a fitness function to evaluate the performance of candidate solutions. Implicit in the system is the capability to use the bond graph generated to formulate the state equations of the system to allow its simulation or analysis, permitting assessment of its performance (i.e., fitness evaluation).

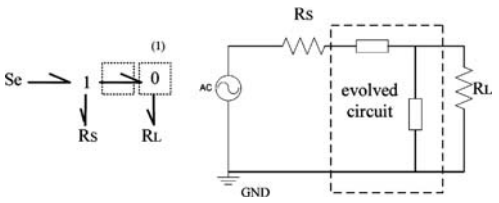


Figure 11-6. Embryo bond graph and its corresponding electric circuit.

In developmental GP, an embryo is used as the root of the GP tree, and is often used to guarantee that each tree contains the minimum structure to allow evaluation of its fitness. For this example, as shown in Figure 6, the embryo assures that each circuit has a voltage source at which the input is applied, a source resistor, and a load resistor across which the output of the filter can be measured. In the GPBG system, the GP tree does not represent the bond graph directly, but is instead a tree-structured program for construction of a bond graph, beginning with the embryo as the root. Figure 11-7 shows a bond graph construction tree, but the details (Fan et al., 2001) are not needed to understand the experiments described here.

Choosing a good function set for bond graph synthesis is not trivial. In our earlier work, we used a very primitive "basic" function set, and later, we developed the following hybrid function set to reduce redundancy while retaining good flexibility in topological exploration:

$$F = \{Insert\_J0E, Insert\_J1E, Add\_C/I/R, EndNode,$$
$$Insert\_Transistor/Insert\_OPAMP, EndBond, ERC\}$$

Figure 11-7 shows a GP tree that specifies how a complete bond graph solution is constructed from the embryo bond graph.

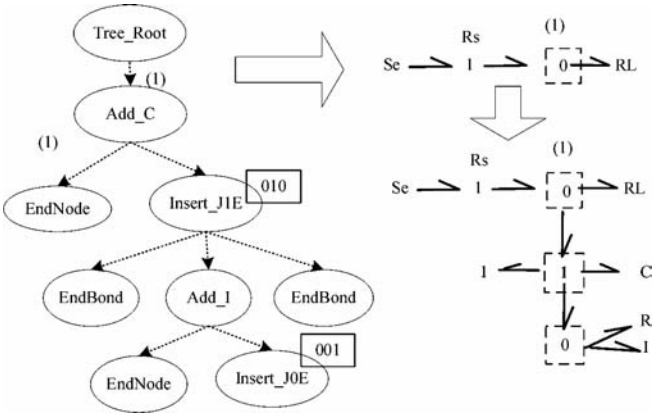


Figure 11-7. A sample GP tree (left), composed of topology operators applied to an embryo (tree_root and bond graph on right), generating a bond graph (lower right) after depth-first execution (numeric nodes omitted).

In this paper, we introduce two new functions: Insert_Transistor and Insert_OPAMP. As was discussed in the previous section, because of the similarity between the common emitter model of the transistor and the operational amplifier, we will use only one generic, active two-port component at a time.

The Example Lowpass Filter Problem

In this study, a lowpass filter with fifth-order Bessel characteristics is to be synthesized. We say a lowpass filter of Bessel characteristics, rather than a Bessel lowpass filter, because a Bessel filter is designed with a strict mathematical equation and a well-defined synthesis procedure, while we simply used the fifth-order Bessel filter magnitude and phase frequency response as reference for design fitness evaluations. Figure 11-8 shows a design of a typical 5th-order Bessel lowpass filter.

In this GPBG-based filter design problem, a bond-graph-represented analog filter composed of capacitors, resistors, inductors, and transistors or operational amplifiers is to be evolved such that the magnitude and phase of its frequency response approximate the Bessel filter frequency response specification. This

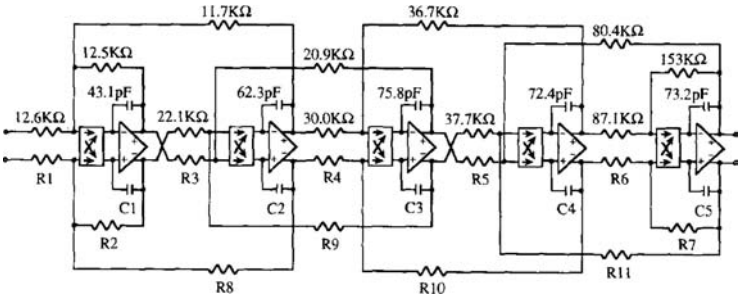


Figure 11-8. 5th-order Bessel Filter.

procedure will not use the sophisticated (and relatively time-intensive) SPICE simulation program, as is typically done in analog circuit analysis. Instead, the frequency response of the circuit modeled by the bond graph can be calculated in a faster and more convenient way: first, the state equation of the bond graph is automatically derived from the model, yielding the A, B, C, and D matrices of linear system theory. The frequency response of the state-space model is then calculated on a Linux PC using C++ simulation code generated by the Matlab 3.0 compiler.

The detailed specifications of the lowpass filter problem addressed here are as follows: the frequency response performance of a candidate filter is defined as the weighted sum of deviations from ideal magnitude and phase frequency responses evaluated at 101 points:

$$F_{magnitude}(t) = \sum_{i=0}^{100} [W(d(f_i), f_i) * d(f_i)] \tag{11.1}$$

$$F_{phase}(t) = \sum_{i=0}^{100} [W(d(f_i), f_i) * d(f_i)] \tag{11.2}$$

The definition of the frequency response magnitude is the same as in our earlier study [Hu, 2004]. However, in this study, we also include frequency response phase in the calculation of fitness, where f_i is the sampled frequency, $d(x)$ is the absolute deviation of candidate frequency response from target response at frequency x, and $W(x, y)$ is the weight function specifying the penalty level for a given frequency response at a specified frequency range. The sampling points range from 1Hz to 100 KHz, evenly distributed on a logarithmic scale. If the deviation from ideal phase is less than 30 degrees, the weight is 1. If the deviation is more than 30 degrees, the weight is 10, aimed at reflecting the relatively small importance of small deviations in the phase response from

ideal. The pass band is [1, 1k] Hz, the stop band is [2K, 10K] Hz. The phase is weighted as 0.1 and magnitude is weighted 0.9 in the final fitness calculation. An alternative could be to treat magnitude and phase as separate objectives in a multi-objective search, but our design here is simple and produces an acceptable result. If we want to have stringent control of the phase, we need to consider other alternatives, but that was not seen as critical to this study.

Before introduction of any robustness considerations, the fitness function is defined as follows. First we calculate the raw fitness defined as the average absolute deviation between the frequency response magnitude and phase of the candidate solution and the target frequency response over all 101 sampling frequencies.

$$f_{raw} = \frac{1}{101}(0.9 * f_{magnitude} + 0.1 * f_{phase}) \tag{11.3}$$

$$f_{norm} = \frac{NORM}{NORM + f_{raw}} \tag{11.4}$$

Differences from the Usual GP System

The GPBG system includes the following "non-standard" features:

- A flag bit mutation operator is introduced to evolve the configuration of C/I/R elements attached to a junction. That is, junctions introduced into a bond graph by the Insert_J0E or Insert_J1E operators may each have zero or one C, I, and R elements, as specified by three binary flags (rectangles in Figure 11-2 are an example). A special mutation operator can manipulate those flag bits at the junctions.

- A subtree-swapping operator is used to exchange non-overlapping subtrees of the same individual (GP tree).

- A Gaussian ERC mutation operator, as is commonly used in evolution strategies, is developed to evolve the parameter values of all C/I/R components; the value generated for the ERC is arbitrary within the range specified for each component.

Elitism of one individual is used throughout the evolution process – that is, the best individual in the population is always preserved to the next generation.

Except for the above, the GPBG system as used in this paper is a standard strongly-typed multi-population generational GP. The running parameters are specified in Section 5.

4. Evolving Robust Active Analog Filters Using Bond Graphs and Evolutionary Algorithms

This section examines the design of analog filters for robustness to parameter variations, There are many ways this might be attempted using GP as an open-ended topological search tool.

GPRMS

For the GPRMS multi-simulation method, the raw fitness for a design solution, including a robustness criterion, is defined as the sum of a number NS (here, 10) of (here, 101-point-) deviation sums from the target frequency response curve, resulting from NS filter simulations of the same design:

$$f_{robustraw} = \sum_{k=1}^{NS} f_{raw}^{k} \tag{11.5}$$

where NS is the number of Monte Carlo sampling evaluations (filter simulations) for each individual, and f is the raw fitness of the kth sampled evaluation with a different Monte Carlo perturbation of the parameters, as defined in Equation 11.3. With this raw robustness from Equation 11.5, we then calculate the final fitness similarly to Equation 11.4

5. Experiments and Results

In this section, a series of experiments is conducted to verify the effectiveness of introducing active components into robust design of an analog filter by genetic programming. In these experiments, the perturbation of the component values during evolution is implemented by adding to each component's parameter(s) Gaussian noise $N(\mu, \sigma)$ with mean $\mu = 0$ and standard deviation σ set at 20% of the parameter value. This perturbation model has been widely used in previous research, and while it may not be an accurate model of component variation (introducing more than is typically present in the components), any excess noise may also be useful in discovery of robust solutions. One difficulty with this definition is that if the original parameter value is zero, then no perturbation will be generated. Although this is rare in evolutionary experiments, it is alleviated here by checking for any component value of zero, in which case the standard deviation for the perturbation is set to 1.0.

In the robust design of GPRMS (using the robustness by multiple simulations approach), multiple simulations (in this case, NS = 10) are used to evaluate the fitness of each single design. For this filter design problem, the computation budget is 1,000,000 simulations, so up to 100,000 different designs can be evaluated in each run. Runs in which best performance fails to advance for X

generations are terminated automatically with fewer than 100,000 individuals evaluated (i.e., fewer than 1,000,000 simulations).

While only one parameter perturbation model was used during the evolutionary synthesis experiments – Gaussian noise $N(\mu, \sigma)$ with mean μ of 0 and standard deviation σ set at 20% of the parameter value, the later (post-run) robustness evaluations of the evolved filters include multiple perturbation magnitudes with extensive simulation.

To assess the statistical significance of the performance differences between these methods, 15 runs were done for each synthesis method. The size of these experiments was determined by the computing resources available. However, since the results were found to be quite stable across multiple runs, this level of replication appears to be sufficient for the purpose of this preliminary study.

All experiments described below used the same embryo bond graph shown in Figure 11-6. The component values of source resistor R_s and load resister R_{load} are both 1Ω for the lowpass filter with Bessel characteristics.

The following sections first describe separately the experimental configuration of each method, the best evolved bond graph model of the filter, and the magnitude and phase responses of the best solution from each method. These results provide some general ideas regarding how robustness is evolved with respect to the parameter perturbations. Then a statistical comparison of the performance is presented. The following common running parameters (Table 11-1 were used throughout all GP experiments in this chapter.

Table 11-1. Shared parameters of experimental runs.

Total population size: 2000	Crossover probability: 0.4
MaxDepth: 10	Standard mutation probability: 0.05
InitDepth: 3-5	Parametric mutation probability: 0.3
Tournament size: 2	Flag mutation probability: 0.3

At the end of the run, the robustness of each final evolved solution was evaluated against a series of perturbation magnitudes: Gaussian noise $N(\mu, \sigma)$ with mean μ at 0 and standard deviation σ at 10% to 50% of parameter values, in steps of 10%, each tested with 5000 samplings with different configurations of the component parameter perturbations.

Below, we display the evolved filter with the highest performance from each of the three run types (passive without perturbations during evolution, passive with perturbations during evolution, and active with perturbations during evolution), to test its noise tolerance in the face of degradation or variation of the component parameters.

Evolving Robust Passive Analog Filters Using Genetic Programming: Open-Ended Topology Innovation for Robust Design

In the experiments, analog lowpass filters with 5th-order Bessel characteristics were evolved using GPRMS and incorporating a robustness criterion (Equation 11.5) in the fitness function Equation 11.3, in two ways: 15 were evolved using only passive components and 15 were evolved also allowing active components – either common-emitter-modeled transistors or op amps.

From Figure 11-9 first, one can see that the frequency response magnitude in the plot of the filters evolved, when evaluated according to the criterion of Equation 11.1, resembles the target, but appears to represent a higher-order filter (with a sharper falloff). It is expected that further evolution might produce characteristics more similar to those of the 5th-order Bessel filter. The observation is that introducing a robustness requirement does not necessarily decrease the performance with nominal parameters significantly. However, the phase response deviates noticeably, particularly at high frequency. A possible explanation is that the weight of the phase term in the fitness function was only 10%, compared to 90% for the magnitude term. This weighting was used to reflect the differential importance of amplitude and phase for most applications. From the perspective of system performance, the phase delay is not usually a critical objective in the real application, often being set instead as a constraint with no effect on fitness so long as it does not fall outside a specified range. However, its performance is contrasted with that of GP and in the statistical comparisons below.

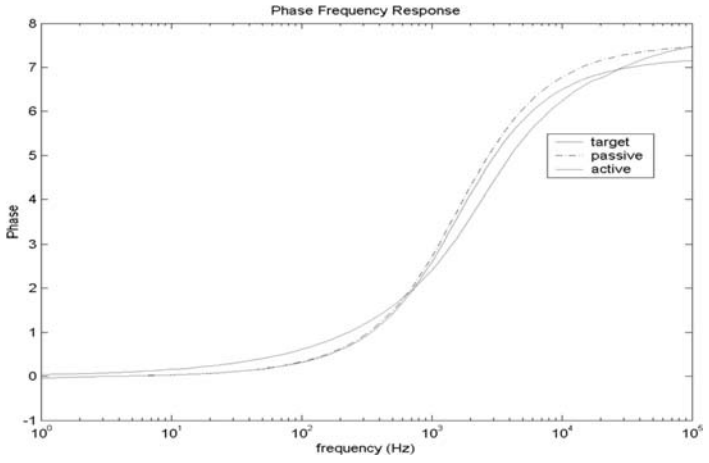


Figure 11-9. Magnitude and Phase Frequency Responses.

Evolving Robust Active Analog Filters

In the second set of runs, robust analog filters with active components are evolved that have higher tolerance to the variations of component values. Figure 10 shows the best filter evolved with only passive components and Figure 11 shows the best evolved with active components. This filter uses far fewer components than the filter evolved only using passive components, while its functional performance remains similar. The robustness of this filter is next compared to that of the best filter evolved with only passive components.

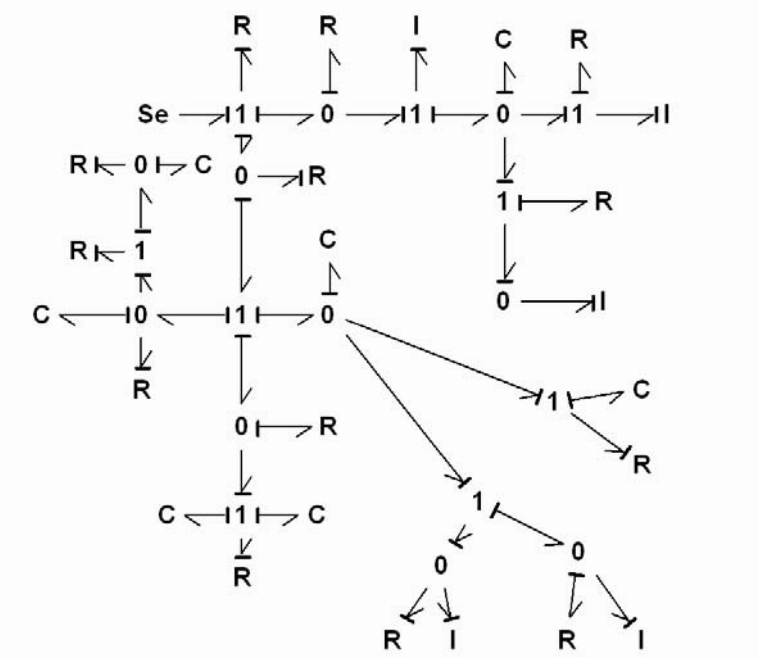


Figure 11-10. The best gond graph evolved with GPRMS and only passive components.

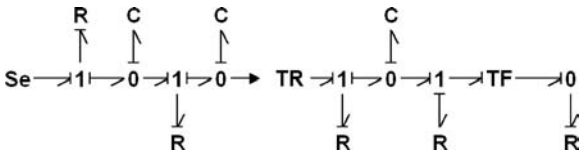


Figure 11-11. The best bond graph evolved with GPRMS with active components available

Statistical Comparisons of the Three Methods

Hypothesis 1: (re passive filters) Systems with similar "base" (unperturbed) performance can be evolved if robustness is considered during the evolutionary process, without the need for a much larger number of function evaluations than is needed for conventional (non-robust) synthesis. A t-test on the differences in performance of the filters evolved by GPRMS and by plain GP, evaluated with their nominal (unperturbed) parameter values, did not reveal any significant difference (e.g., $P > 0.20$). They were evolved using an identical number (1 million) of filter simulations.

Hypothesis 2: (re passive filters) Introduction of noise during the process of evolution improves the robustness over the "plain GP" results: a t-test on the results of the lowpass filter problem was used to compare the robustness of the evolved solutions by GPRMS and standard GP in terms of fitness at the 0.2 (20%) perturbation level. A significance level of $P <= 0.001$ was achieved; strongly indicating that GPRMS improved the robustness over the filters evolved by plain GP, using the same number of filter evaluations.

Hypothesis 3: (for active vs. passive filters) Introduction of active components during the process of evolution allows the circuit size to decrease without loss of performance or robustness. A t-test was done on the results from the lowpass filter with passive components and active components to compare the robustness of the evolved solutions in terms of fitness at the 0.2 perturbation level. It did not reveal any significant difference (e.g., $P > 0.20$), which indicates that the decrease in size (number of components) observed for the active filters did not decrease the robustness of the filters evolved.

6. Conclusions and Future Work

This chapter exploits the open-ended topological search capability of genetic programming to conduct preliminary studies of robust design of dynamic systems with active components. The common emitter transistor model and operational amplifier are represented using bond graphs with a signal-flow extension, and are implemented as new components in the GPBG system for evolutionary design. This extended system is used to test hypotheses involving the use of topological innovation in the conceptual design stage to improve the robustness of the systems evolved. Specifically, GPBG in the RMS (robustness by multiple simulations) framework is used to design analog filters of high robustness. Evolving robustness is a rich research theme and there are several interesting topics to be further investigated. This experiment is the only the first step enabled by addition of active components to the GPBG system, broadening the scope of design problems for which it can be used.

References

Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Kluwer.

Carlson, J. M. and Doyle, J. (2002). Complexity and robustness. In *Proceedings of National Academy of Science (PNAS)*, volume 1, pages 2538–2545.

Du, X. and Chen, W. (2000). Towards a better understanding of modeling feasibility robustness in engineering design. *ASME J. Mech*, 122(4):385–394.

Fan, Zhun, Hu, Jianjun, Seo, Kisung, Goodman, Erik D., Rosenberg, Ronald C., and Zhang, Baihai (2001). Bond graph representation and GP for automated analog filter design. In Goodman, Erik D., editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 81–86, San Francisco, California, USA.

Hammel, U. and Back, T. (1994). Evolution strategies on noisy functions: how to improve convergence properties. In *Proceedings of Parallel Problem Solving from Nature*, volume 3, pages 159–168.

Hu, J., Goodman, E.D., Seo, K., fan, Z., and Rosenberg, R. (2005). The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(1).

Jin, Y. and Sendhoff, B. (2003). Trade-off between optimality and robustness: An evolutionary multi-objective approach. In Fonseca, C., editor, *In Proceeding of the Second Int. Conf. on Evolutionary Multi-Criterion Optimization*, pages 237–251. Spinger.

Karnopp, D.C., Margolis, D.L., and Rosenberg, R.C. (2000). *System Dynamics: Modeling and Simulation of Mechatronic Systems*. John Wiley and Sons, Inc., New York.

Koza, John, Keane, Martin, Streeter, Matthew, Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003). Routine human-competitive machine intelligence. In *Genetic Programming IV*. Kluwer Academic Publishers.

Tay, E. and Taguchi, W. (1993). *Taguchi on Robust Techology Development: Bringing Quality Engineering Upstream*. American Society of Mechanical Engineering Press, New York.

Thompson, A. (1998). On the automatic design of robust electronics through artificial evolution. *International Conference on Evolvable Systems*, pages 13–24.

Tsutsui, S. and Ghosh, A. (1997). Genetic algorithms with a robust solution searching scheme. *IEEE Trans. Evolutionary Computation*, 1(3).

Zhu, J. (2001). Performance distribution analysis and robust design. *Journal of Mechanical Design*, 123(1):11–17.

Chapter 12

TRUSTABLE SYMBOLIC REGRESSION MODELS: USING ENSEMBLES, INTERVAL ARITHMETIC AND PARETO FRONTS TO DEVELOP ROBUST AND TRUST-AWARE MODELS

Mark Kotanchek[1], Guido Smits[2] and Ekaterina Vladislavleva[3]
[1]*Evolved Analytics,LLC, Midland, MI, USA;* [2]*Dow Benelux B.V., Terneuzen, the Netherlands;* [3]*Tilburg University, Tilburg, the Netherlands.*

Abstract Trust is a major issue with deploying empirical models in the real world since changes in the underlying system or use of the model in new regions of parameter space can produce (potentially dangerous) incorrect predictions. The trepidation involved with model usage can be mitigated by assembling ensembles of *diverse* models and using their consensus as a trust metric, since these models will be constrained to agree in the data region used for model development and also constrained to disagree outside that region. The problem is to define an appropriate model complexity (since the ensemble should consist of models of similar complexity), as well as to identify diverse models from the candidate model set.

In this chapter we discuss strategies for the development and selection of robust models and model ensembles and demonstrate those strategies against industrial data sets. An important benefit of this approach is that all available data may be used in the model development rather than a partition into training, test and validation subsets. The result is constituent models are more accurate without risk of over-fitting, the ensemble predictions are more accurate and the ensemble predictions have a meaningful trust metric.

Keywords: Symbolic regression, Pareto optimality, trust metrics, ensembles, confidence, robust solutions

1. Introduction

The problem with empirical models

Data-driven models are important in real-world applications since in many cases first-principle models either are not possible or practical, because of an absence of valid theory, complexity of input interactions, execution time requirements of a first-principles model or a lack of fundamental understanding of the underlying system. In these situations, a data-driven model is the only viable option to infer the current state of a critical variable, make predictions about future behavior, emulate the targeted system for optimization, or extract insight and understanding about driving variables and their influence.

Unfortunately, there is a problem: *Most empirical models are big piles of "trust me"*.

There is a fundamental limitation of data-driven models in that they are only 100% valid (assuming noise-free data) at the points at which there is data. Since there are an infinite number of models which will perfectly fit a finite data set, we typically impose a preference for simplicity (parsimony), impose a model form or use additional (test and/or validation) data sets to make sure that the model is valid at some other regions of parameter space and hope for the best in using the model.

Alas, these models are NOT valid if used outside the region of parameter space used for the model development ,and possibly not valid within that region, if the underlying model dynamics have changed, or if the model was over-fitted to the data. There is no easy way to detect that a developed model should not be trusted. Inappropriate use of an invalid model can be dangerous – either physically, financially, or both.

The symbolic regression-centric approachmodels

Conventional symbolic regression does not have an inherent advantage for developing trustable models. However, there are three pillars which do support that development:

- Pareto-aware symbolic regression algorithms to develop models of appropriate complexity,
- Interval arithmetic for identifying robust models, and
- Ensembles of diverse models to provide a trust metric.

Together these pillars support developing robust and trustable models – which is a unique and very significant capability for empirical models.

Pareto-aware symbolic regression. Pareto-aware symbolic regression algorithms (Kotanchek et al., 2006) which explicitly explore the trade-off between

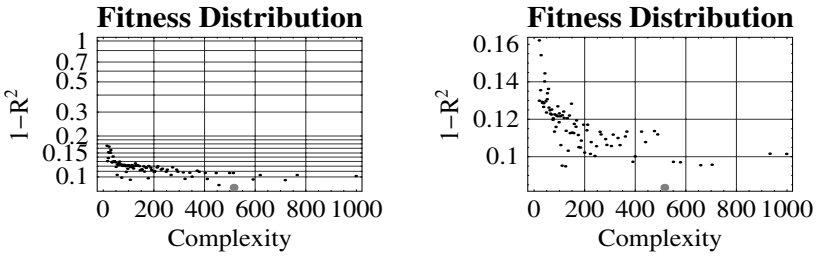


Figure 12-1. Here we show the Pareto front trading off accuracy (1-R^2) complexity for developed inferential sensor models. Note the knee of the curve indicates the point of diminishing returns and is the region from which we would likely prefer to retrieve our models.

model complexity and accuracy allow us the luxury of selecting models which provide the best balance between model accuracy and complexity – with the implicit assumption that overly complex models are at risk of being over-fitted and having pathologies. This behavior is illustrated in Figure 12-1, which shows the distribution of developed models against these two criteria. (The symbolic regression algorithm we use rewards models for being near the Pareto front – which is why the population of models are banded as shown.).

A benefit of being able to define a region of model fitness (accuracy vs. complexity) space which provides the best return on model complexity is that we can use ALL of the data in the model development – which enables quality model development even for systems with small or fat data sets. (This bold statement will be supported through the course of the rest of the chapter.)

Robust model development and identification. Millions of models will be explored in a typical symbolic regression effort. Even if we restrict our attention to the most attractive models, we could have tens of thousands of models that meet our nominal accuracy requirements, and are in the region at the knee of the Pareto front. From this abundance, we need to select models which are also robust – in the sense that they do not contain numerical pathologies which would cause operational problems – as well being accurate and not over-fitted.

Searching nonlinear models with multiple variables is a very computationally intensive exercise and does not have a guarantee of actually finding singularities. Maarten Keijzer (Keijzer, 2003) proposes an alternative based upon interval arithmetic which is very attractive due to its relative speed. This approach may be overly conservative, since some parameter combinations which cause singularities may not be achievable in the real-world due to variable coupling. Alternately, the existence of a pathology may actually be appropriate for some variable situations.

Rather than selecting models for robustness in post-processing, this criterion may be included during the evolutionary development. To some extent we can guide the development by either judicious choice of function building blocks or by eliminating ill-structured models as they are developed. Alternately, a nonlinearity metric can be included in the Pareto-based selection process. Since the efficiency of the multi-objective selection breaks down as the dimensionality of the objectives is increased, an attractive alternative is to use alternating fitness metrics wherein the parsimony aspect (e.g, model complexity, model nonlinearity, etc.) is switched each generation. Of course, the accuracy metric (1-R^2, scale-invariant noise power, norm, etc.) can also be toggled in this algorithmic variant. This approach has been shown (Vladislavleva and Smits, 2007) to improve both the efficiency of the model development, as well as the robustness of the models.

Diverse ensembles for accuracy plus trust. Diverse and independent models will be constrained to agree where there is data and, to a large extent, constrained to disagree away from those constraint points. Obviously, the degree of model divergence will depend upon how far the prediction point is away from the data points. Therefore, the consensus within an ensemble (i.e., a collection of diverse quality models) can implicitly detect extrapolation even in high-dimensional parameter spaces. This is a very unique capability as well as very valuable in real-world applications as a trust metric. Similarly, if the modeled system undergoes fundamental changes, the ensemble models will likely also diverge in their predictions which provides an early warning and awareness that would not otherwise be possible.

One of the keys to a good ensemble is that the models are of similar complexity as well as similar accuracy. The multi-objective Pareto front perspective, therefore, is very important in identifying models for possible inclusion.

Arguably the easiest way to identify diverse models is to look at the correlation of their error residuals and pick a diverse set based upon a joint lack of significant correlation. However, because the developed models share a common reference they will tend to be correlated; otherwise, they would not be the quality models that are wanted. Noisy data will increase that correlation since models will tend to navigate through the center of the fuzzy observed response surface. Because of these two factors, the definition of acceptable levels of correlation needs to be adjusted somewhat.

Although use of ensembles is relatively novel in symbolic regression and genetic programming, they have been a major factor in the industrial success of stacked analytic networks (Kordon et al., 2006) for fifteen years. Climate and meteorological prediction has also embraced ensembles to provide a confidence in weather event prediction (Hamill, 2002). Finally, the machine learning community is beginning to use ensembles in a classification framework (Wichard,

2006). Similar perspectives have also been used in stock portfolio definition (Korns, 2006).

Classic approaches to resolving this conundrum

We have a conundrum. On the one hand, we need a model and we must derive it from the available data and, on the other hand, we suspect that we cannot completely trust the developed models. As we shall see, there is a possible solution via a judicious selection of multiple symbolic regression models; however, let us first review how this problem is addressed by other modeling and machine learning approaches: linear statistics, neural networks and support vector machines.

General foundations for a trustable model. There are a number of data characteristics which make development of a trustable model easier:

- **Representative data** – the data represents the current (and future) response behavior;
- **Balanced data** – the data captures the dynamics and does not unduly represent any region of parameter space;
- **Significant inputs** – nuisance variables are not included in the data set;
- **Abundant data** – this enables coverage of the parameter space as well as definition of model validation data sets.

These ideals are often not achievable in real-world situations. Most systems of interest undergo changes with time either slowly (e.g., parts wear or the economy shifts) or in a step change (e.,g., a down-stream condenser is removed). We may also want the developed model to be transferrable to a similar but different system (e.g., to recommend treatment plans given a set of medical diagnostic tests on a new patient). If the system is in our control, we may be able to run a designed experiment to attempt to generate a balanced data set. However, that may not be an option if there are many variables and we do not know which are the truly significant ones, or if production or safety constraints limit the range or number of experiments which can be executed. If the system is not under our control, we can, obviously, not perform what-if experiments and we are constrained to the data stream that is captured.

Linear Statistics & Trustable Models. The classic linear statistics approach of a control chart is really only applicable to constant output and suffers from a latency problem in that new data must come in and be recognized as different from the expected behavior before action can be taken. If the charted

variable changes over time, this makes the recognition of a model failure much more difficult.

Another limitation is an implicit assumption that the variables used in the model be uncorrelated. To some extent, this can be detected by examining the variance inflation factors (VIF); however, that tends to be a labor-intensive operation. The easiest way to avoid this issue is to use a principle components analysis to identify orthogonal variables as inputs. The downside of this approach is that the resulting models may lose interpretability and the construction of the input variables implicitly assumes a linear coupling of the input variables.

In each iteration of linear model building, we assume an *a priori* a model structure and look for the coefficients which best fit that model to the data. If the assumed model structure matches that of the targeted system, then we have a high quality model. If not, then we must revise the model structure and try again. Although this iterative process is numerically efficient, it can be quite inefficient from a human time standpoint. As such, we are generally restricted to searching low-order polynomials both from an assumption that simple models will be more robust as well as the human effort required to interactivity explore and refine candidate model structures; unfortunately, a mismatch between the true and mathematically convenient models can lead to accuracy and robustness problems[1].

Neural Networks & Trustable Models. Our goal in data modeling is to model the underlying system and not any noise which might be also present in the available data. Since the amount of noise in the data is not known *a priori*, the available data is partitioned into training, test and (possibly) validation subsets. The traditional neural network (NN) approach assumes a model structure and does a (nonlinear) search for the coefficients which best fit the model to the training data with the search being stopped when the performance against the test set degrades since, presumably, at that point the neural network is starting to model the noise rather than the system fundamentals. As an additional check, a validation set may be used to discriminate between developed NNs for the final selection. (Since the coefficient search is a very nonlinear optimization problem, the model coefficients – and, hence, model response behavior – which change with each randomly initialized training run even if the model structure is the same.)

Parsimony also is a factor for NN selection in that we prefer simple model structures. Towards that end, we generally include a direct connection between the input and output nodes as well as via the hidden layers. This produces an

[1]There is a synergy between linear statistics-based modeling and symbolic regression via genetic programming via the discovery of linearizing transforms (i.e., identifying metavariables – variable combinations or transforms) which allow the implicit mathematical assumptions to be satisfied, (Castillo et al., 2004)

underlying linear model which means that the NN has an easier time discovering the global response surface as well as providing a linear response behavior when the model is asked to extrapolate outside the training realm. Because of the preference for parsimony, NN development algorithms will suppress low-significance connections or iteratively evolve network structures in an attempt to achieve robustness.

As with the linear models, identifying that the model should not be trusted relies upon *post facto* assessment of model performance against new data which implies a corresponding lag on detection and response to system changes or new operating regimes. Additionally, if data is scarce, partitioning the data into viable training and test sets becomes an issue. Data balancing also becomes a serious issue in defining the various data subsets.

Support Vector Machines & Trustable Models. A support vector machines (SVM) or, more specifically in this case, support vector regression (SVR) identifies the key data points (vectors) in a data set and builds a response model by placing kernels (e.g., polynomials or radial basis functions) at those points. The predicted response is, therefore, the cumulative contribution of the kernels located at each of the support vectors. Judicious kernel selection can lead to models which are both accurate and extrapolate reasonably well.

The problem still remains that system changes or new operating regimes can only be detected *post facto*. SVR also suffers from the curse-of-dimensionality in that inclusion of spurious inputs can cause robustness problems.

Summary on non-symbolic regression approaches. All of these modeling techniques can produce excellent models for static situations where the operating region of the parameter space is well covered by the data used in the model development, the operating region does not change over time and the targeted system does not change over time. They all will tend to struggle if correlated inputs or spurious inputs are used in the model development. None will provide an assessment of deployed model quality except via after-the-fact analysis.

2. Ensemble Definition and Evaluation

In this section we walk through the process of model development and the definition and use of ensembles built from those models.

Developing diverse models

If the models used to define an ensemble are minor variations on a theme, they can all happily agree as they guide the user to march off a cliff. Hence model diversity is a critical ingredient in a successful ensemble. We have a

number of mechanisms we can use to assist the symbolic regression processing to produce diverse models:

- **Independent runs** – due to the founders effect as well as randomness, each evolutionary exercise will explore a different trajectory in the model search. The more searches we execute, the more likely we are to develop diverse models.
- **Different rescale ranges** – although GP can handle model building with data in the natural data ranges, we can often improve the efficiency of the model building process (sometimes very significantly) by rescaling input variables to a common range. The choice of range will affect both the ease of quality model discovery as well as structure of the developed models. Thus, model building with different rescale ranges helps to produce diverse model structures.
- **Use different subsets** – if data is abundant, we can use different data subsets in model development. This is related to the ordinal optimization approach to developing robust models wherein different subsets are used for each generation within a single evolution (Kotanchek et al., 2006), (Smits and Vladislavleva, 2006), except that here we are using a different subset for each evolution but maintaining that subset throughout the processing.
- **Use different functional building blocks** – obviously the choice of functional building blocks will influence the structure of the developed models. Hence, using a variety of function sets will produce a variety of model structures.
- **Change supplied variables** – Pareto-aware symbolic regression is quite powerful in that it will automatically select the most important variables – even of the supplied variables are correlated. However, for the purposes of an ensemble, we may wish to include intentionally myopic models which feature sub-optimal variables since they can contribute to overall robustness as well as to the quality and accuracy of the trust metric (Korns, 2006).

Other than independent evolutions, the above mechanisms may or may not be important for developing independent candidate models. The relative importance of each diversity introduction mechanism as well as best-practices for their use is an open research area.

Selecting candidate models

Assuming that a large number of (hopefully diverse) models have been developed, we are now faced with the problem of reducing this abundance to

a manageable number as a precursor to the actual ensemble development. A sequence which we have found to be effective is:

1. Identify a rectangular region in fitness space which includes the knee of the Pareto front and select the models which lie within that region,
2. Select the robust models from that subset using an interval arithmetic test over the nominal data ranges (or an extension of that range, if extrapolation performance is important)
3. Select a manageable size subset from the robust models based upon their adjacency to the Pareto front. We will typically select between 500 and 3,000 models.

In Figure 2.0 we show the results of such a model down-selection processing. Although algorithmically simple and efficient, this step is quite important since the quality of the final ensemble will depend upon the quality of the candidate models. Unfortunately, definition of the region from which to draw the models as well as the number of final models to be included in the candidate set varies from problem to problem and requires engineering judgement as to what is appropriate.

Defining ensembles

Rather than selecting THE model from the candidate model set, our goal is to select a diverse ensemble of models. Diversity could be defined a number of different ways:

- Model **structure**,
- Constituent **variables** embedded within the model,
- Lack of prediction error **correlation**, etc...

Unfortunately, although intuitively pleasing, the first two of the above are difficult to quantify. Thus, our approach is to use a lack of residual correlation as the criteria for the ensemble definition with the implicit expectation that models which satisfy that criteria will also satisfy the others.

Algorithms to identify uncorrelated model sets. Using the relative lack of error correlation is an N^2 scaling problem which rapidly becomes intractable as the number of models increases. We generally use a divide-and-conquer approach wherein we randomly partition the models into manageable size subsets ($\sim$100 models) apply a selection algorithm:

1. Build the covariance matrix from the error residuals of the supplied models.

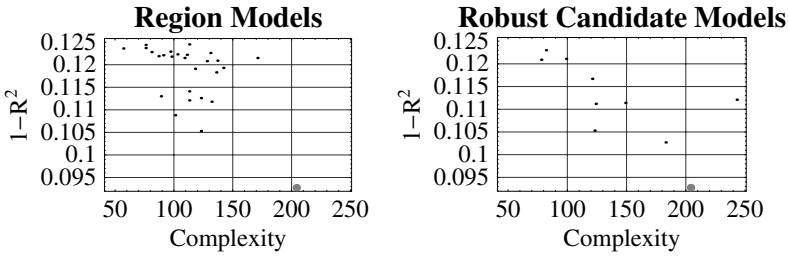


Figure 12-2. Here we show the result of focusing on a region of the fitness landscape for the inferential sensor. Note that by selecting models based upon their adjacency to the Pareto front, we can suppress the inclusion of models with a relatively high complexity and relatively low accuracy. In this case, the number of models went from 2,587 in the rectangular region to 1,576 robust models in the region to 817 in the subset used for ensemble definition. The robustness test selected models which did not have any interval-arithmetic-based pathologies on a ±50% expansion of the nominal training parameter ranges.

2. Select the most uncorrelated model pair that is less than a specified correlation threshold.
3. Select the most uncorrelated model relative to the previously selected models which meets the specified correlation threshold.
4. Repeat step 3 until no models remain which satisfy the correlation threshold.
5. If no models meet the independence threshold, return the most typical model based upon an eigenvalue analysis.
6. Merge the models returned from the subsets and repeat the algorithm to return the final set of uncorrelated models.

As inferred previously, the definition of uncorrelated needs to be adjusted relative to the standard linear statistics threshold of a 30% or less correlation. The default value we use for a threshold is 80% or less correlation; however, this threshold must generally be adjusted based upon the size of the data set as well as the amount of noise in the data set.

An alternate algorithm that is also very efficient is to:

1. Select a reference model,
2. Calculate the correlations of all models with respect to that reference and choose the model with the absolute lowest correlation with the reference model,
3. Re-calculate the correlations of all of the models relative to this new model,

4. Choose the model which has the lowest sum of absolute correlations relative to the selected models,

5. Repeat step 3 and 4 until the desired number of models have been retrieved or there are no models left which satisfy the significance threshold.

Selecting the ensemble components. The outlined methods to select the most diverse models from the candidate model set will generally NOT select from the models on the Pareto front. To some extent, we should expect this since the Pareto front models are, by definition, the optimal performing models and, as a result should be taking the most central path through the fuzzy response surface. Although diversity is important to provide a viable trust metric through the model consensus, not including the Pareto front models is not very emotionally satisfying since they are optimal and quite a bit of effort went into their development.

Since our ensemble will have two goals — consensus metric and prediction — we can make the argument that we should include models from the Pareto front. Towards that end, a strategy that seems to work well in practice is to build the operational ensemble from three components:

- Diverse models selected from the overall candidate set;
- Diverse models selected from the Pareto front of the candidate set;
- The "most typical" model from the Pareto front models.

Intuitively, this strategy should overload in the middle of the (fuzzy) response surface. In our ensemble evaluation scheme, the central models will dominate the prediction while the outer models will dominant the consensus metric. Figure 2.0 illustrates the distribution of the selected models for the inferential sensor.

Note that an ensemble is simply a container for the selected models each of which will be evaluated against parameter sets. Converting these disparate model predictions into an overall ensemble prediction and trust measure is the topic of the next section.

Ensemble evaluation

Ensemble evaluation consists of evaluating all of the constituent models embedded within the model and producing two numbers: a prediction and a consensus metric. The assumption is that the consensus is an indication of the trustworthiness of the prediction.

We generally target to have between 10 to 50 models in the final ensemble with the number of embedded models determining what is viable from a prediction and consensus evaluation standpoint. As a general rule for consensus,

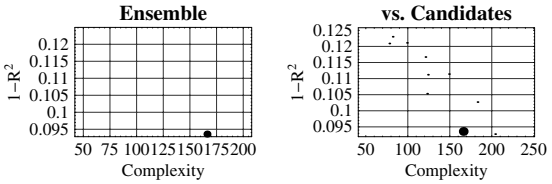


Figure 12-3. Here we look at the distribution of ensemble models on the fitness landscape relative to the overall candidate set. The correlation threshold for ensemble selection from the candidate model set was increased to 90% due to the relatively noisy data set.

we prefer robust statistical measures (mean, MAD, quantiles, etc.) as opposed to conventional statistical metrics (mean, standard deviation, etc.) since the conventional statistics are vulnerable to distortion from outliers. The conventional measures have the advantage that they are faster to evaluate since the constituent model outputs do not need to be sorted; however, that is generally not a significant issue from an operational viewpoint. Note that a common reference frame is important so conventional and robust metrics should not be generally mingled.

An attraction of the mean relative to the median as a ensemble prediction method is that the mean will be a smoother function since the median will undergo step changes as the dominant (median) model changes. A compromise which we use is to provide prediction smoothness as well as robustness is the "median average" — i.e., the average of the predictions from the 3–5 models surrounding the median.

We currently use the median average (averaging at least three models and more if a large ensemble is being used) with the spread (maximum - minimum prediction) used as the trust metric for small ensembles and either the spread, 10-90% quantile range or the standard deviation for large ensembles. However, the choice of ensemble prediction and consensus functions is an open research topic.

3. Ensemble Application

Our contention is that, "The consensus metric from a properly constructed ensemble will give an immediate warning that the ensemble prediction is suspect." This is a significant improvement for real-time systems over conventional methods since the delay until the prediction errors accumulate to a noticeable level is avoided along with the associated risks. Having a consensus metric is also useful for off-line analysis since such information can guide additional data collection as well as give human insight.

Off-line data analysis

One of the major benefits of symbolic regression — the human insight derived from examining the structure of the evolved expressions — is lost when many disparate models are gathered into an ensemble. That said, it may be that examining the selected models may be more insightful than examining the Pareto front models because of the implicit diversity of structure.

Of course, the response surface of an ensemble can be explored in a similar fashion as an individual model; however, the real benefit of an ensemble could be the exploration of the consensus surface since the divergence of the ensemble models indicates locations in parameter space where data is missing. This could be useful in applications such as combinatorial chemistry which use iterative model development. In essence, this would support an adaptive design-of-experiments approach.

On-line prediction

On-line data analysis is where the use of ensembles has its greatest benefits since the consensus metric acts as a trust metric to warn that the model predictions may not be accurate as well as an indicator that new or refined models may need to be developed. Whether the deviation in the prediction of ensemble models is due to operating in new regions in parameter space or fundamental changes in the targeted system must be determined by the user — however, he or she has been warned!

4. Example: An Inferential Sensor

Inferential sensors

Often in industry we want to monitor a variable which is hard to measure directly. Sometimes also called a "soft sensor", an inferential sensor uses measurements which can be reliably collected on-line to infer the state of the targeted response. One of the keys is to develop a function which will map from the easily observable parameters into the hard-to-observe response. The response may require special equipment or off-line lab testing which means that the training and test data is relatively precious. Of course, there are direct analogues of the inferential sensor to other domains such as financial modeling.

In this example, we have intentionally designed a test data set which spans a larger region of parameter space than the training set used to develop the symbolic regression models. This allows us to demonstrate the ability of a properly designed ensemble to identify when it is unsure of the validity its predictions.

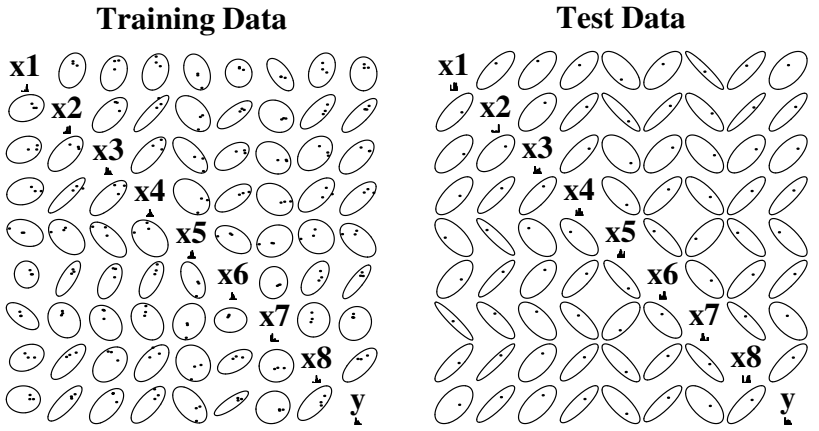


Figure 12-4. The test data set used spans a region of parameter space approximately ±20% larger than that of the training set used for model development. Note that all of the inputs are correlated with the output .

The data

The industrial data in this example is relatively small: 8 input variables and a response. The data was intentionally designed such that the test set (107 data records) covered a larger region of parameter space than the training set (251 records) used in the model development. As illustrated in Figure 4.0, all of the input variables are correlated with the output (lbs/day); this correlation is even stronger with the test data than with the training data.[2]

The models

Although only nine models were selected for the final ensemble in this case, we will not show them in the interests of brevity. Note that four inputs were dominant in the sense that they were present in every model. We should also note that only two models only contained the dominant variables. The other models featured between five and eight inputs.

Ensemble prediction performance

Figure 4.0 shows the model prediction performance (predicted vs. actual) for the training data set. The median model value is shown as is the extrema of

[2]The example modeling, analysis and figure generation were done using Evolved Analytics' DataModeler (*DataModeler*, 2007) add-on package for Mathematica.

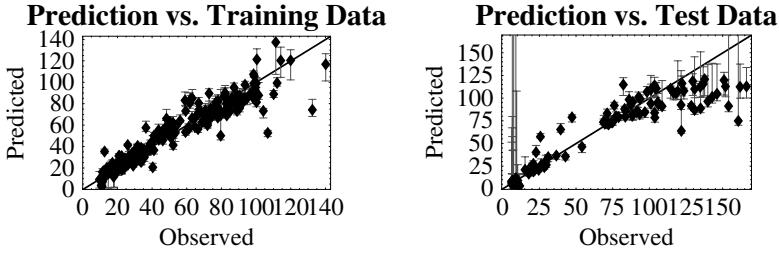


Figure 12-5. Here we show the ensemble performance against the training and test sets. The prediction here is the median model prediction and the consensus metric is the extrema points. The test set covers about 20% greater data range than used for the model development Notice that extrapolation is detected from the model divergences

the predictions. Note that the model does a relatively good job of prediction — with some problems for the few points at the high end of the range. However, these points do show a prediction divergence which is intuitively comforting.

Figure 4.0 also shows the model performance against the test data set — which was designed to test the ensemble prediction and detection of extrapolation abilities when encountering new regions of parameter space. There are three key points we must make about this graph:

- Extrapolation was clearly detected and flagged as shown by the consensus metric band.
- At the low end, the predictions were actually quite good; however, the extrapolation was detected.
- At the high end, the predictions are incorrect and are flagged as being untrustworthy. However, the ensemble predictions do show a graceful degradation since the predictions are also not wildly incorrect at any point.

The graceful degradation of the models is a result of choosing models from the appropriate region of fitness space (i.e., near the knee of the Pareto front) and further filtering the models by testing for robustness using interval arithmetic. From this foundation of robust and accurate models, we built the ensemble emphasizing diversity so that a valid trust metric would be generated.

The shape of the consensus surface

The response surface of the ensemble is the ensemble prediction as a function of the input parameters. Similarly, the consensus surface is the model disagreement behavior (defined by the consensus measure) as a function of the input parameters. In Figure 5.0 we look at the shape of the example inferential

sensor consensus surface under a ±20% extrapolation. Since detection of extrapolation in high-dimensional spaces is generally quite difficult, this is a very important attribute. We can search the consensus surface within the nominal operating range to identify points of model disagreement which would, therefore, be points where we might make extra effort to collect data to refine and improve the quality of the models.

5. Summary

The key message

The essence of this chapter is fairly simple:

- **All the data should be used** in the model development. To do otherwise is to intentionally produce myopic models.
- Models should be developed from a **multi-objective** viewpoint and chosen *post-facto* from the appropriate region of model fitness space.
- Models from this region should be tested for robustness via **interval arithmetic** to eliminate the risk of inappropriate pathologies.
- Ensembles of **diverse models** should be defined from this pool of accurate, simple and robust models.
- The **consensus of models** within a deployed ensemble should be monitored to detect operation in new regions of parameter space as well fundamental changes in the underlying system.

Following these recommendations does not obviate the need for monitoring the quality of the predictions that would be *de rigor* for a conventional machine learning model since it is possible that a system change could be undetected. However, the ability to deploy a model and have immediate assessment of the quality of model predictions is a huge improvement over conventional empirical modeling technologies.

Summary & Conclusions

In this chapter we discussed the development of ensembles of diverse robust models and their operational advantages due to the trust metric provided by the consensus of those models. The associated industrial example demonstrated that an ensemble could detect that it was encountering new regions of parameter space and flag that fact via a consensus metric. From a practical standpoint, this is very significant. The trepidation of deploying a conventional empirical model is based upon four potential problems:

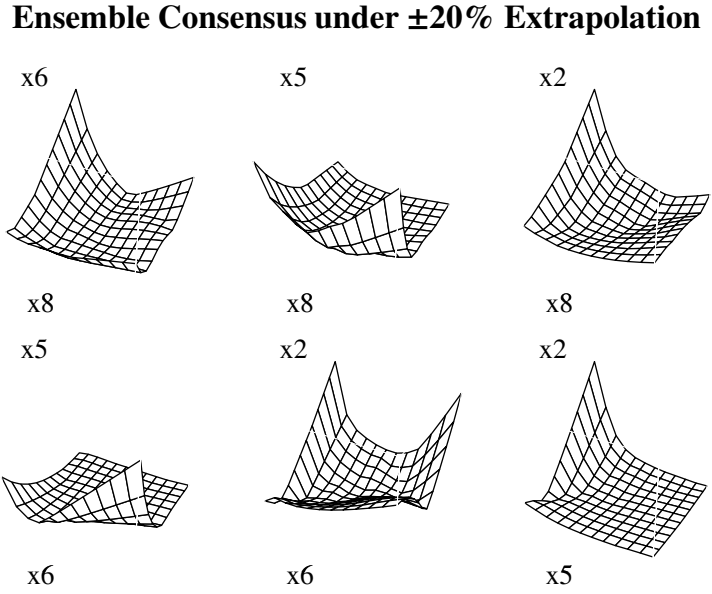


Figure 12-6. Here we look at the consensus surface plot of all pairwise combinations of the four variables which appear in every model within the ensemble (non-varying inputs are held constant at their mean values). As expected, the models diverge when extrapolating; this is an important capability since it is generally difficult to detect extrapolation in a high-dimensional parameter space.

- The model can not handle **data outside the training space** and, as a result, could have wildly erroneous predictions;
- The **underlying system could change** and the model would happily predict assuming that the training data (system) was still appropriate;
- The chosen model was **over-fitted to the training data** and, as a result, could have wildly erroneous results even if new data was within previous operating conditions and the underlying system had not changed;
- Problems in model quality could only be detected after-the-fact with a further **detection delay** imposed to collect sufficient new data to declare the deployed model to be invalid.

The traditional way to mitigate the overfitting risk is to partition the data into multiple subsets, train against one of the subsets and select based upon the model performance against the multiple subsets. To a large extent using the Pareto front in symbolic regression and selecting models from the knee of the Pareto front naturally guards against either over-fitting or under-fitting. Using interval arithmetic to eliminate models with potential pathologies further mitigates the empirical modeling risk.

Conventionally, detecting that the model was no longer valid either due to the underlying system changing or due to input data outside the training range essentially involved diligent oversight on the part of the user to track the model output against reality and to make a judgement call as to whether errors were normal or if there was a structural problem with the model-reality match. The problem in practice is that the human oversight required is either not done or not done in a timely fashion; in any event, the problem with model prediction quality could only be discovered after the fact which implies an unavoidable time delay in response. A trusted incorrect model can be costly or dangerous. Using ensembles of diverse models mitigates this risk since immediate assessment of prediction quality is provided via the trust metric derived from the diversity of predictions from the models embedded in the ensemble.

Diverse model ensembles enable some very profound practical advantages for real-world model deployment:

- We can now **use ALL available data in the model development**. If data is precious, we are essentially removing the fogged glasses we placed on the modeling process to avoid the risk of over-fitting. Traditionalists are going to be VERY disturbed at this; however, there is no real need for the test/training/validation/etc. data partitioning!
- We can **let the ensemble warn us when the model output should not be trusted**. The ensemble can detect that the ensemble is extrapolating into

new regions of parameter space and warn us immediately. Timely awareness allows the human to make appropriate judgements as to whether processing parameters need to be adjusted or a new model is required to reflect fundamental changes and avoid costly surprises.

Although we believe that the use of symbolic regression ensembles represents a profound opportunity in real-world model deployment, it should not be construed that we advocate the abdication of human responsibility for ensuring that the models are accurate and applicable. Any empirical model should be monitored and calibrated, as appropriate. Some symbolic regression models that we have developed are still in active use in the chemical process industry close to a decade after the original development so it is possible to develop robust and effective models.

Issues and future efforts

The god-father of ensembles for symbolic regression are the stacked analytic networks (Kordon et al., 2003), which have been deployed in industrial application for over fifteen years as of this writing. Lessons learned from that technology is that ensemble models should be of similar complexity but diverse. Unfortunately, those two characteristics are harder to define in a symbolic regression context. Some of the open issues we are currently addressing are:

- The nature of empirical modeling is that model predictions will be highly correlated – otherwise, they would not be quality models. Hence, we need to define an appropriate correlation threshold to declare model independence. This threshold will change depending upon the system complexity, number of data points and noise levels.

- Consensus (or, rather, lack of consensus) is the key trust metric of an ensemble. We currently use the spread or a summary statistic (e.g., standard deviation) depending upon the number of models in the ensemble. Is there something better?

References

Castillo, Flor, Kordon, Arthur, Sweeney, Jeff, and Zirk, Wayne (2004). Using genetic programming in industrial statistical model building. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 3, pages 31–48. Springer, Ann Arbor.

Hamill, Thomas (2002). An overview of ensemble forecasting and data assimilation. In *Preprints of the 14th conference on Numerical Weather Prediction*, Ft.Lauderdale, USA. American Meteorological Society.

Keijzer, Maarten (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, Conor, Soule, Terence, Keijzer, Maarten, Tsang, Edward, Poli, Riccardo, and Costa, Ernesto, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex. Springer-Verlag.

Kordon, Arthur, Smits, Guido, Kalos, Alex, and Jordaan, Elsa (2003). Robust soft sensor development using genetic programming. In Leardi, R., editor, *Nature-Inspired Methods in Chemometrics: Genetic Algorithms and Artificial Neural Networks*. Elsevier, Amsterdam.

Kordon, Arthur, Smits, Guido, and Kotanchek, Mark (2006). Industrial evolutionary computing. In *GECCO 2006: Tutorials of the 8th annual conference on Genetic and evolutionary computation*, Seattle, Washington, USA. ACM Press.

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

Kotanchek, Mark, Smits, Guido, and Vladislavleva, Ekaterina (2006). Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 3, pages –. Springer, Ann Arbor.

Smits, Guido and Vladislavleva, Ekaterina (2006). Ordinal pareto genetic programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver. IEEE Press.

DataModeler (2007). Add-on analysis package for *Mathematica*.

Vladislavleva, Ekaterina and Smits, Guido (2007). Order of non-linearity as a complexity measure for models generated by symbolic regression via genetic programming. In *review at IEEE Trans. on Evolutionary Computation (sumbitted)*.

Wichard, Joerg (2006). Model selection in an ensemble framework. In *Proceedings of the IEEE World Congress on Computational Intelligence WCCI 2006, Vancouver, Canada.*

Chapter 13

IMPROVING PERFORMANCE AND COOPERATION IN MULTI-AGENT SYSTEMS

Terence Soule[1], Robert B. Heckendorn[1]
[1]*Computer Science Department, University of Idaho, Moscow, ID, 83844-1010, USA.*

Abstract Research has shown that evolutionary algorithms are a promising approach for training agents in heterogeneous multi-agent systems. However, research in evolving teams (or ensembles) has proven that common evolutionary approaches have subtle, but significant, weaknesses when it comes to balancing member performance and member cooperation. In addition, there are potentially significant scaling problems in applying evolutionary techniques to very large multi-agent systems. It is impractical to train each member of a large system individually, but purely homogeneous teams are inadequate. Previously we proposed Orthogonal Evolution of Teams (OET) as a novel approach to evolving teams that overcomes the weaknesses with balancing member performance and member cooperation. In this paper we test two basic evolutionary techniques and OET on the problem of evolving multi-agent systems, specifically a landscape exploration problem with heterogeneous agents, and examine the ability of the algorithms to evolve teams that are scalable in the number of team members. Our results confirm that the more traditional evolutionary approaches suffer the same weakness with multi-agent systems as they do with teams and that OET does compensate for these weaknesses. In addition, the three algorithms show distinctly different scaling behavior, with OET scaling significantly better than the two more traditional approaches.

Keywords: Teams, ensembles, multi-agent systems, cooperation, Orthogonal Evolution of Teams, cloning, OET, scalable

1. Introduction

Many practical problems are too large, too complex, or are structured inappropriately for single, monolithic intelligent agents to solve successfully or efficiently. For example, large, complex classification problems may contain specialized sub-domains within the larger problem space that a single agent is likely to overlook. Or a problem may be structured such that a single agent cannot be reasonably expected to have the specific resources to solve each subproblem encountered, such as in an exploration problem involving a very large space with multiple terrains. Finally, a problem space may simply be too large to be efficiently processed by a single agent and a cooperative approach will yield better results in limited time. For problems with these constraints, cooperative multi-agent systems are an obvious, and often a necessary, approach.

A significant concern with multi-agent systems is their scalability. As the number of agents increases the space of possible behaviors grows extremely rapidly, especially in systems that depend on heterogeneous agents with high levels of cooperation. High levels of cooperation and performance in a system of heterogeneous agents require that each agent have a unique control program that must be programmed or trained individually. Thus the space grows rapidly because each additional agent leads to multiple additional opportunities for (cooperative) interactions.

In this paper we approach the problem of scalability by initially training a small heterogeneous team of agents; the agents are then copied (cloned) N times each to create a larger, semi-heterogeneous team. This is a form of *hybrid team learning* as defined by Panait and Luke (Panait and Luke, 2005). Since larger teams require more computing resources to evaluate and train, using a relatively small team significantly reduces training time. However, because we are dealing with deterministic programs there is a significant risk that the N clones will "overlap" noncooperatively and not significantly contribute to the large system's fitness. (A *random* operator is available, but may or may not be used–see the vector model below.) To overcome this problem and retain training efficiency we experiment with **limited cloning** and test several evolutionary algorithms. It is hoped that limited cloning during training will improve the evolution of cooperation in larger teams.

The evolutionary algorithms we test are classic team and island algorithms, in addition to our own approach Orthogonal Evolution of Teams (OET). Each of these algorithms applies varying amounts of selective pressure on team members, and on teams as a whole. Our goal is to determine how the balance between pressure for individual performance versus team performance affects the behavior of the larger scaled systems.

Our results show that OET produces the most efficient and effective teams. Furthermore, we show that teams generated by OET perform better when cloned

to form larger teams, i.e. they are the most scalable with respect to team size. This makes it possible to evolve relatively small teams, which requires significantly less training time, and then clone the evolved members to create much larger teams–without a significant reduction in performance. In addition, we find that training with limited cloning, in which agents are copied once before being evaluated, significantly improves the final teams' scalability.

2. Background

On many interesting problems, a multi-agent system must consist of agents that individually perform well and that cooperate well as a team. Typically this means that the members *specialize* on distinct, but potentially overlapping, sub-domains of the problem space. This requires heterogeneous control structures, which makes programming the agents difficult. Furthermore there are often multiple ways to divide a problem into unique sub-domains and it is unclear *a priori* what choice of sub-domains will lead to an optimal solution. For these two reasons it can be advantageous to have all of the agents learning together, so they can learn to specialize effectively; what Panait and Luke refer to as *heterogeneous team learning* (Panait and Luke, 2005).

Research has shown the evolutionary techniques, including both genetic algorithms (GAs) and genetic programming (GP), are extremely effective at evolving multi-agent teams. Evolutionary approaches for training multi-agent teams have been applied to a wide range of knowledge representations, including teams of: neural networks (Liu et al., 2000), oblique decision trees (Cantu-Paz and Kamath, 2003), stack-based predictors (Platel et al., 2005), and teams of induced functions (Soule, 1999). Evolutionary approaches have also been applied to a wide range of problem domains including robot navigation (Iba, 1997), team sporting strategies (Raik and Durnota, 1994), predator strategies (Haynes et al., 1995; Luke and Spector, 1996), hazard assessment (Obitz et al., 1999), and cancer and diabetes diagnosis (Cantu-Paz and Kamath, 2003; Liu et al., 2000).

In previous research we have shown that island approaches–approaches that evolve team members independently–produce highly fit team members, but there is a high probability that those members have correlated errors resulting in less than optimal team performance. We have also shown that team approaches, approaches in which members are evaluated and selected as a team, can produce team members with inversely correlated errors, leading to relatively good team performance, but the members themselves are relatively poor, which limits the team's performance (Soule, 2003; Soule and Komireddy, 2006; Thomason and Soule, 2007).

To overcome the weaknesses of the island and team evolutionary approaches, we have developed a novel class of genetic programming algorithms that al-

ternates between treating the evolving population as consisting of independent islands of agents, and treating it as single population of teams. These orthogonal views of the population lead to the name Orthogonal Evolution of Teams (OET) (Soule and Komireddy, 2006). In previous work, we showed that with both classification problems and a multi-agent exploration problem, OET produces teams whose members perform better than those generated with team approaches, and which cooperate better than those generated using island approaches (Soule and Komireddy, 2006; Thomason and Soule, 2007; Soule and Heckendorn, 2007).

However, for all three of these approaches as team size increases the training time increases significantly, both because more agents must be evaluated and because the search space is significantly more complex due to the increasing opportunities for interactions. This is a particularly serious problem in multi-agent systems that involve tens, hundreds, or even thousands of agents. In this paper we use a simple technique to overcome the problem of evolving large teams; a small team is evolved and its members are copied to create a much larger semi-heterogeneous team. This expansion-by-copying approach has potentially significant weaknesses. The agent copies may behave identically, not contributing to the solution, but using resources. The copies may even interfere with each other, lower the fitness of the system. In this paper we compare the performance of the three general approaches: island, team, and OET to determine which approach produces teams that can be effectively expanded, by making copies of the initial agents.

3. The Problem Environment

The environment is divided into a two dimensional grid (40*40). At the beginning of each evaluation each grid square has a twenty percent chance of being labeled as "interesting". The interesting squares are determined randomly for each evaluation so that agents cannot memorize where the interesting squares are. Instead the agents must learn general search algorithms.

There are two agent types: **scouts** and **investigators**. A scout's role is to find interesting squares and mark them with a beacon that is detectable at a distance by the investigators. An investigator's role is to investigate interesting squares and mark them as **investigated**. Scouts travel at up to twice the speed of investigators. If a scout is in an interesting square or is next to an interesting square, it automatically places a beacon in the interesting square (unless there is already a beacon there). If an investigator enters an interesting square, regardless of whether the square is marked with a beacon, it changes the square to investigated and deactivates any beacons in the square. It is important to reinforce here that the investigators work at half the speed of the scouts, but can see the beacons at a distance. Therefore the space can be more efficiently

explored by fast scouts marking interesting areas with beacons and investigators using the beacons to go directly to the areas to be investigated. Furthermore, the two groups of agents have different sub-goals and must divide up the space to be searched efficiently since the task has a time limit.

Agents can leave, and later return, to the area to be explored. However, they are penalized (see the fitness function below) for moves that end outside of the exploration area.

This model represents an abstraction of a number of practical problems. For example, scouts and investigators could represent two robot types exploring a minefield. Scouts fly overhead marking locations of potential mines and investigators deactivate the mines. Alternatively, they could represent an automated planetary surveying team. Scouts identify potentially interesting geological formations and investigators follow up by taking soil samples, etc.

We seek to answer several fundamental questions about heterogeneous team training and scalability.

1. When using cloning to scale up team sizes, what level of selective pressure on individuals versus teams optimizes the scaled teams' performance?
2. Does using limited cloning during evolution significantly improve the performance of the scaled up teams?
3. When limited cloning is used during evolution, what level of selective pressure on individuals versus teams optimizes the scaled teams' performance?

A vector model for agent movement

The agents' environment is a two-dimensional real-valued space. Agent movement is determined by a vector expression, represented by an expression tree, that calculates the next move. This expression calculates and returns a vector, based on current input vectors, and the agent moves in that direction. Investigators are limited to moves of length one and scouts are limited to moves of length two. It is this expression tree that is the representation of the solution space in the form of a program for each agent to determine its behavior. The objective of the evolutionary algorithms is to evolve a good expression tree for each team position.

Input vectors (terminal nodes in an agent's evaluation tree):

- North: a unit vector pointing North, 0 radians.
- Constant: a vector that is initially generated randomly at the time an agent's program is initialized. The constant remains so throughout the lifetime of agent, but can be overwritten during mutation for instance.
- Random: a vector that is randomized at each time step.

- Nearest scout: a vector to the nearest scout.
- Nearest investigator: a vector to the nearest investigator.
- Nearest beacon: a vector to the nearest beacon.
- Last move: a vector representing the agent's last move.
- Nearest edge: a vector to the nearest boundary of the search space.

In the current implementation there is no limit on an agent's vision; e.g., an agent can "see" the nearest beacon regardless of its distance. If an input is meaningless, e.g. nearest beacon when no beacons are present, the zero vector (direction = 0, magnitude = 0) is returned.

Vector operations (non-terminal nodes in an agent's evaluation tree):

- Invert: takes a single vector argument and inverts it (rotates π radians).
- Sum: takes two vector arguments, returns the vector sum.
- IfLTEmagnitude: takes four vector arguments, if the magnitude of the first argument is less than the magnitude of the second argument it returns the third argument, otherwise it returns the fourth argument.
- IfLTEdirection: takes four vector arguments, if the angle of the first argument is less than the angle of the second argument it returns the third argument, otherwise it returns the fourth argument.

Mutation is restricted to leaf nodes.

Clearly the choice of inputs and operations has a significant influence on how the agents can evolve. We chose a fairly extensive set of input vectors. These vectors represent some distance sensors but the agents do not share information. Future research will look at the effect of changing this set: reducing the range of agent's vision, allowing agents to sense each other by "name", etc.

4. Fitness Evaluation

A significant difficulty in many multi-agent systems is how to assign credit (fitness) to individual team members. We chose to begin with a simple, basically greedy, approach. Each evaluation of fitness involves a new random problem space of interesting regions being setup and the bots placed dead stop in the center pointing "North". Then they are given a fixed amount of time.

The fitnesses for the scouts and investigators are as follows:

$$fitness_s = 3\beta - .1b \qquad\qquad fitness_i = 3I - .1b$$

Where β is the number of beacons placed, b is the number of time steps outside of the bounded problem area, I is the number of interesting areas investigated.

Table 13-1. Summary of the evolutionary algorithm parameters.

	Standard	**Restricted Resources**
Population Size	100	100
Crossover Probability	1.0	1.0
Mutation Probability	$\frac{2}{tree}$ size	$\frac{2}{treesize}$
Selection	3 member tournament	3 member tournament
Run Time	1500 evaluations	1500 evaluations
Maximum Size	None	None
Initial Population	Full trees of depth 4	Full trees of depth 4
Number of trials	30	30
Grid Size	40*40	40*40
Number of time steps	534	200

Thus, scouts and investigators are rewarded for finding interesting squares and investigating them, respectively, and penalized for leaving the boundaries. Interestingly we have observed that if the boundary penalty is too large agents will evolve that simply sit still to avoid incurring a penalty.

The fitness of a team is the sum of the team members' fitnesses. The fitness of a particular member or team will vary somewhat between evaluations because for each evaluation the environment is randomly generated. An individual's fitness is recomputed (possibly with different results due to the randomness of the environment) whenever its team's fitness needs to be recomputed.

5. Evolutionary Algorithms

A steady-state evolutionary algorithm is used. In these algorithms, selection, crossover, mutation, and insertion occur on an individual basis rather than by performing each step on the whole population at once yielding a new population. The basic algorithm parameters are given in Table 13-1. **Number of trials** is the number of times the algorithm is run for statistical purposes. **Number of time steps** is the virtual time allowed the team of agents to perform on a given problem. **Run time** is the number of teams evaluated for their fitness during an evolutionary algorithm's run.

During the mutation step, only leaf nodes (terminals) are mutated. Non-terminals can change through crossover. Crossover involves choosing, at random, a single subtree in each of the parent evaluation trees and exchanging them. Terminal and non-terminal nodes were chosen with equal probability.

To vary the amount of selective pressure for team versus individual performance, we tested three evolutionary algorithms: Team, OET, and Island. Each

algorithm varies in how teams and team members are selected. We then compare the fitness of the teams and the team members under the three algorithms.

Team Algorithm

In the team algorithms all evolutionary pressure is applied to teams as a whole. In each iteration, tournament selection (tournament size of 3) is used to select two "parent" teams. The teams undergo crossover with each of the team members in the first parent being crossed with the equivalent team member in the second parent (e.g. member 1 of parent team 1 is crossed with member 1 of parent team 2; member 2 of parent team 1 is crossed with member 2 of parent team 2; etc.). This produces two new "offspring" teams. Each member of the new offspring teams is subjected to a mutation operation. During mutation each terminal node has a 1 in 10 chance of being randomly mutated into a new leaf node. Internal nodes are not affected by mutation. Two reverse tournaments, tournaments in which the individual with the lowest fitness "wins', of size 3 are used to select two poorer teams. These teams are replaced by the offspring teams.

In the team algorithm all of the selective pressure applies to the teams. Parents are selected based on the performance of the team as a whole. Teams are selected for replacement based on the performance of the team as a whole compared to members of a tournament of teams.

Orthogonal Evolution of Teams (OET)

In the OET algorithm pressure is applied to both members and teams. In each iteration tournament selection (tournament size of 3) is used to select team members to generate, in piecewise fashion, two parent teams. That is, tournament selection is applied to the first members of the teams to pick a "winning" team member 1, then tournament selection is applied to the second members of the teams to pick an above average team member 2, and so forth, until a team consisting of above average members is created. Thus, if teams consist of three scouts and three investigators, the process is repeated six times to create a new "all star' team.

This process is repeated to create two "parent'' teams. These two parent teams undergo crossover and mutation as in the team algorithm. And also as in the team algorithm, two below average teams are replaced.

Thus, team members must individually be at or above average in a tournament to be selected for the parent teams. Teams must also be above average in a team tournament or they may be replaced by the offspring teams. Thus there is direct pressure on both the team members and the teams as a whole.

Island Algorithm

In the Island algorithm, pressure is applied to members only. In each iteration, for each team member, four tournaments of size 3 are performed. Tournament selection is used to select two winning members in the tournaments and two losing members. The losing members are replaced by the winning members. The two replacement members then undergo crossover and mutation. The two teams containing the newly crossed and mutated members are then evaluated. In this algorithm the fitness of a team as a whole is never considered during the evolutionary process for the individual. Thus, evolution will emphasize the individuals' performance.

6. Experiments

To test which algorithm performs best when the number of individuals must be scaled up to handle we ran the three algorithms (island, team, and OET) in two scaling experiments:

1. Control: Teams consisting of three scouts and three investigators are evolved. After evolution the best team is cloned, each agent is copied four times to create a team with 24 members, and that team is evaluated. This tests the base scalability without changing the learning algorithm to account for a larger team.

2. Limited cloning: Teams consisting of three scouts and three investigators are evolved with limited cloning. When a team is evaluated each agent is copied twice, creating a team of size six, and that team is evaluated. (The fitness of an agent is the sum of the fitness of its two clones.) After evolution the best team is cloned, each agent is copied four times to create a team with 24 members and that team is evaluated. This tests if by evolving the teams to deal with just one other clone is enough to learn to deal with many more clones. This would be far more efficient than evolving a full sized team.

The number of agent evaluations is kept constant for each experiment. For example, with limited cloning evolution occurs for half as many iterations as in the control case, because with limited cloning the teams include twice as many agents (12 versus 6).

7. Discussion

To review, there were two sets of experiments. The first was to train three scouts and three investigators to solve a cooperative problem. Then clone each of the six individuals in the best team into a set of four identical agents and run the set of 24 to see if they had evolved general solutions to cooperation even

Table 13-2. Mean, median, and standard deviation of performance of the best teams for different algorithms as evaluated by the larger 4 clone team.

Nonclonal			Limited Cloning		
Island	OET	Team	Island	OET	Team
5559.73	5549.41	3050.78	4942.72	6370.85	4648.57
5649.75	5937.5	3167.19	4944.79	6592.34	4708.15
951.041	1672.17	1603.65	904.609	867.911	900.987

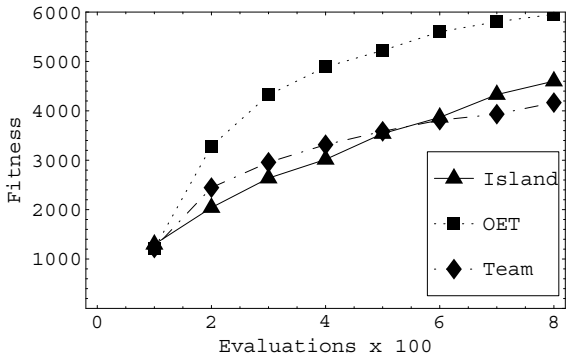


Figure 13-1. Aveage of best team performance using limited cloning averaged over 30 trials.

when cloned. This is referred to as the **nonclonal** experiments. The second experiment was to train using two identical copies of each of the six individuals for a total of 12 individuals. It was hoped that this will evolve better cooperation under cloning. To test this we again cloned each of the six different individuals in the best team into a set of four identical agents and ran the set of 24 to see if they had learned to better cooperate under the cloning condition. This is referred to at the **limit cloning** experiments.

Three types of evolutionary training algorithms were tried to see how they performed under this cloning/noncloning regime. These were described above as island, team, and OET. In this section we will see how the various algorithms and experiments compared.

The results clearly show that in all but one case the type of evolutionary algorithm has a significant effect on the evolution of the teams. This is seen in P values for the two sided Student's t-test for average fitness of the population at the end of run, averaged over 30 runs and P values for fitness of the best team at the end of run, averaged over 30 runs (Table 13-3). These figures indicate that with a $P < .01$ all samples were statistically different, many with near certainty. One exception to this was in the limited cloning experiment

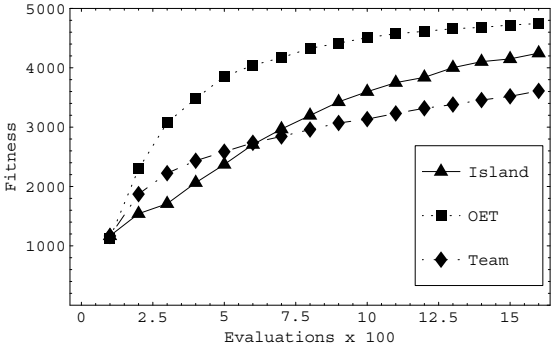


Figure 13-2. Average of best team performance of nonclonal evolution averaged over 30 trials.

Table 13-3. A table comparing the performance of nonclonal learning algorithms for **average team fitness** and **best team fitness** in population by P values for two-sided Student's T Test of similarity of distribution.

		Team vs. OET	Team vs Island	Island vs OET
Nonclonal	Average Team	8.84×10^{-10}	0.0000528	.000701
	Best Team	6.28×10^{-14}	1.72×10^{-7}	2.44×10^{-6}
Limited Cloning	Average Team	1.43×10^{-17}	0.836	1.94×10^{-18}
	Best Team	2.39×10^{-21}	0.000592	1.62×10^{-17}

where the island and team algorithms did not produce significantly different populations, however the best teams from the 30 sample runs were significantly different at much better than the $P < .01$ level. These numbers suggest that any observations comparing pairs of algorithms will be statistically significant at at least the $P < .01$ level unless the island/team population is considered for the limited cloning experiment.

The performance of the algorithms in the nonclonal experiment is graphed in Figure 13-2. While the team approaches gradually outperforms the island model the OET far outperforms either from early on in the evolutionary process, suggesting that OET is quickly acquiring the advantages of both team and individual learning. The limited cloning experiment is even more dramatic. Again early superiority in the evolution is gained, but the difference between team and individual is not as great and is clearly seen in the P values of Table 13-3. Any number of factors may explain why there is the lack of difference between team and island approaches for limited cloning. For instance, it may be that the time scales of the two graphs in Figures 13-2 and 13-1 are not directly comparable. It is clear, however, that the OET approach is superior.

Table 13-4 presents the statistics for the average and best teams and the members of the best teams at the end of the evolutionary runs. Reading down in a given cell the values are the mean, median, and standard deviation. The data are from 30 independent trials.

For the nonclonal runs the OET algorithm outperformed the island algorithm, which outperformed the team algorithm, for both the average fitness of the population and best teams. Interestingly, in the limited cloning runs, the performance of the OET and team algorithms improved, while the performance of the average team generated with the island algorithm declined. This strongly suggests that the OET and team algorithms can effectively learn to use the extra copies of agents produced by limited cloning, while the island algorithm finds it more difficult to take advantage of the clones.

The second two sections of Table 13-4 show the average performance of the best, worst, average agents and the difference between the best and the worst agents in the best teams. The island algorithm produces team members with the most similar fitnesses. This can be seen by comparing the average best and worst in teams in Table 13-4 and by looking at the value of the width (best minus worst) in that table. Presumably this occurs because in the island algorithm all of the selective pressure is applied to individual members–any individual with significantly below average fitness is removed from the population and the fitnesses of the agents remain similar. The results are similar in the limited cloning experiments.

In contrast, the team algorithm produces team members with the most variation in fitnesses. Most of this variation occurs because the worst members created via the team algorithm are significantly worse than the worst members created by the other algorithms. This is a common problem with pure team approaches – they allow 'hitchhikers' who contribute little to the team's fitness. Despite the presence of the very poor members, the teams perform reasonably well suggesting that the members are indeed cooperating. Although we have yet to verify it, this suggests that even the poor performers could be getting extra points for the team by picking up high cost hard to achieve sub-objectives, e.g. finding and investigating a few interesting squares in the far corners of the space.

Interestingly, the improvement in the team algorithm's performance with the limited cloning approach is a result of an improvement in the best and average team members, not the worst members (see the minimum agent performance entries in Table 13-4. That is, with limited cloning the team algorithm seems to train a few members that are very good and that perform well when cloned, but it also maintains a few agents that are very poor, even when cloned.

The OET algorithm appears to balance these two approaches, blending pressure to get all members performing well and pressure to get the team working together as a whole. The members of the best teams created via the OET al-

gorithm have more variance than the members created via the island approach, and less variance than the members created via the team approach. Overall, the average team in the population produced via the OET algorithm are significantly better than the average teams produced by the island algorithm and by the team algorithm (Table 13-4). The best teams produced via the OET algorithm are significantly better than the best teams produced by the island algorithm or by the team algorithm.

This performance advantage of OET is even more significant with limited cloning. In OET and team algorithms, the best agents in a team improve with limited cloning, which means they are effective when cloned. And as with the island approaches the worst members perform reasonably well, as do their clones.

In the second experiment we investigate the scalability of the evolved teams. We cloned the best teams into a team of 24 individuals with 4 clones of each different agent and reevaluated. Table 13-2 shows the mean, median, and standard deviation of performance of the teams over 30 trials in this new larger team. The performance of both OET and team improved significantly by using limited cloning (see Figure 13-3). This suggests that team-based selective pressure on teams with limited cloning better prepares the agents for working with multiple clones. Surprisingly the performance of the island model decreased with limited cloning suggesting that agents evolved with independent populations cannot learn how to work with clones from being trained on smaller sets of example clones.

To verify that the performance of the clones is due to performance of the "parent" agent we measured the correlation between performance of the agents and the average performance of their clones (see Table 13-5). In all cases the clones' performance was correlated with the performance of their parents. The highest correlation is for nonclonal island and limited cloning OET. We hypothesize that the nonclonal island is highly correlated because it has evolved to create individuals which work independently and so their performance is little affected by the presence of other agents. We hypothesize that the limited cloning OET teams are highly correlated because they have successfully learned in an environment with a few clones. We have yet to verify this result.

8. Conclusions

A significant hurdle to creating large multi-agent systems, systems consisting of hundreds (or in the case of nanobots, thousands or even millions of agents) is the ability to efficiently generate heterogeneous cooperative behaviors. Clearly training, or programming, each agent individually is not practical, even with automated techniques like GP. A seemingly reasonable approach is to take an efficient multi-agent system and "clone" its members generating teams as large

Table 13-4. The first two sections of this table are the team statistics for the various algorithms and experiments taken at the termination of evolutionary training. The remaining parts of the table are the fitness of individuals. For each algorithm we measured best in team, worst in team, average of all individuals in the teams, and the width, which is maximum minus minimum. In either case, each cell has mean, median, and standard deviation over 30 evolutionary runs.

	Nonclonal			Limited Cloning		
	Island	OET	Team	Island	OET	Team
Avg Team	3123.7	**3448.4**	2618.6	2992.7	**4592.5**	2970.9
	3099.8	3485.5	2618.8	2976.9	4712.2	2972.3
	374.0	326.1	506.1	282.5	532.4	499.8
Best Team	4309.0	**4758.4**	3617.6	4733.9	**6084.5**	4275.8
	4325.8	4748.2	3649.2	4830.9	6144.2	4314.7
	363.8	295.6	512.5	460.3	392.7	514.0
Max Scout	983.2	**1102.3**	1062.0	1106.1	**1366.8**	1358.4
	974.4	1100.4	1067.6	1004.5	1368.6	1272.5
	141.6	126.9	176.9	297.7	204.3	365.0
Min Scout	480.4	**504.5**	199.1	493.1	**636.1**	198.8
	485.2	534.0	180.3	469.1	607.9	164.6
	122.9	173.8	148.6	130.4	205.0	194.7
Avg Scout	729.2	**810.9**	619.3	771.8	**1001.5**	723.7
	746.1	800.8	619.2	786.4	1006.0	727.5
	69.0	63.8	91.1	93.8	67.2	113.3
Width Scout	502.8	597.8	**862.9**	613.0	730.7	**1159.6**
	509.6	590.9	879.1	496.9	816.8	1098.2
	207.7	272.6	275.7	357.7	360.5	501.5
Max Investigator	945.6	**1027.9**	1003.3	1186.5	**1388.0**	1562.8
	949.9	1052.9	1019.8	1182.6	1396.4	1484.2
	114.1	113.6	113.6	198.1	216.9	426.1
Min Investigator	447.6	**479.9**	155.3	412.8	**625.0**	22.9
	467.8	518.2	66.7	415.7	649.2	0.8
	116.6	175.4	210.0	134.0	257.8	109.9
Avg Investigator	707.1	**775.3**	586.6	806.1	**1026.6**	701.6
	709.6	774.6	600.7	803.1	1033.1	684.1
	59.9	41.5	88.7	71.8	68.8	85.6
Width Investigator	497.9	548.0	**848.0**	773.7	763.0	**1539.9**
	472.0	524.2	960.9	746.7	737.5	1455.9
	180.8	254.7	278.0	290.3	413.3	497.0

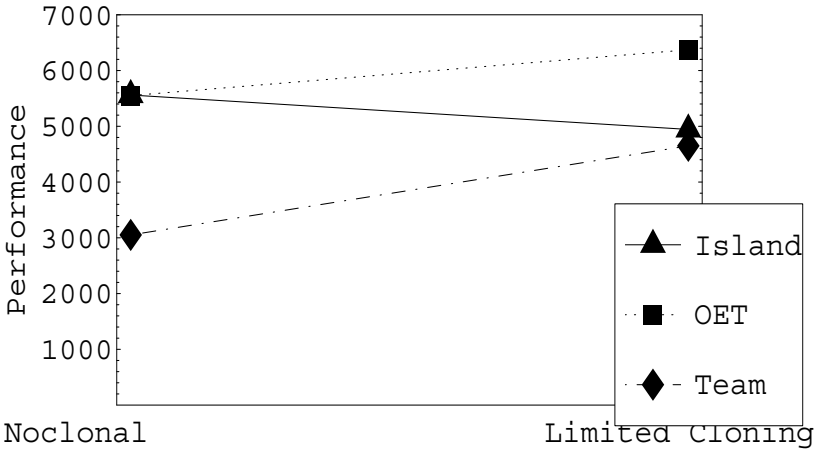


Figure 13-3. Changes in performance due to limited cloning for the three algorithms.

Table 13-5. Pearson's correlation coefficient between the performance of the each original trained team member and the average performance of its clones in the context of the larger problem. These numbers compare with for example a value of .019

	Island	Team	OET
Nonclonal (permutation)	.839 (.093)	.248 (-.0215)	.750 (-.106)
Limited Cloning (permutation)	.667 (.102)	.662 (-.0174)	.930 (.0524)

as desired. However, it is not guaranteed that the cloned members will continue to cooperate well, indeed there is good reason to fear that rather than cooperating the clones will interfere with each other.

We have shown that for problems requiring cooperation within such semi-heterogeneous teams (teams with multiple copies of heterogeneous agents) our evolutionary algorithm (OET), which encourages both competition between individuals and teams, significantly out performs either approach alone. Equally significantly we have shown that limited cloning, in which teams are evaluated with a few clones, produced teams with agents much more adapted to working with their clones and that again OET was significantly better than the other algorithms tested.

This means that by using OET and limited cloning it is possible to evolve agents that perform well individual, cooperate well, and remain effective when copied to create larger teams. This makes it possible to envision extremely large cooperative teams without the need to evolve each member individually or to restrict the teams to purely homogeneous members.

Acknowledgment

This publication was made possible by NIH Grant #P20 RR16448 from the COBRE Program of the National Center for Research Resources and by NSF grant #0535130.

References

Cantu-Paz, Erick and Kamath, Chandrika (2003). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68.

Haynes, Thomas, Sen, Sandip, Schoenefeld, Dale, and Wainwright, Roger (1995). Evolving a team. In Siegel, Eric V. and Koza, John, editors, *Working Notes of the AAAI-95 Fall Symposium on GP*, pages 23–30. AAAI Press.

Iba, Hitoshi (1997). Multiple-agent learning for a robot navigation task by genetic programming. In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick R., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 195–200. San Francisco, CA: Morgan Kaufmann.

Liu, Yong, Yao, Xin, and Higuchi, Tetsuya (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387.

Luke, Sean and Spector, Lee (1996). Evolving teamwork and coordination with genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick R., editors, *Genetic Programming 1996: Proceedings of the*

First Annual Conference on Genetic Programming, pages 150–156. Cambridge, MA: MIT Press.

Obitz, D. W., Basak, S. C., and Gute, B. D. (1999). Hazard assessment modeling: An evolutionary ensemble approach. In *Proceedings of the Genetic and Evolutionary Computation Conference: GECCO-1999*, pages 1543–1650. Morgan Kaufmann.

Panait, Liviu and Luke, Sean (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Platel, Michael Defoin, Chami, Malik, Clergue, Manuel, and Collard, Philippe (2005). Teams of genetic predictors for inverse problem solving. In *Proceeding of the 8th European Conference on Genetic Programming - EuroGP 2005*.

Raik, Simon and Durnota, Bohdan (1994). The evolution of sporting strategies. In Stonier, Russel J. and Yu, Xing Huo, editors, *Complex Systems: Mechanisms of Adaption*, pages 85–92. IOS Press.

Soule, Terence (1999). Voting teams: A cooperative approach to non-typical problems. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 916–922, Orlando, Florida, USA. Morgan Kaufmann.

Soule, Terence (2003). Cooperative evolution on the intertwined spirals problem. In *Genetic Programming: Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, pages 434–442. Springer-Verlag.

Soule, Terence and Heckendorn, Robert B. (2007). Evolutionary optimization of cooperative heterogeneous teams. In *Evolutionary and Bio-inspired Computation: Theory and Applications, Proceedings of SPIE*, volume 6563. International Society for Optical Engineering.

Soule, Terence and Komireddy, Pavankumarreddy (2006). *Orthogonal Evolution of Teams: A Class of Algorithms for Evolving Teams with Inversely Correlated Errors*. Springer.

Thomason, Russell and Soule, Terence (2007). Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2007)*. Morgan Kaufmann.

Chapter 14

AN EMPIRICAL STUDY OF MULTI-OBJECTIVE ALGORITHMS FOR STOCK RANKING

Ying L. Becker[1], Harold Fox[2], Peng Fei[1]

[1]*Advanced Research Center, State Street Global Advisors, Boston, MA 02111.*
[2]*MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA 02139.*

Abstract Quantitative models for stock selection and portfolio management face the challenge of determining the most efficacious factors, and how they interact, from large amounts of financial data. Genetic programming using "simple objective" fitness functions has been shown to be an effective technique for selecting factors and constructing multi-factor models for ranking stocks, but the resulting models can be somewhat unbalanced in satisfying the multiple objectives that portfolio managers seek: large excess returns that are consistent across time and the cross-sectional dimensions of the investment universe. In this study, we implement and evaluate three multi-objective algorithms to simultaneously optimize the information ratio, information coefficient, and intra-fractile hit rate of a portfolio. These algorithms – the constrained fitness function, sequential algorithm, and parallel algorithm – take widely different approaches to combine these different portfolio metrics. The results show that the multi-objective algorithms do produce well-balanced portfolio performance, with the constrained fitness function performing much better than the sequential and parallel multi-objective algorithms. Moreover, this algorithm generalizes to the held-out test data set much better than any of the single fitness algorithms.

Keywords: multi-objective algorithm, equity market, stock selection, quantitative asset management

1. Introduction

A successful investment strategy requires maximizing investment opportunities while minimizing risks. Active quantitative investment is a proven approach for doing both. Underlying a quantitative strategy is a quantitative model that ranks assets based on a variety of metrics to help portfolio managers (PMs) make decisions to buy or sell securities. Usually, multiple criteria have to be

taken into account simultaneously to avoid biased decisions. The previous study by (Becker et al., 2006) has demonstrated that an advanced computer algorithm, genetic programming (GP), is an effective technique for selecting factors and constructing multi-factor models for ranking stocks according to a single objective fitness function. Yet, to more accurately reflect the complexity of investment reality, using a multi-objective fitness function is necessary to simultaneously satisfy different investment criteria.

The application of genetic programming has an increasing interest in the investment community. This natural evolution based technique arose from the initial genetic algorithm of (Holland, 1975) and developed by (Koza, 1992). GP has been successfully applied to a number of scientific areas such as biology, analog design, computational neuroscience and robotics. Early applications of GP in the investment industry include (Allen and Kajalainen, 1999) using a genetic algorithm to learn technical trading rules for the S&P 500 index. (Neely et al., 1997) use GP to construct technical trading rules in the foreign exchange markets. (Wang, 2000) applies GP to enhance trading and hedging in equity spot and futures markets. (Li and Tsang, 1999) propose a GP approach to combining individual technical rules and adapting the thresholds from past data. (Lawrenz and Westerhoff, 2003) use GP to develop a simple exchange rate model to get a deeper understanding of the forces that drive foreign exchange markets.

Researchers at State Street Global Advisors have used genetic programming for several different asset investment tasks: (Zhou, 2003) develops an effective emerging markets stock selection model combining traditional techniques with genetic programming. (Caplan and Becker, 2004) use GP to develop a stock ranking model for the high technology manufacturing industry in US. Recently, (Becker et al., 2006) have explored various single-objective fitness functions for genetic programming to construct stock selection models that satisfy investment philosophy specifics with respect to risk. These GP generated multi-factor models rank stocks from high to low according to their expected performance over the next quarter.

The contribution of this paper is to explore the GP capability of constructing stock selection models by using multi-objective algorithms. We have implemented three customized algorithms and evaluated their performance extensively. In our previous work, we have used two objective functions individually to judge the stock ranking produced by an individual formula. Both of these objective or fitness functions produce desirable, yet complementary formulas. In this study we design various algorithms that use the two fitness functions jointly to develop stock ranking models containing the favorable attributes considered by the portfolio managers in the investment process. The first fitness function is information ratio (IR) shown in Equation 14.1 and Equation 14.2. For example, based on 500 stocks in each time period, we construct a portfolio that is long on

the predicted top 10% of stocks and short on the predicted bottom 10%. This portfolio produces a return over a three-month horizon. The information ratio for formula f is the annualized average return for these portfolios divided by the annualized standard deviation.

$$Spread_t(f) = \frac{\sum_{i=1}^{50} R_t(\alpha_{i,t}) - \sum_{i=451}^{500} R_t(\alpha_{i,t})}{50} \tag{14.1}$$

$$IR(f) = \frac{\overline{(Spread(f))}_{annual}}{(\sigma_{Spread(f)})_{annual}} \tag{14.2}$$

where $\alpha_{i,t}$ is the ith ranked stock at time *t* according to the current formula *f*, and $R_t(s)$ is the three-month return for stock *s* at time *t*.

The second fitness function we have used is the information coefficient (IC) in Equation 14.3 and Equation 14.4. This is the Spearman rank correlation between a formula's predicted ranking and the actual, empirical ranking of the stock returns.

$$IC_t(f) = \frac{\sum_{s=1}^{500}(\rho_{s,t} - \overline{\rho_t})(r_{s,t} - \overline{r_t})}{\sqrt{\left(\sum_{s=1}^{500}(\rho_{s,t} - \overline{\rho_t})^2\right)\left(\sum_{s=1}^{500}(r_{s,t} - \overline{r_t})^2\right)}} \tag{14.3}$$

$$IC(f) = \frac{\sum_{t=1}^{T} IC_t(f)}{T} \tag{14.4}$$

where $\rho_{s,t}$ is the ranking for stock *s* at time *t* predicted by our formula f and $r_{s,t}$ is the ranking observed by sorting the stock returns.

The baseline for stock prediction performance is a linear model that is constructed based on multi-variable regression method. For example, we could run an ordinary least squares regression of stock returns relative to the factors that are used to forecast stock future returns. In our previous work, formulas produced by genetic programming have performed better than this standard linear model. However, these GP-developed formulas have suffered from two issues: first, they do not generalize consistently to novel data (i.e. data not used in the training process); second, formulas trained to maximize information ratio have disappointing information coefficients and vice versa.

The first issue is extremely serious regarding the robustness of the method. The second issue is also critical, since investors and PMs expect to see an

information coefficient that fits with their expectation of how a well-performing portfolio generator should behave. For the S&P 500, a good information coefficient is approximately 0.07.

In our latest GP experiments, we have attempted to combine these fitness measures to generate a formula that produces more robust and more balanced results. In addition to IR and IC, we have also used the Intra-Fractile Hit Rate (IFHR) as a judge of ranking performance. This is a measure of the accuracy of the formula's ranking. Of the top 10% of stocks in the ranking, we count the number that performed better than the average return. Of the bottom 10%, we count those that performed worse than the average. The IFHR is the sum of these counts divided by the total number of stocks in the top and bottom 10%.

$$IFHR_t(f) = \frac{\sum_{i=1}^{50} sgn(R(\alpha_{i,t}) - M_t) + \sum_{i=451}^{500} sgn(M_t - R(\alpha_{i,t}))}{100} \quad (14.5)$$

$$IFHR(f) = \frac{\sum_{t=1}^{T} IFHR_t(f)}{T} \quad (14.6)$$

where M_t is the average return of all stocks at time *t*. $sgn(x)$ denotes the operator that is 1 when x $\geq$ 0 and 0 when x $<$ 0. The IFHR is the percentage of our stock picks which were correct. For the S&P 500, a good value for the IFHR is above 55%.

The challenge of the study is to combine these three fitness functions, IR, IC, and IFHR into one algorithm to select one good formula. Since IR, IC, and IFHR are conceptually highly related, we believe that it is possible to find a formula that is optimal across all three measures, and our results broadly support this hypothesis. Optimizing multiple objectives is a new and active area of research in genetic programming and genetic algorithms. In general, these algorithms maintain a Pareto front of candidate genomes, each of which is no worse than any other. That is, genome g1 may beat genome g2 on fitness function f1. But g1 may lose to g2 on fitness function f2. Multi-objective algorithms hold the potential to improve formula generalization as well as performance breadth. (Bleuler et al., 2001) empirically demonstrate that the SPEA2 algorithm applied to genetic programming produces smaller genome trees. This is highly desirable, since smaller trees generalize better and are thus less likely to overfit the training data. The result is intuitive, since a formula that must perform well on several measures ought to be smoother and more uniform than one that targets a single objective.

Two powerful algorithms for multi-objective evolution are the Strength Pareto Evolutionary Algorithm 2 developed by (Zitzler et al., 2001) and the

Non-Dominated Sorting Genetic Algorithm 2 developed by (Deb et al., 2002). In our initial experiments, we have implemented a simpler set of algorithms than these Pareto front methods that are the state of the art in modern multi-objective optimization. We wanted simple algorithms, so that we could quickly test whether multi-objective algorithms were an effective technique for generating more robust formulas. Also, our requirements are somewhat different from the typical multi-objective case. For us, information ratio is the most important, predominating portfolio measurement. IC and IFHR serve more as constraints than as objectives to be optimized in themselves.

In this empirical study, we will discuss our initial approaches to incorporating IR, IC and IFHR into our evolution process and then describe our specialized technique for avoiding overfitting. The results show that our constrained fitness multi-objective algorithm generalizes significantly better than a GP that just maximizes IR or IC on its own. On a held-out test data set, the winning formulas in this multi-objective evolution have IRs that are at least as good as the winning formulas from the process maximizing IR on its own. Finally, the ICs of the multi-objective winners are consistently better than the winning formulas in the single-objective evolution process. We also perform a detailed analysis of the top 4 formulas from all experiments. This analysis shows that the best two formulas are from two of the multi-objective algorithms.

The organization of this paper is as follows: Section 2 describes the data set that we have used for our experiments. Section 3 describes our methodology: the general GP setup and our multi-objective algorithms. Section 4 describes our results: summaries as well as detailed comparisons of sample formulas. Section 5 concludes and discusses future work.

2. Financial Data

We use Standard and Poor's 500 index as our research universe. We exclude the financials and utilities sectors, which have very unique business models, from our study universe in order to maintain the homogeneity of the underlying assets. The monthly financial data and stock return time series used for the research are retrieved from Compustat database for the period of January 1990 to December 2005. For each given date, there are around 350 investable stocks, each of which is described by sixty-five variables. On average, there are about 350*65 independent variables available for analysis. The financial data includes variables describing firms' fundamentals from balance sheets and income statements, earnings estimates, historical price returns and monthly market perspectives. GP selected factors are grouped into five categories in Table 14-1. Valuation factors measure firms' intrinsic value relative to market value. Growth factors provide information of firms' growth potentials. Quality factors measure firms' financial leverage, management efficiency or financial

profitability. Analyst opinion provides forward looking information about a stock' s future price movement. Momentum measures market sentiment reflected in recent price trends.

Table 14-1. Sample of Factor for Inclusion in Stock Selection Models.

Factor	Description
Valuation	
V_1	Cash flow generation
V_2	Normalized revenue generation
V_3	Growth adjusted valuation ratio
Growth	
G_1	Normalized net income growth version 1
G_2	Normalized net income growth version 2
G_3	Normalized revenue growth
Quality	
Q_1	Asset Utilization
Q_2	Financial leverage version 1
Q_3	Financial leverage version 2
Q_4	Financial leverage version 3
Q_5	Financial leverage version 4
Q_6	Profitability
Q_7	Operating efficiency
Analyst	
E_1	Analyst opinion version 1
E_2	Analyst opinion version 2
Momentum	
P_1	Short term price momentum
P_2	Long term price momentum

3. Methodology

Basic Genetic Programming Setup

First, we will describe the starting point for our genetic programming setup. The primary problem with using GP to predict stock performance is data overfitting. Stock data contain many subtle dependencies, and it is easy to find formulas that work over one time period and fail over another. Also, the US stock market is close, if it is not, to efficient, so almost all data that could predict stock performance are already incorporated in the stock price. Most of the vari-

ation in stock returns is unpredictable noise. Thus, it requires a well designed searching algorithm to identify the signals versus the noise.

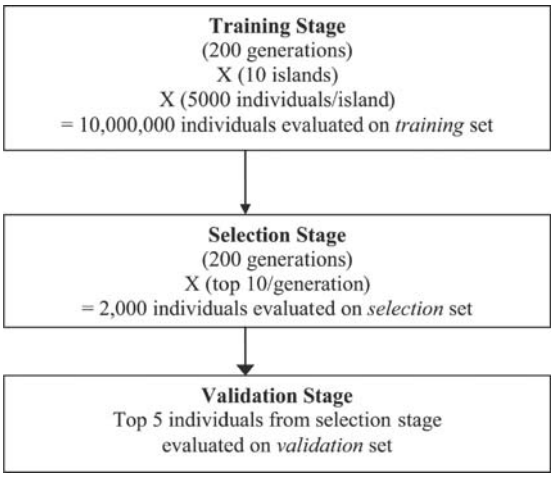


Figure 14-1. Multi-stage selection process for typical parameter values

Our special approach to using GP in stock selection, shown in Figure 14-1, is to use a multi-stage selection process. At each generation, we select the fittest individuals based on their performance on the training set. This training set is a random subset of dates from our complete date set. We then evaluate those individuals on a completely new date set, the selection set. We take the top 5 formulas from this selection period and evaluate them on a third date set, the validation set. Consistency across all three date sets indicates a robust combination of factors. We also prevent over fitting by penalizing large tree sizes in the fitness functions. Smaller trees are more regular and generalize better. In practice, the multi-stage selection process works well. The produced trees are simple and easy to understand, and their performance does not usually degrade significantly on the validation set.

Our system, built with GAlib (Wall, 2000), uses the island form of evolution, shown in Figure 14-2 – Figure 14-4. Each island has a population of p individual genomes. On each generation, each individual g is assigned a score based on its performance on the training data set. We use a tournament selection algorithm, picking two random individuals and selecting the one with the higher score. Propagation is done by selecting two individuals. With some probability, they perform crossover, exchanging a random sub-tree. Otherwise, they are propagated unchanged. With some mutation probability, sub-trees are dropped from the propagating individuals. On each generation, the top performers from

island i are moved to the island $i+1$ above it. This approach promotes variety of genomes as well as mixing of positive traits.

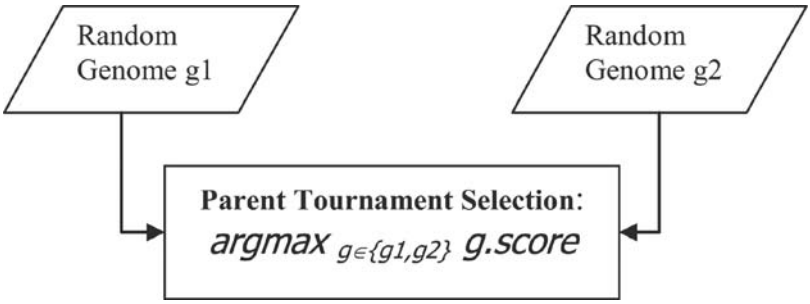


Figure 14-2. Randomized parent tournament selection.

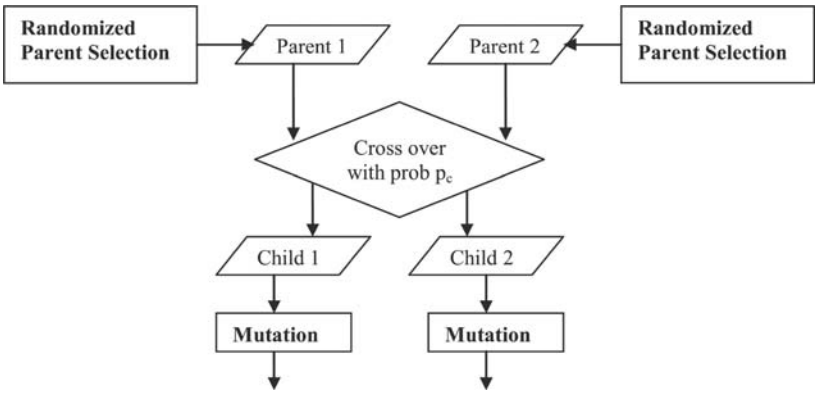


Figure 14-3. Evolution from one generation to the next.

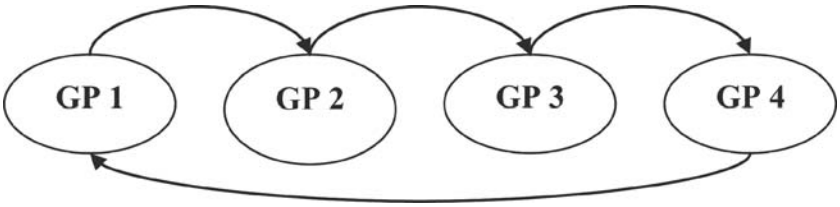


Figure 14-4. Island evolution: each island evolves independently of the others. After each generation, the top genomes migrate to the neighboring island.

Multi-Objective Algorithms

Constrained Fitness Function. We will now detail the three multi-objective algorithms that we have implemented and tested. They are the constrained fitness function, the sequential approach, and the parallel approach. The constrained fitness function method combines the three primitive fitness functions, IR, IC, and IFHR, into a composite fitness function. The sequential approach considers one primitive fitness function at a time. When the performance of the population flattens out between generations, the algorithm switches to the next primitive function. Finally, the parallel approach evolves different islands of genomes in parallel, each island using a different primitive fitness function. At each generation, the top performers from each island migrate to a neighboring island. In Section 4, we evaluate the relative performance of these algorithms by measuring their generalization performance and by studying in detail several of the formulas they produce.

For the composite fitness function, the simplest strategy is to form a linear combination of the three primitive fitness functions. However, this strategy misses the essential way that investors and portfolio managers evaluate the performance of a particular quantitative portfolio. Essentially, investors would like to maximize the information ratio of a portfolio subject to a reasonable performance on IC and IFHR. The problem is better characterized as maximizing IR subject to IC and IFHR above certain thresholds. Past experience with our S&P 500 data set has told us that a good portfolio should have an IC greater than 0.07, and it should have a hit rate (IFHR) greater than 0.55. Experience also tells us that a good IR should lie between 1.5 and 2 for three-month returns. Thus, an initial candidate for the constrained fitness function for a portfolio p is:

$$C(p) = IR(p) + H(IC(p) - 0.07) + H(IFHR(p) - 0.55) \qquad (14.7)$$

where $H(x)$ is the unit step function: $H(x) = 1$ for $x \geq 0$ and $H(x) = 0$ for $x < 0$.

The problem with this fitness function is its discontinuity. Formulas that come very close to the threshold are weighted no better than formulas that are wide of the mark. Thus, we approximate the step function with the logistic or sigmoid function, $S_k(x) = \frac{1}{1+e^{-\frac{x}{k}}}$. See Figure 14-5. The curve is steep when x is close to 0 and flat when x is far from zero. The parameter k determines how steep the curve is.

Our final fitness function is:

$$C(p) = IR(p) + S_{0.02}(IC(p) - 0.07) + S_{0.05}(IFHR(p) - 0.55) \quad (14.8)$$

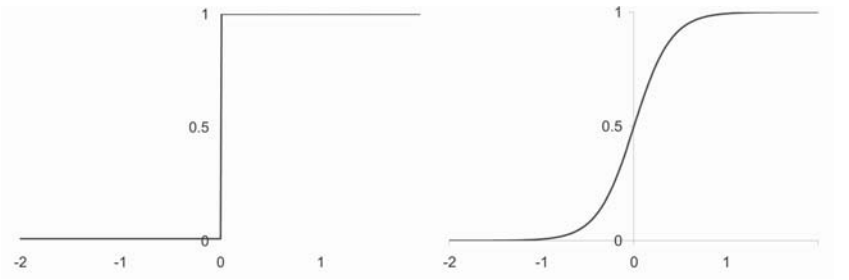


Figure 14-5. Step function $H(x)$ and logistic function $S_k(x)$.

We use this composite function to evaluate formulas in every GP stage exactly the same way as we use the primitive fitness functions.

Sequential Approach. The sequential and parallel approaches modify the GP process in a more radical way. The sequential approach rotates among the three primitive fitness functions over time. When the population of genomes stabilizes under the current fitness function, the algorithm rotates to the next fitness function. The rationale is that GP will discover components that perform well for a particular primitive function. It could keep these components while it searched for complementary components to evolve under the next fitness function. The final formula should optimize all three of the primitive functions. The approach is inspired by the punctuated equilibrium theory of biological evolution, whereby genome evolution occurs in spurts in response to dramatic environmental changes that radically redefine a population's natural selection bias.

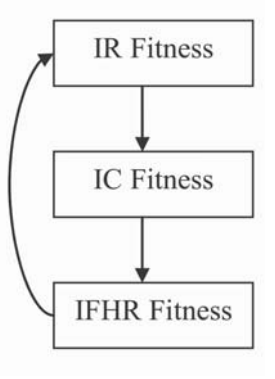


Figure 14-6. Sequential Fitness Function Switching.

At each time step, we compute the average top fitness, which is the average of the fitness values for the number one genomes from each island. We also compute the best fitness value from all islands. We switch fitness functions when the average top fitness has declined over the past *k* time periods and when the best fitness for the entire time period of this fitness function has occurred at least *k* time periods before. That is, avg_{t-k} is at least as big as $avg_{t-k+1}...avg_t$ and $BestEver_{t-k}$ is at least as big as $BestEver_t$.

Parallel Approach. In the parallel approach, we assign different fitness functions to different islands. The approach is similar to the Vector Evaluation Genetic Algorithm (Schaffer, 1985). After each generation, the top *m* genomes migrate to the neighboring island above them. The rationale is that genomes that do well under one fitness function can share code with genomes doing well under a different fitness function.

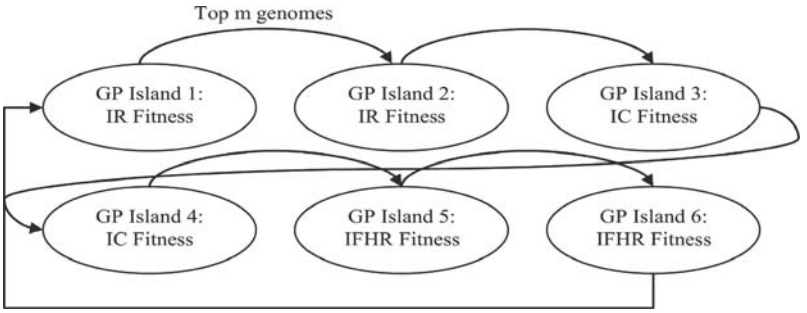


Figure 14-7. Parallel Evolution.

In the selection stage, for each generation, we evaluate the number one performer from each island using the constrained fitness function, described above, on the held-out selection data set. The top 5 genomes from the constrained fitness function on this data set constitute the final output of the algorithm.

4. Results and Discussion

To test our algorithms, we run three large evolutions for each algorithm: the three primitive fitness functions and the three multi-objective programs. Each evolution returns the top 5 candidates. Thus, there are 6 GP fitness algorithms, each of which returns 15 top individual stock ranking formulas. We test each of these individuals on the same held-out validation set. We then select the best overall formula from the IR fitness function and from each of the multi-objective algorithms. Finally, we perform a detailed financial analysis for each of these 4 formulas. The overall results and the detailed analysis show that the

constrained fitness multi-objective algorithm produces formulas that generalize significantly better than the formulas produced by the single fitness algorithms.

Genetic Programming Output

Table 14-2 shows the mean and standard deviation of IR, IC, and IFHR for each program on the held-out validation data set. The constrained fitness function algorithm performs best. Its output formulas have higher average IR than those of the single IR fitness program. Its average IC is larger than the average IC for the single IC fitness program. The standard deviation is also lower, indicating that the constrained method is more stable. The difference in average IR between the single IR program and the constrained program is not statistically significant, but the difference in IC is statistically significant. Average results from the constrained program are close to the goal of a 0.07 IC and a 55% IFHR rate.

Table 14-2. Validation set results for the top formulas for each program.

	IR		IC		IFHR	
Fitness Functions	Mean	Std Dev	Mean	Std Dev	Mean	StdDev
Single Fitness Functions						
Single (IR)	1.22	0.40	0.061	0.014	53.5%	1.3%
Single (IC)	0.75	0.19	0.063	0.005	53.3%	0.6%
Single (IFHR)	0.13	0.14	0.006	0.005	54.9%	0.5%
Multiobjective Algorithms						
Constrained	1.28	0.23	0.068	0.012	53.8%	0.9%
Sequential	1.21	0.23	0.063	0.015	53.4%	1.0%
Parallel	1.11	0.22	0.067	0.004	53.8%	0.4%

The IFHR fitness program produces very poor results. It is likely that the returned formulas overfit the data. They only capture patterns that hold for certain stocks in certain time periods. But these formulas lack consistent performance and they would not be suitable investment strategies under any circumstances.

A more important result is that the constrained multi-objective algorithm appears to produce formulas that generalize much better than any of the single fitness algorithms. Table 14-3 and Table 14-4 show the average generalization performance from the selection stage to the validation stage. Since the output formulas were chosen as the top performers on the selection data set, we expect that performance will degrade on the validation set. Erosion measured by percentage change is used to measure the degree of degradation from selection to validation. The smaller the performance erosion, the greater confidence we can have that our algorithm is finding robust formulas that capture the true underlying patterns in the data. Column 3 of Table 14-3 shows that the performance

degrades by 23.8% for the single IR fitness function. For the constrained fitness function, the performance degrades by only 12.3%. Similarly, the average erosion from selection IC to validation IC is 31.9% for the top single IC fitness functions. The top constrained formulas decrease by only 5.2% on average, and there is no erosion in sequential and parallel formulas. This generalization result is very important, since it gives us confidence that the output formulas of the constrained algorithm will work well in the future. The results validate our hypothesis that a multi-objective algorithm would produce more robust results in addition to more balanced results.

Table 14-3. Differences in IRs from selection to validation set.

	Selection		Validation		Erosion	
Fitness Functions	Mean	Std Dev	Mean	Std Dev	Mean	StdDev
Single Fitness Functions						
Single(IR)	1.62	0.18	1.22	0.40	23.8%	26.3%
Multiobjective Algorithms						
Constrained	1.48	0.14	1.28	0.23	12.3%	20.0%
Sequential	1.65	0.25	1.21	0.23	26.2%	13.5%
Parallel	1.51	0.27	1.11	0.22	25.7%	10.3%

Table 14-4. Differences in IRs from selection to validation set.

	Selection		Validation		Erosion	
Fitness Functions	Mean	Std Dev	Mean	Std Dev	Mean	StdDev
Single Fitness Functions						
Single(IR)	0.092	0.002	0.063	0.005	31.9%	4.6%
Multiobjective Algorithms						
Constrained	0.072	0.012	0.068	0.012	5.2%	16.7%
Sequential	0.061	0.011	0.063	0.015	-4.7%	21.8%
Parallel	0.070	0.015	0.067	0.004	-1.7%	25.8%

Figure 14-8 shows a scatter-plot of IR versus IC on the validation data set for the outputs of the single IR, single IC, constrained, sequential, and parallel formulas. First, there is a clear trend between a good IC and a good IR. Second, the top constrained formulas are more tightly clustered than the top single IR formulas, and they have higher IC's. In this plot, the formula with the highest validation IR comes from the single IR fitness algorithm. However, after performing a more detailed financial analysis of the top constrained formula with the top single IR formula using full data set, the top constrained formula actually performs better.

Of the three tested multi-objective algorithms, the constrained fitness function produces better, more robust formulas than either of the sequential or parallel approaches. The constrained fitness function selects formulas with good combined performance at every stage of sorting and selection. The parallel algorithm only combines the fitness measures at the selection stage, and the sequential algorithm only considers multiple objectives across many generations. We believe that to obtain robust formulas, a genetic programming algorithm must consider all objectives in the training stage. Sorting formulas by multiple criteria at the earliest GP stages is necessary to weed out formulas that overfit.

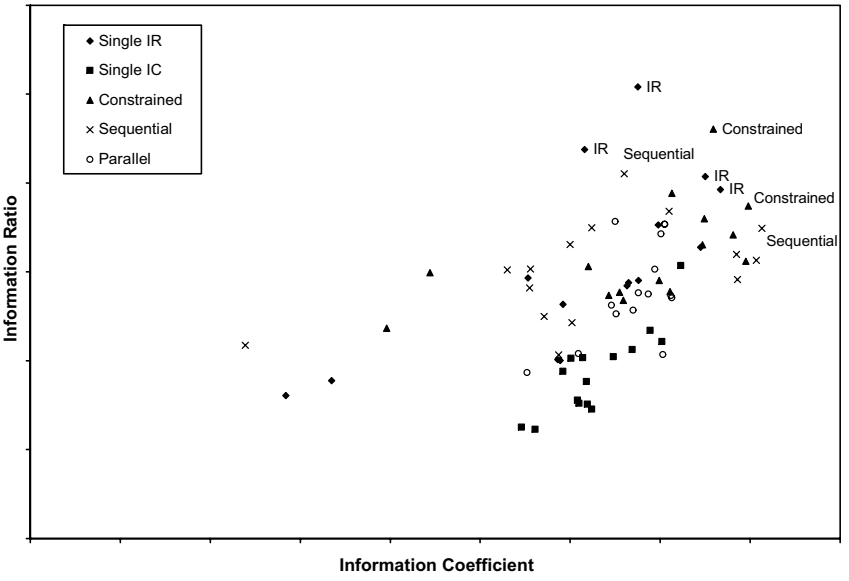


Figure 14-8. Scatter plot of IC versus IR using validation data for the top formulas for each program.

The sequential algorithm exhibits some unusual and interesting behavior. For long periods, when IR is the objective function, it operates exactly like the single IR fitness algorithm. However, it also spends a considerable amount of time searching in the optimal formula spaces of the other fitness functions. Thus, the sequential algorithm is potentially exploring a much larger set of formulas than any single fitness function algorithm. Therefore, the sequential algorithm can also require many more generations to converge. Most of the top candidates that the sequential algorithm finds are discovered in much later generations than the other algorithms. We hypothesize that the sequential approach could work better over much longer searching times.

Using the scatter plot in Figure 14-8, we select the best model from the single IR, constrained, sequential and parallel programs as representatives for detailed model evaluation. They are partially described in Equation 14.9 to Equation 14.12.

$$ModelA(SinglefitnessIR) = f(V1, V2, G2, Q1, Q3, Q4, E1) \quad (14.9)$$

$$ModelB(Constrainedfitness) = f(V1, Q4, Q5, Q6, E1, E2, S1) \quad (14.10)$$

$$ModelC(Sequentialfitness) = f(V1, V2, G2, Q1, Q3, Q4, E1) \quad (14.11)$$

$$ModelD(Parallelfitness) = f(V1, V2, G2, Q1, Q3, Q4, E1) \quad (14.12)$$

Model Evaluation

Statistical Signifiance. In order to evaluate the efficacy of each model in ranking stocks, we compare their statistical significance using ICs and corresponding p-values reported in Table 14-5. Since a good model needs to be reliable and stable over time, we summarize the statistics for different portfolio holding periods from one month through twelve months. For each individual model and each holding period, the correlation between the models and the future returns are all positive and statistically significant. It indicates that the models generated by different GP fitness functions have predictive power in ranking stocks' future returns. Using one month ICs as an example, it is shown that three models from multi-objective GP algorithm (constrained, sequential and parallel) all outperform the one from single-objective algorithm (using IR). The outperformance is consistent across different holding periods.

Table 14-5. Information Coefficients of models with forward returns (1/3/6/12 months).

Information Coefficients		Representative Models from Different Fitness Functions			
		Single(IR)	Constrained	Sequential	Parallel
1-month	Mean	3.09%	5.06%	3.44%	3.66%
	Std	8.93%	8.01%	7.41%	10.29%
	P-value	<0.0001	<0.0001	<0.0001	<0.0001
3-month	Mean	5.21%	6.99%	5.06%	5.66%
	Std	9.74%	8.45%	7.97%	11.64%
	P-value	<0.0001	<0.0001	<0.0001	<0.0001
6-month	Mean	6.05%	7.89%	6.65%	7.05%
	Std	9.89%	8.42%	8.38%	11.78%
	P-value	<0.0001	<0.0001	<0.0001	<0.001

**Universe: S&P 500 excluding financials and utilities*
**Period: February 1990 to December 2005*

Historical Portfolio Simulation. Another important measure of a model's stock selection power is simulating the historical returns of a portfolio constructed based on the stock selection model. A long-short portfolio return is generated by buying the top fractile and short selling the bottom fractile stocks. The historical portfolio simulations including statistics of monthly portfolio returns' mean, std, IR, Hit Rate and IFHR for each model are summarized in Table 14-6. A portfolio's hit rate is the percentage of time periods when the portfolio has a positive return. The models from the constrained and sequential GP algorithms perform better relative to the model from the single IR fitness function in almost every aspect. The parallel algorithm's model shows better mean portfolio spread returns and slightly higher IFHR than the single IR algorithm's model, but smaller IR and hit rate.

Table 14-6. Total portfolio returns from buying top decile and short selling bottom decile stocks.

Decile Spread Returns		Representative Models from Different Fitness Functions			
		Single(IR)	Constrained	Sequential	Parallel
1-month	Mean	1.21%	1.81%	1.59%	1.61%
	Std	2.97%	3.76%	3.21%	4.01%
	IR	1.41	1.67	1.72	1.39
	Hit Rate	64%	71%	72%	65%
	IFHR	52%	53%	52%	53%
3-month	Mean	3.27%	4.47%	4.41%	3.88%
	Std	5.25%	6.90%	5.91	7.16%
	IR	1.25	1.29	1.49	1.08
	Hit Rate	76%	74%	79%	68%
	IFHR	53%	55%	54%	53%
6-month	Mean	5.10%	6.74%	7.76%	6.98%
	Std	7.05%	9.46%	9.03%	10.27%
	IR	1.02	1.01	1.21	0.96
	Hit Rate	79%	80%	83%	77%
	IFHR	53%	55%	55%	54%

**Universe: S&P 500 excluding financials and utilities*
**Period: February 1990 to December 2005*

We can also simulate the cumulative returns investors would achieve when they design a portfolio by buying top decile stocks and a portfolio selling bottom decile stocks, which are shown in Figure 14-9 and Figure 14-10, respectively. The benchmark is the equal weighted S&P 500 stock returns excluding financials and utilities. These portfolios are rebalanced monthly without turnover constraint and transaction costs. In each figure, five portfolios are created: single (IR), constrained, sequential, parallel and the benchmark. Over the sixteen year simulation period, all the buying portfolios based on models from GP outperform the benchmark, and all the selling portfolios based on models from

GP underperform the benchmark. Among the portfolios of buying top decile stocks, parallel has the highest cumulative return relative to the benchmark and single (IR) has the lowest cumulative return relative to the benchmark. Among the portfolios of selling bottom decile stocks, both sequential and constrained have the lowest cumulative return relative to the benchmark and single (IR) has the highest cumulative return relative to the benchmark.

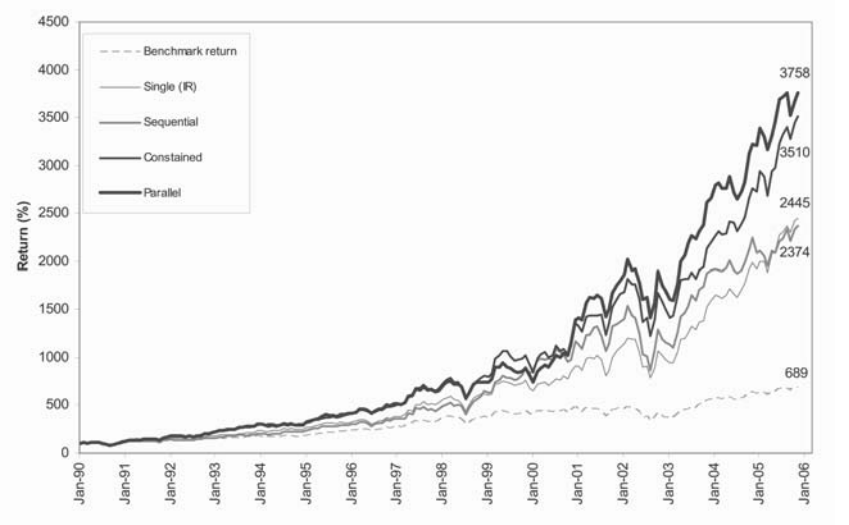


Figure 14-9. Total cumulative returns of longing decile portfolios vs. benchmark.

Model Consistency. To test model stability through time, we calculate 1-month and 3-month auto-correlations of four models generated by single or multi-objective fitness functions. If the forecasting capabilities of signals decay rapidly, the portfolio manager must trade often to ensure owning the best securities and selling the worst most of the time. However, trading is not free and transaction costs need to be taken into account for model performance evaluation. High turnover models require a high cost execution. Table 14-7 shows the pace at which the signals decay over time and should coincide with the turnover of the portfolio. The model from the single IR fitness function has the highest auto-correlations for one month and three months horizons. Although the previous analysis shows that multi-objective fitness functions achieve pre-defined goals (IR, IC, and IFHR) better relative to the single fitness function (IR), the improvement has a tradeoff in sacrificing a higher transaction cost, a factor not considered in the GP fitness functions.

A balanced model should work robustly in different investment market conditions such as whether the market favors growth or value stocks. Table 14-8

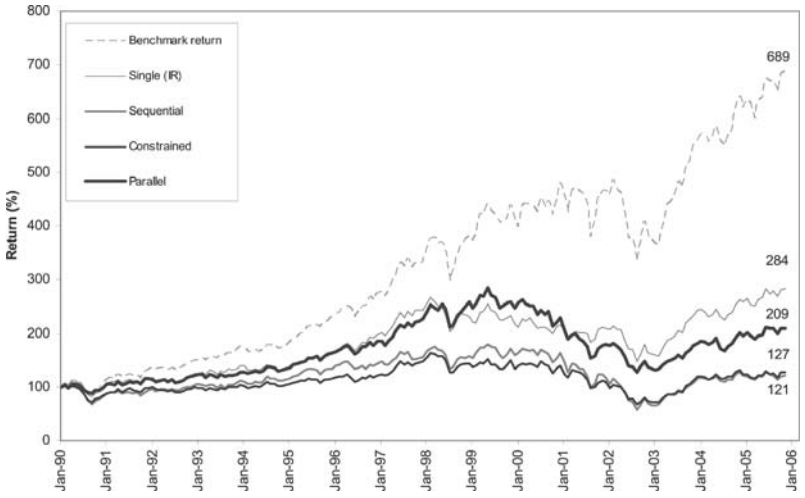


Figure 14-10. Total cumulative returns of shorting decile portfolios vs. benchmark.

Table 14-7. Auto Correlation of Models.

Fitness	Correlation with 1 month Lag			Correlation with 3 month Lag		
Functions	Mean	StdDev	P-value	Mean	StdDev	P-value
Single (IR)	93.4%	5.3%	<0.0001	82.2%	3.5%	<0.0001
Constrained	83.5%	5.3%	<0.0001	67.5%	4.5%	<0.0001
Sequential	84.2%	6.0%	<0.0001	72.3%	5.6%	<0.0001
Parallel	91.4%	7.5%	<0.0001	75.8%	4.1%	<0.0001

**Universe: S&P 500 excluding financials and utilities*
**Period: February 1990 to December 2005*

shows that the single (IR) model is slightly biased toward value periods. The constrained model overall shows better spread returns and IR across different holding periods, but the enhancement comes mainly from the value period with worse results in growth periods. The parallel model similarly has very unbalanced performance with growth and value periods. The sequential model has more balanced performance with strong gains in both value and growth market regimes. Since the balance of portfolio performance in different market regimes is not included as an objective in our GP programs, it is natural that performance is not achieved homogenously across different market conditions.

Table 14-8. Model robustness evaluation in Growth and Value market regimes.

Market Regime*	Single(IR) Avg [%]	SD [%]	IR	Constrained Avg [%]	SD [%]	IR	Sequential Avg [%]	SD [%]	IR [%]	Parallel Avg [%]	SD [%]	IR
1m Forward Return Spreads												
Growth	1.04	3.11	1.16	1.18	3.59	1.14	1.76	2.78	2.20	0.82	3.75	0.76
Value	1.38	2.84	1.69	2.44	3.85	2.20	1.42	3.60	1.37	2.41	4.12	2.03
Total	1.21	2.97	1.41	1.81	3.76	1.67	1.59	3.21	1.72	1.61	4.01	1.39
3m Foward Return Spreads												
Growth	2.76	4.70	1.17	3.13	6.47	0.97	4.55	5.42	1.68	2.27	6.77	0.67
Value	3.79	5.72	1.32	5.91	7.05	1.67	4.27	6.40	1.33	5.51	7.20	1.53
Total	3.27	5.25	1.25	4.47	6.90	1.29	4.41	5.91	1.49	3.88	7.16	1.08
6m Foward Return Spreads												
Growth	4.74	6.48	1.03	5.91	9.20	0.91	8.37	9.20	1.29	5.71	9.42	0.86
Value	5.45	7.60	1.01	7.69	9.66	1.13	7.15	8.87	1.14	8.26	10.96	1.07
Total	5.10	7.05	1.02	6.74	9.46	1.01	7.76	9.03	1.21	6.98	10.27	0.96

**Universe: S&P 500 excluding financials and utilities*
**Period: February 1990 to December 2005*

5. Conclusion

The core of a quantitative investment strategy is a quantitative model that ranks the assets based on the likelihood of their excess returns against a relative benchmark, or forecasts assets' future returns. The biggest challenge is to develop a model that can capture multiple key characteristics of complex reality, each of which reflects the desired behavior. Our previous study (Becker et al., 2006) has demonstrated that GP is an effective technique for selecting factors and constructing multi-factor models for ranking stocks according to a single objective fitness function. However, empirical results have shown that such a GP-developed model does not have a well-balanced performance on the multiple investment criteria a portfolio manager would consider. This paper explores a variety of multi-objective genetic programming algorithms that aim to satisfy the different investment criteria simultaneously.

In this study, we implement and evaluate three multi-objective algorithms to simultaneously optimize the information ratio, information coefficient, and intra-fractile hit rate of a portfolio. These algorithms, the constrained fitness function, sequential algorithm, and parallel algorithm take widely different approaches to combining these different portfolio metrics. Our results indicate that the multiple objective functions can result in a more robust model to deal with the data overfitting issue. It is shown that the constrained fitness function produces the best formulas with the most robust performance on novel data. This simple multi-objective fitness function performs better than the other two, more complicated multi-objective algorithms. The very top formula from this constrained algorithm performs well in different market regimes, different market segments, and across different time periods. The factor components are logical, and the formula's decision-making is clear and rational to a port-

folio manager. Our results also indicate that the performance improvement of multi-objective functions are not always homogenously distributed since there remain some investment criteria not included in the fitness function.

In future work, we plan to evaluate the more recent multi-objective algorithms such as SPEA2. Since this algorithm combines all objectives in the training stage, it would possibly generalize better than the sequential and parallel approaches that we have considered here.

Nevertheless, the constrained fitness function consistently produces excellent results, and we believe that it will be difficult to surpass this algorithm's performance. However, the parameters of this fitness function were hand-tuned from a deep familiarity with the S&P 500 domain. It would be ideal to have an algorithm that could produce the same performance in a less ad hoc, more principled way.

Acknowledgements

We gratefully appreciate Anna Lester, portfolio manager of US Active Equity group, for her valuable comments on investment insights, and Mark Hooker, Senior Managing Director of the Advanced Research Center at State Street Global Advisors, for his valuable discussion and support.

References

Allen, F. and Kajalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, 51:245–271.

Becker, Ying, Fei, Peng, and Lester, Anna M. (2006). Stock selection : An innovative application of genetic programming methodology. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages –. Springer, Ann Arbor.

Bleuler, Stefan, Brack, Martin, Thiele, Lothar, and Zitzler, Eckart (2001). Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.

Caplan, Michael and Becker, Ying (2004). Lessons learned using genetic programming in a stock picking context. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 6, pages 87–102. Springer, Ann Arbor.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *IEEE Transactions of Evolutionary Computation*, 6(2):182–197.

Holland, J.H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Lawrenz, C. and Westerhoff, F. (2003). Modeling exchange rate behavior with a genetic algorithm. *Computational Economics*, 21:209–229.

Li, J. and Tsang, E.P.K (1999). Improving technical analysis predictions: an application of genetic programming. In *Proceedings of Florida artificial intelligence research symposium*.

Neely, Christopher J., Weller, Paul A., and Dittmar, Rob (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405–426.

Schaffer, J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100.

Wall, M. (2000). Galib: A c++ library of genetic algorithm components.

Wang, J. (2000). Trading and hedging in s&p 500 spot and futures markets using genetic programming. *Journal of Futures Markets*, 20(10):991–942.

Zhou, A. (2003). Enhanced emerging market stock selection. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice I*. Kluwer.

Zitzler, E, Laumans, M, and Theile, L (2001). Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Swiss Federal Institute of Technology (ETH), Zurich. Computer Engineering and Networks Labratory.

Chapter 15

USING GP AND CULTURAL ALGORITHMS TO SIMULATE THE EVOLUTION OF AN ANCIENT URBAN CENTER

Robert G. Reynolds[1], Mostafa Z. Ali[1] and Patrick Franzel[1]

[1] *Wayne State University 5143 Cass Avenue, 431 State Hall, Detroit, MI 48202 USA*

Abstract Numerous models of modern and ancient urban landscapes have been proposed. While is of interest to classify examples of early urban centers, it is even more interesting to model their origins. Since these emergent centers can be viewed not only as adaptations to their social and biological environments, but also as a source of further change. Thus, the meaning or semantics of an emergent center reflects the processes by which it was formed. In the study of one ancient center, Monte Albán, we have used data mining techniques to extract a large number of decision trees describing its settlement over time. Each decision tree specifies the values for selected attributes that can predict settlement activity in the terraces that comprise the site. However, not all terraces with those properties are occupied in the same way. This can result from economic or social reasons. Also, the variables employed do not always have the semantics needed to make sense of the distinction between occupied and unoccupied terraces. In this paper we focus on the latter reason. Here, GP and CA are used to add semantics to rules to make them more understandable to experts in the area. Future work will examine how GP can be used to integrate social and economic constraints in with the basic decision tree rules.

Keywords: Cultural Algorithms, Decision Trees, Data Mining, Urban Origins

1. Introduction

Genetic Programming has proven successful in the development of programmer expertise. When the programming activity relates to an intensive domain-based application, the integration of domain knowledge into the programming process can be an important contributor to the success of the application. Even the most skilled programmer can learn from the heuristics that derive from the use of knowledge in a given domain.

Embedding the Genetic Programming process within the Cultural Algorithm allows the Genetic Programming system to evolve programs within the knowledge associated with a given application domain (Blanton, 1978). For example, in previous work this synergy was demonstrated in terms of a problem in modern economic systems (Ostrowski and Reynolds, 2003). There, a technique designed to optimize automobile sales based upon buyer incentives was modified to reflect changes in assumptions made about consumer knowledge.

This paper applies a hybrid Cultural Algorithm and Genetic Programming system to a problem associated with prehistoric societies. In particular, the goal is to model the evolution of archaic urban areas. While ubiquitous now, urban systems have emerged only within the last 2000 years. It is of interest to see the extent to which our models and understanding of modern urban centers carries over to these archaic emergent centers. In our approach, domain knowledge will be extracted from an extensive archaeological database using data mining techniques in order to develop models of agent behavior that led to the observed morphological features of early sites.

While there are a number of examples of archaic cities, we select Monte Albán, a city that evolved in the Valley of Oaxaca in Central Mexico around 500 B.C. This site was selected because, unlike many ancient sites, its material remains are relatively well-preserved. As a result, extensive archaeological surface surveys have been conducted as well numerous excavations. The site of Monte Albán covers over 300 hectares with 1/5 of that area densely occupied. This site has produced a database of more than 2000 terraces where each terrace constitutes an occupational unit within the Monte Albán site. For each terrace over 200 variables are available to describe the terrace location and the cultural artifacts that it contains. In addition, several hundred variables were used to describe the pottery content in each terrace.

Figure 15-1 gives a view of the central plaza for Monte Albán, a hilltop site. Early settlement took place on the hilltop in the central plaza area. The site contains several ceremonial centers as well as many cultural artificats. Figure 15-2 shows a sample artifact.

In section 2 we begin by describing the temporal and spatial dimensions of the site. Next, in section 3 we briefly describe the data mining techniques used to generate the rules that will form the raw material for the GP system here

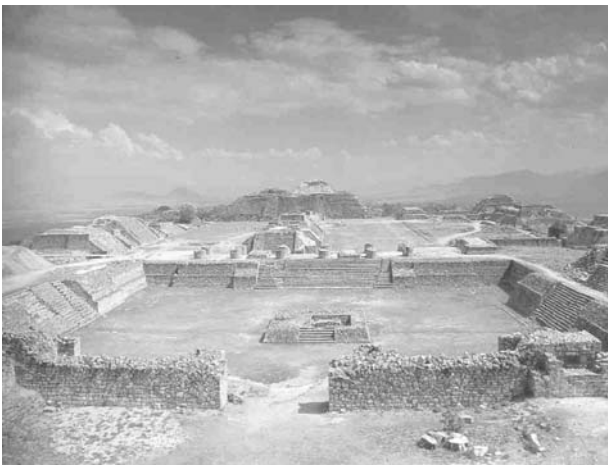


Figure 15-1. The central plaza area of Monte Albán.

Figure 15-2. A sample stone artifact from Monte Albán

and describe the rule extraction process. As it turned out the extracted rules, while useful in predicting terraces occupation, had some problems. First, the variables used in the rule did not always have the semantics needed to allow users a meaningful understanding of why terraces were located where they were, even though the rule was a good predictor. Second, there were often more desirable sites available than were occupied at a given time period. This suggested that social and economic context was important as well in determining location. In this paper we focus on the former criterion. In section 4 we discuss the use of GP and Cultural Algorithms to re-express the extracted rules in terms of semantically more meaningful variables. These rules will then be used to drive future agent-based models that simulate urban emergence. Section 5 gives our conclusions and suggests future applications of GP to knowledge integration at the site.

2. The Monte Albán Example

While the basic occupational phases of the valley extend from the early village formation in Tierras Largas to Monte Albán V, the urban site of Monte Albán emerged in period 1a and continued to be occupied through Monte Albán V. So in generating our data mining framework we will begin with Period 1a and ignore the previous phases initially.

Table 2 gives all of the relevant periods of social evolution in the valley. Tierras Largas marks the beginning of early village settlement there. The state emerged at Monte Albán in period Monte Albán Ia. The valley came under control of the state by Monte Albán II, and Monte Albán IIIa signaled the decline of the state and its succession by a collection of city-states localized in different parts of the valley. The phases as described there represent uneven slices through time.

To illustrate our approach we will develop a set of rules to determine whether or not a terrace was likely to be occupied in each of the established periods (Ia, Ic, II, IIIa, IIIb-IV, and V). These rules will come from a dataset of 2073 terraces that were surveyed by Blanton (Blanton, 1978; Blanton et al., 1982). While many environmental and cultural variables were collected for each terrace at the site, only the environmental variables that describe terrace location will be examined here. The questions to be answered here include: What locational features of the environment were preferred for settlement at each time period? What are the consistencies in these locational rules from period to period? Are there more suitable sites in a period than are occupied?

The first thing that must be done is to determine which of the terraces in the database is occupied in each of the time periods. There are certain ceramics that are indicative of a specific time period based upon their stylistic attributes. These stylistic categories were developed by Caso, Bernal, and Acosta and called the CBA classification here. For a given pottery category, when the piece is broken it can produce smaller pieces of different categories, each of which is related to the original CBA. That is, for a given style there can be rim pieces, base pieces, handles, etc. The ceramics in Table 15-1 are the ones that indicate Period Ia. The CBA class is given in column 1, the minimum number of pieces needed to represent the presence of a particular stylistic type is given in column 2. Column 3 gives the categories associated with pieces of that pottery type. The fourth column includes categories whose pieces may overlap with or resemble those of another CBA category.

The following ceramic categories for the rest of the time periods considered here are given in Table 15-2. There are 566 sites with at least one piece of ceramic from the diagnostic categories that indicate the Ia Period. 390 of those sites have two or more sherds, and 304 of them have 3 or more. However, if we only consider the terraces with three or more pieces, then we will exclude

Table 15-1. The Basic Occupational Phases of the Valley

Period	Approximate Date
Tierras Largas	1400 – 1150 BC
San Jose	1150 – 850 BC
Guadalupe	850 – 700 BC
Rosario	700 – 500 BC
Monte Albán Ia	500 – 300 BC
Monte Albán Ic	300 – 150/100 BC
Monte Albán II	150/100 BC – AD 200
Monte Albán IIIa	AD 200 – – AD 500
Monte Albán IIIb	AD 500 – 700/750
Monte Albán IV	AD 700/750 – AD 1000
Monte Albán	AD 1000 – 1521

almost half the terraces that can be considered occupied during the Ia Period. Therefore, we decided to classify any site with two or more Ia Period ceramic pieces as being occupied at that time and all the rest as unoccupied. The number of occupied terraces in each of the periods under study is given in Table 15-3.

The environmental variables used to predict the location of the occupied terraces in the next section are:

[Location (Column T1 in the dataset), North grid coordinate (Column T10-12), East grid coordinate (Column T13-15), elevation (Column T24-26), topography (Column T27), soil type (Column T28), soil depth (Column T29), silting (Column T30), presence of a spring (Column T32), barranca or wash adjacent (Column T33), type of vegetation (Column T34), vegetation abundance (Column T35), special resources (Column T36), distance from road (Column T72), and the Ia Terrace occupation classification (sum of indicator ceramic greater than one).]

3. Generating the Building Blocks: Data Mining and Rule Extraction

In order to successfully apply Genetic Programming techniques it was important to identify the building blocks from which the programs will be composed. Five different data mining techniques for rule extraction were applied to the

Table 15-2. Ceramic categories that are used to indicate a Period 1a occupation. Ceramic categories in italics are not included in the databases and underlined categories are listed multiple times but counted only once

CBA designation	Count Special Requirements	Included Categories	Excluded Categories
C-2	Only Count if There are 4 or More Pieces	*0008*, 0022, 0031, 0032, 0038, *0056*, 0121, 0122, 0123, *0381, 0382*, 0383, 0384, 0385, 0386, 0561	None
C-4	None	0016, 0018, 0387, 0389, 0390, 0391, 0393, 0394, 0395, *0396*	None
K-3	Only Count if There are 2 or More Pieces	2010, 2042, *2064*, 2065, 2072, 2076, *2077*, 2080, *2411*	None
K-8	None	2079	2052, 2078, 2085
G-15	None	*1319, 1333, 1336, 1337*, 1342, 1343, 1345, 1346, 1347, 1348, 1357, *1358*, 1361, 1362, 1363, 1364, 1365, 1367, 1369, 1370, *1373*	None
G-16	None	1332, 1339, 1340, 1342, 1343, 1344, 1347, *1357, 1358*, 1364, 1365, 1366, 1368, 1369, 1370, *1373*	None
G-17	None	*1331*, 1332, *1334*, 1340, 1366, *1372*	1338

Table 15-3. Ceramic indicators for the remaining periods. Ceramic categories in italics are not present in the database

Time Period	Indicator Ceramic Categories
Ic	1297, 1338, 1353, 1355
II	0001, 0002, 0003, 0004, 0005, 0006, 0021, *0023*, 0407, 1194, 1419, 1420, *2061, 2416, 2417*, 3408, *3409*
IIIa	1264, 1265, 1312, 1421, 3410, 3411
IIIb-IV	1120, 1122, 1123, 1125, 1126, 1137, 1138, 1140, 1259, 1263, 1274, 1277, 1422, 2418
V	1102, 1104, 1105, 1106, 1107, 1109, 5007, 5329

Table 15-4. The number of occupied terraces in each phase

Time Period	Ia	Ic	II	IIIa	IIIb-IV	V
Number of Occupied Sites	390	220	260	149	1166	343

data set. It was not clear ahead of time what approach will be most effective in the extraction of rules for each period, and whether one approach can be effectively used across all occupational periods.

The WEKA data mining toolkit was the source for the techniques used here (Witten and Frank, 2005). The following data mining techniques were used; the One Rule Hypothesis, Naïve Bayes, Alternating Decision Trees, J48 Decision Trees, and the Naïve Bayes Decision Tree techniques. It is important to note that the database being used is a collection of terraces that were populated over a period of about 2000 years. Very few of the sites were inhabited throughout all six of the time periods that make up those 2000 years. In fact, during most of the time periods, less than 1/5 of the sites are inhabited (according to the current Period identifications). The IIIb-IV Period is the only exception, with a little over half of the sites occupied. It is also important to note that only residential sites were included in the database.

This means that, for a given time period, a site not being occupied does not mean that it is a site that would not have been occupied. It may just mean that the terrace is slightly less desirable than the ones already occupied. So we decided to use a training set that included the instances representing the Ia Period terraces and an equivalent number of randomly chosen non-Ia Period terraces from the remainder of the database. We then ran each of the machine learning techniques with the WEKA default parameters on this training set and tested the resultant rules' accuracies on all 2073 instances. Table 3 gives the results.

All of the techniques did quite well at predicting the location of occupied Phase 1a terraces. They were less successful in predicting unoccupied terraces. Since it is our assumption for this study that the factor that limited the number occupied terraces in the area during the Ia Period is the lack of people and not the lack of desirable terraces, then the percent of correctly identified occupied Ia Period terraces is more important than the percent of correctly identified unoccupied Ia Period terraces. That being said, we still consider the correct identification of unoccupied terraces to be important and the J48 decision tree did this better then the others. Since all of the techniques performed at a high enough level to be considered successful at identifying occupied terraces, the J48 decision tree technique was selected on that basis to be the best at identifying which terraces were desirable and which were undesirable during the Ia Period.

We are looking at which sites were occupied in an effort to try to identify which sites were desirable enough to occupy. It is safe to assume that any

Table 15-5. Comparison of 1a classification results

Ia Terrace Classification — 402 'no' instances and 390 'yes' Instances (792 total) as Training Set, results are after a test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
One Rule	-B 6	70.6%	65.3%	93.3%
Naïve Bayes		70.8%	64.9%	96.2%
AD Tree	-B 10 -E -3	76.5%	71.3%	98.7%
J48 Tree	-C 0.25 -M 2	81.70%	78.30%	96.20%
NB Tree		78.40%	74.10%	96.90%

site that was occupied was desirable, but it is not safe to assume that any site not occupied was undesirable. By looking at some of the instances we can see that some unoccupied sites have very similar attributes to occupied sites. Examples of this can be seen in Table 15-5. Instances 20 and 1234 have the same attributes and are located very near to each other, yet one is occupied and the other is not. The same can be seen with instances 272 and 260. It might be said that the sites are too close together and that when the one became occupied, the other become undesirable because of its proximity to an occupied site. However, there are sites that are closer together than these and are still both occupied. This can mean that there are additional social and economic factors that influence terrace occupation, or that there simply weren't enough people in the area to occupy all the desirable terraces. We shall assume for the purposes of this paper that these sites would have been occupied had there been more people. Therefore, classifying more terraces as desirable than are occupied in a period is not indicative of erroneous results. However, in future work we will investigate the use of GP as a tool to integrate rules relating to social and economic context in with the terrace specific rules generated here.

The same set of rule extraction techniques were applied to each of the other five time periods with the results shown in Tables 15-6–15-10. The results preceded with a '*' are those that best classify which sites are desirable and which are not in a given period. The best was chosen by taking the one that had the highest total percent correct among those that were within 6% of the best 'yes' identifier. Those enclosed in square brackets are the second best. It should be noted that the rules produced by the J48 machine learning technique performed the best for five of the six time periods and was a close second for

Table 15-6. Example instances from Period 1a

Attributes	**Instance 20**	**Instance 1234**	**Instance 272**	**Instance 260**
Location	Monte Albán	Monte Albán	Monte Albán	Monte Albán
North Grid Coordinate	189	183	175	171
East Grid Coordinate	340	339	311	308
Elevation	375	375	400	400
Topography	Sloped	Sloped	Near Flat	Near Flat
Soil Type	1	1	1	1
Soil Depth	0	0	0	0
Silting	None	None	None	None
Presence of a Spring	Absent	Absent	Absent	Absent
Barranca or Wash Adjacent	Absent	Absent	Absent	Absent
Type of Vegetation	Grass and Brush	Grass and Brush	Grass and Brush	Grass and Brush
Vegetation Abundance	Moderate	Moderate	Moderate	Moderate
Special Resources	Quarryable Stone	Quarryable Stone	None	None
Distance from Road	Close	Close	Close	Close
Occupied in Ia Period	No	Yes	No	Yes

the other time period. Therefore, we will use the rules produced by the J48 method as the basic rules used by the GP system.

The J48 method produces C4.5 trees for each of the periods. The tree for period 1a is given in Figure 15-3. The key variables here are elevation and location since they are both found near the top of the tree. Rules are generated from the tree by following a path from the root to each leaf node with a non-zero number of occupied terraces associated with it.

We then extract rules from each of the decision trees by following paths from the root node to leaf nodes that possess certain qualities. The leaf nodes need to contain a minimum number of terraces and need to be clearly representative of one class or another. In other words, it should contain over 10 objects, the majority of which are occupied terraces. Hundreds of such rules are generated over a number of different dimensions including environmental variables, residential occupation type, and craft activities.

Each rule has a spatial expression as shown in Figure 15-4. In the figure, a dark grey filled dot means desirable and occupied, whereas a light grey filled dot represents desirable and unoccupied, and a white filled dot with grey boundaries

Table 15-7. Terrace classifications for the remaining phases

Ic Terrace Classification - 222 'no' instances and 220 'yes' Instances (442 total) as Training Set, results are after test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
One Rule	-B 6	68.1%	65.8%	86.8%
Naïve Bayes		69.3%	66.8%	90.0%
AD Tree	-B 10 -E -3	81.4%	81.7%	78.6%
*J48 Tree	*-C 0.25 -M 2	*79.70%	*78.80%	*87.30%
[NB Tree]		[77.20%]	[75.90%]	[87.70%]

Table 15-8. Terrace classifications for the remaining phases

II Terrace Classification - 242 'no' instances and 260 'yes' Instances (502 total) as Training Set, results are after test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
One Rule	-B 6	86.8%	88.9%	72.3%
Naïve Bayes		79.9%	78.3%	91.2%
AD Tree	-B 10 -E -3	82.0%	81.3%	86.9%
*J48 Tree	*-C 0.25 -M 2	*87.00%	*87.10%	*85.80%
[NB Tree]		[82.40%]	[81.40%]	[89.60%]

represents undesirable according to the rule. The figure shows the number of sites predicted to be occupied by rule 5 for Period II. This rule focuses on sites adjacent to ancient roads and its textual description is given below:

> **"If terrace elevation <= to 350, and not occupied in the previous phase (1c), and East grid coordinate is between 337 and 246, and directly adjacent to a road, then occupied = "yes", otherwise occupied = "no" "**

Notice that although the spatial expression of a rule expresses an understandable pattern it is hard to see that from its original structure. In the figure one can

Table 15-9. Terrace classifications for the remaining phases

IIIa Terrace Classification - 157 'no' instances and 149 'yes' Instances (306 total) as the training Set, results are after a test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
One Rule	-B 6	84.1%	85.2%	70.5%
Naïve Bayes		77.0%	76.4%	84.6%
AD Tree	-B 10 -E -3	75.1%	74.0%	89.9%
[J48 Tree]	[-C 0.25 -M 2]	[78.10%]	[77.30%]	[87.90%]
*NB Tree		*80.00%	*79.70%	*83.90%

Table 15-10. Terrace classifications for the remaining phases

IIIb-IV Terrace Classification - 907 'no' instances and 1166 'yes' Instances (2073 total) as the training Set, results are after a test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
[One Rule]	[-B 6]	[71.2%]	[55.0%]	[83.7%]
Naïve Bayes		68.7%	69.0%	68.5%
AD Tree	-B 10 -E -3	71.5%	72.7%	70.6%
*J48 Tree	*-C 0.25 -M 2	*82.00%	*73.30%	*88.70-%*
NB Tree		75.40%	70.70%	79.20%

see a clearly etched pattern of sites that follow the trails of the major ancient roads through the site. Our goal in the next section is to develop a GP system that is able to synthesize a generalized rule that produces the same spatial affects as a group of rules but improves the understandability by using more semantically more meaningful variables.

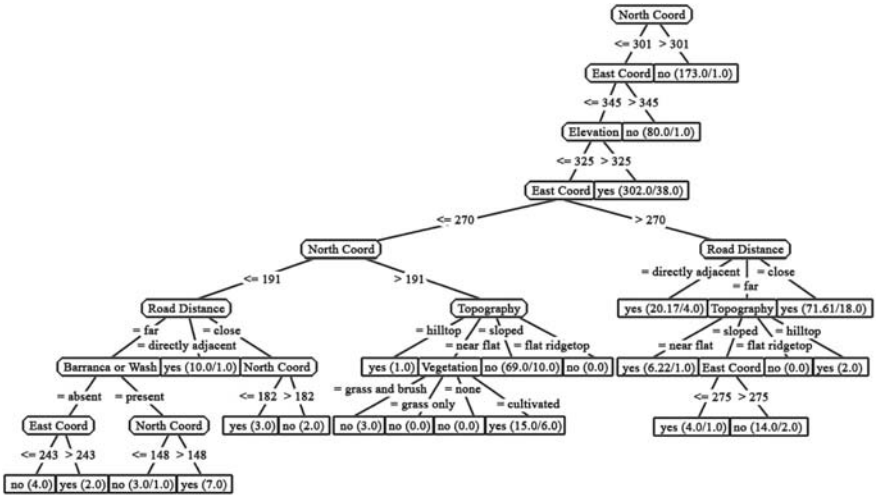


Figure 15-3. The decision tree for Period 1a

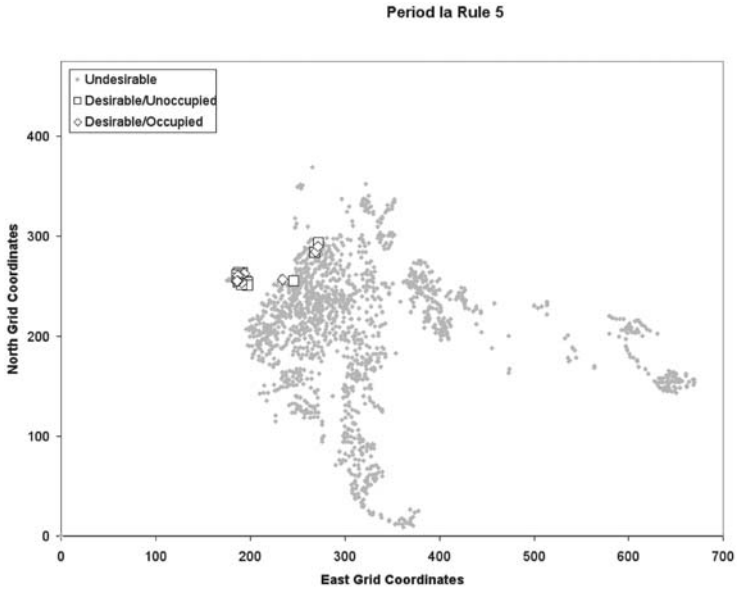


Figure 15-4. The spatial expression of an extracted rule for Period II of the Monte Albán occupation

Table 15-11. Terrace classifications for the remaining phases

V Terrace Classification - 337 'no' instances and 343 'yes' Instances (680 total) as the training Set, results are after a test on all 2073 instances				
Classifier	**Parameters**	**Total Percent Correct**	**'no' Class Percent Correct**	**'yes' Class Percent Correct**
One Rule	-B 6	66.7%	62.6%	87.2%
Naïve Bayes		60.8%	54.3%	93.6%
[AD Tree]	[-B 10 -E -3]	[71.8%]	[68.4%]	[88.6%]
*J48 Tree	*-C 0.25 -M 2	*73.30%	*69.30%	*93.30%
NB Tree		72.50%	70.20%	84.00%

4. Using CA and GP to generate neighborhood descriptions

Now we proceed to use the rules extracted above as the basic ingredients for the GP process. Figure 15-5 gives the Cultural Algorithm framework within which the GP process will be embedded here (Cowan and Reynolds, 2003). The system has two components. In the black box component we employ a Genetic Algorithm population space. Each gene on a chromosome corresponds to a variable in one of the rules in our base of extracted rule and its value is a "1" if it is to be used and it is a "0" otherwise. Also included are variables not used in the extracted rules but suggested to be potentially meaningful by the expert users. Each chromosome is tested using an agent based simulation system that simulates the colonization of the site using the subset of selected rules. The goal is to identify set of variables that predicts the colonization of the city as good as or better than the original set but is more understandable. Next, we use these variables as the building blocks for GP to produce a new semantically meaningful rule.

The best variables in each generation are "accepted" into the belief space and used to identify "schema" of variables that are associated with good performers. Those schemata can then be injected back into the population space by the "influence function" to guide the evolution process. Here if the injected subsets are already present in a chromosome they are less likely to be modified by mutation and crossover.

After performance has stabilized at a desired level the evolved variable sets are given to the second or white box stage. The population space in this stage is a Genetic Program System. The selected variables used in the rules and the related operators are added to the GP language to be used to evolve the new rules or programs there. The goal is to evolve a GP program that is a generalization of the originally extracted rules. Instead of having a collection of independent and detailed rules like rule 5 shown earlier, a hierarchically structured generalization will be produced the interprets the collective spatial pattern in a more understandable way.

For example, the one rule extracted for the predicting of lithic workshop locations in period Ic is the following:

If east Grid Coordinate location =< 211 then occupied= 'yes', otherwise occupied='no'.

This rule predicts 7 of the 10 workshops as yes and 3 as no. When this rule and the rest of the rule set is given to the GP system, the following is one of the new rules produced:

If East Grid Coordinate location =< 211 and quarryablestone = 'yes' or East Grid Coordinate location >300 and quarryablestone = 'yes' then occupied otherwise 'not occupied'.

This new rule now predicts all ten of the known sites as occupied. In addition, the rule provides information about the location of this craft activity. The rule suggests that while location near quarryable stone is important, the west side of the site is more attractive in this regard. However, there is activity on both sides of the central plaza which is the reason for the coordinate gap. It turns out that this reflects the presence of a limestone deposit that runs along the western perimeter of the hilltop, along with more limited deposits on the east side of the central plaza.

The resulting GP program will then generate a collective spatial expression such as that shown in Figure 15-6. This figure gives the predicted occupation for the earliest phase of the site, Ia. In the figure each dark plotted dot represents a terrace that was predicted as occupied by the subset of selected variables combined into a new rule. Notice that the dark plotted terraces are found in the area surrounding the plaza area, which was pictured in Figure 15-1. There is also a strong tendency for the occupied terraces to be situated along the major north south route within the site. This pattern reflects the extent to which the road network supported the earliest colonization there. The light plotted dots represent terraces that are predicted to be unoccupied in that period according to the rules. Thus, the original occupants of the site were very selective in the locations that they used for the first terraces relative to the total number of occupational units. The predictive power of the GP program can be determined by comparing the accuracy of its expression to the accuracy of the original rule set that was used to generate the original solution.

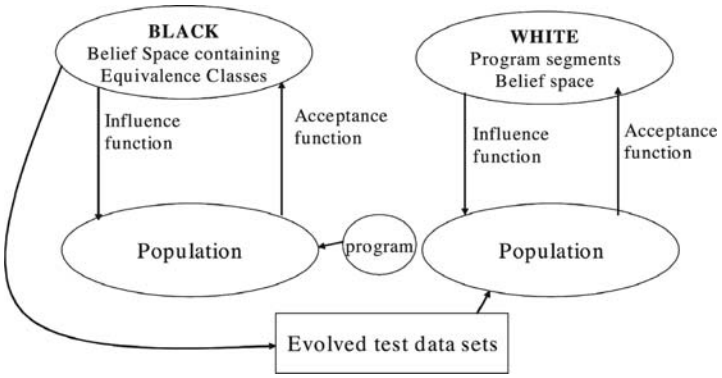


Figure 15-5. The chained GP CA configuration used here. The Black box GPCA extracts the variables that are used to describe a configuration of occupied terrace for a given period. The subset of extracted variables is given to the White box GPCA component that attempts to synthesize an equivalent plan or rule.

5. Conclusion

In this paper we proposed an approach to extract settlement plans from archaeological data in order to attempt to create the emergence of early urban settlement. The extracted rules for different occupational activities exhibited reasonable predictive power but displayed some problems. First, in some cases

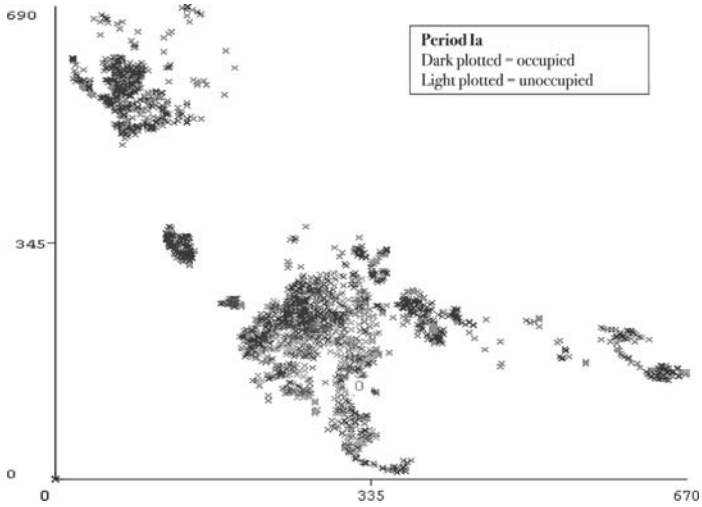


Figure 15-6. Simulated occupation of Monte Albán in Phase 1a based on the extracted rules.

the variables used were not semantically useful. Second, some rules predicted that there were more desirable terraces than occupied ones. This suggests that there were social and economic factors that influenced the occupation of a terrace in addition to its specific environmental characteristics.

In this paper, we addressed the first issue by using Genetic Programming and Cultural Algorithms to first select more meaningful variables that are similar to the original ones in terms of their predictions. Theses selected variables were then given to a GP system as building blocks that were used to produce new combined rules. The resultant rules were often more semantically meaningful, and in some cases more predictive than the original.

In the future, we will add rules that relate to residential occupation and economic activity in the terraces to identify the importance that locational context plays in terrace occupation and functionality. In this situation, GPCA will be used to integrate these different rule sets, one for occupation, residence, and functionality respectively.

References

Blanton, R. (1978). Monte albán: Settlement patterns at the ancient zapotec capital. ACM Press.

Blanton, R., Kowalewski, S., Feinman, G., and Appel, J. (1982). *The Prehispanic Settlement Patterns of the Central and Southern Parts of the Valley of Oaxaca, Mexico.* Monte Albán's Hinterland, Part I. The Regents of the University of Michigan, The Museum of Anthropology.

Cowan, G.S. and Reynolds, R. (2003). *Acquisition of Software Engineering Knowledge*, volume 14. World Scientific Publishing.

Ostrowski, David A. and Reynolds, Robert G. (2003). Using software engineering knowledge to drive genetic program design using cultural algorithms exploiting the synergy of software engineering knowledge in evolutionary design. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice*, chapter 5, pages 63–80. Kluwer.

Witten, I. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, Amsterdam.

Index