Genetic and Evolutionary Computation

Rick Riolo
Jason H. Moore
Mark Kotanchek *Editors*

Genetic Programming Theory and Practice XI

Springer

Genetic and Evolutionary Computation

Series Editors:

David E. Goldberg
John R. Koza

For further volumes:
http://www.springer.com/series/7373

Rick Riolo • Jason H. Moore • Mark Kotanchek
Editors

Genetic Programming Theory and Practice XI

Foreword by Arthur K. Kordon

Springer

Editors
Rick Riolo
Center for the Study of Complex Systems
University of Michigan
Ann Arbor, MI, USA

Mark Kotanchek
Evolved Analytics
Midland, MI, USA

Jason H. Moore
Institute for Quantitative
Biomedical Sciences
Dartmouth Medical School
Lebanon, NH, USA

ISSN 1932-0167
ISBN 978-1-4939-0374-0 ISBN 978-1-4939-0375-7 (eBook)
DOI 10.1007/978-1-4939-0375-7
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2014934666

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

> Theory is knowledge that doesn't work. Practice is when everything works and you don't know why.
>
> Hermann Hesse

> In our lab theory and practice are combined: nothing works and nobody knows why.
>
> Laboratory Joke #1

Building strategic trustable relationships between theoretical gurus and practitioners is one of the key challenges in any emerging scientific field. For more than a decade, the Genetic Programming Theory and Practice (GPTP) workshop organized annually by the Center for the Study of Complex Systems at the University of Michigan, Ann Arbor is one of the best examples of how theory and practice in the growing field of Genetic Programming (GP) are effectively integrated. The 11th GPTP workshop, held on May 9–11, 2013, has shown a new level of successful collaboration, and the key results are presented in this book.

As one of the veterans of this event, attending eight workshops, including the first one in 2002, I try to answer the question: 'What brought me here?' The key attraction to me is the unique creative spirit of GPTP. It is based on several factors. The first factor is the vision that the tighter the link between theory and practice, the more successful the growth of the field. In theory, any research area has this objective, but GPTP has succeeded in putting it into practice. The second factor is the format of the workshop which leads to an effective dialogue between academics and practitioners. The secret is in an almost optimal time balance between presentations, discussions, and networking. The most exciting are the discussions, sometimes emotional and heated but always constructive and open. The third factor is the right "population size" of 35–40 participants of leading figures from both theoretical and practical sides of the field. This gives opportunities for building an effective network during the event, which is difficult to accomplish in a big crowded conference. The fourth factor is the highly respected communication path of the results from the workshop by the GPTP book series published by Springer. Each invited presentation is a book chapter and the annual

volume represents a condensed snapshot of the key theoretical and practical state of the art of GP.

The main accomplishment of GPTP is the developed and, with the years, improved effective collaboration between the key academics and practitioners. Gradually, a GPTP community has formed as a recognized driver of the theoretical and practical issues in GP. Both sides benefit from the dialogue. The theoreticians have a direct feedback from the practitioners on the potential for value creation from their academic ideas and information about the current demand of real-world problems to be solved. The practitioners have a unique chance to be familiar with the latest research ideas and to discuss them with the leading theoretical gurus in GP. This fertilized ground for the exchange of ideas contributes to building bonds and growing trust and credibility. The specific results are documented in all the GPTP volumes, many published papers, practical applications, and developed software.

Even a success story, such as that of GPTP, has room for improvement. From a practitioner's point of view, I would suggest to open the door to and focus on hybrid systems that include GP as a component. The current shift from manufacturing and engineering applications toward business modeling requires solutions based on multiple approaches. Integrate and conquer is the winning strategy for establishing GP in the top application areas in advanced analytics and big data. Fresh blood of academics and practitioners with a synergetic mindset is needed.

Freeport, USA Arthur K. Kordon
August 2013

Preface

The work described in this book was first presented at the Eleventh Workshop on Genetic Programming, Theory and Practice, organized by the Center for the Study of Complex Systems at the University of Michigan, Ann Arbor, held on May 9–11, 2013. 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/gptp-workshops/.

We thank all the workshop participants for making the workshop an exciting and productive 3 days. In particular we thank the authors, without whose hard work and creative talents neither the workshop nor the book would be possible. We also thank our keynote speakers Dr. Charles Ofria, associate professor at Michigan State University in the Department of Computer Science and the Ecology, Evolutionary Biology, and Behavior Program, director of the MSU Digital Evolution Laboratory and the deputy director of the BEACON Center for the Study of Evolution in Action, and Prof. Michael Affenzeller, professor at University of Applied Sciences, Hagenberg Campus, and head of the Josef Ressel Center Heureka! at Hagenberg.

The workshop received support from these sources:

- The Center for the Study of Complex Systems (CSCS);
- John Koza, Third Millennium Venture Capital Limited;
- Michael and Gilda Korns;
- Mark Kotanchek, Evolved Analytics;
- Jason Moore, Computational Genetics Laboratory at Dartmouth College;
- Babak Hodjat and Genetic Finance LLC;

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 Susan Carpenter, who makes GPTP workshops run smoothly with her diligent efforts before, during, as well as after the workshop. After the workshop, many people provided invaluable assistance in producing this book. Special thanks go to Kadie Sanford, who did a wonderful job working with the authors, editors and publishers to get the book completed very quickly. Courtney Clark and Melissa Fearon provided invaluable editorial efforts, from the initial plans for the book through its final publication. Thanks also to Springer for helping with various technical publishing issues.

Ann Arbor, USA — Rick Riolo
Lebanon, USA — Jason H. Moore
Midland, USA — Mark Kotanchek
August 2013

Contents

Contributors

Michael Affenzeller is leader of the Heuristic and Evolutionary Algorithms Laboratory (HEAL) and Professor for Heuristic Optimization and Machine Learning at the University of Applied Sciences Upper Austria, Hagenberg (michael.affenzeller@fh-hagenberg.at).

Nicholas A. Allgaier is a PhD candidate in Mathematics and Statistics at the University of Vermont, USA (Nicholas.Allgaier@uvm.edu).

Joshua C. Bongard is an Associate Professor of Computer Science at the University of Vermont, USA (Josh.Bongard@uvm.edu).

Jan Broeckhove is a Professor in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include computational science and distributed computing. He received his PhD in Physics in 1982 at the Free University of Brussels (VUB), Belgium (jan.broeckhove@uantwerpen.be).

Bogdan Burlacu is a PhD student working in the Heuristic and Evolutionary Algorithms Laboratory at the University of Applied Sciences Upper Austria, Hagenberg (bogdan.burlacu@fh-hagenberg.at).

Mauro Castelli is an Invited Professor at the Instituto Superior de Estatística e Gestão de Informação (ISEGI), Universidade Nova de Lisboa, Portugal (mcastelli@isegi.unl.pt).

Christopher M. Danforth is an Associate Professor of Mathematics and Statistics at the University of Vermont, USA (Chris.Danforth@uvm.edu).

Scott M. Dudek is a software developer in the Center for Systems Genomics at the Pennsylvania State University, USA (sud23@psu.edu).

Barry Fitzgerald is a PhD student at the Limerick Institute of Technology and is lead engineer at the startup company NVMdurance.

Hugh P. Garavan is an Associate Professor of Psychiatry at the University of Vermont, USA (Hugh.Garavan@uvm.edu).

Thomas Helmuth is a graduate student in the Computer Science Department at the University of Massachusetts, Amherst, MA, USA (thelmuth@cs.umass.edu).

Erik Hemberg is a Postdoctoral Researcher in the ALFA group at the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, USA (hembergerik@csail.mit.edu).

Douglas P. Hill is a software engineer at the Institute for Quantitative Biomedical Sciences at Dartmouth Medical School, USA (douglas.hill@Dartmouth.edu).

Babak Hodjat is cofounder and Chief Scientist at Genetic Finance LLC in San Francisco, California, USA. He received his PhD in Machine Intelligence from Kyushu University in Japan (babak@geneticfinance.net).

Emily R. Holzinger is a graduate student in the Human Genetics Program at Vanderbilt University, USA (emily.r.holzinger@vanderbilt.edu).

Ilknur Icke is a Postdoctoral Research Associate in Computer Science at the University of Vermont, USA (Ilknur.Icke@uvm.edu).

Michael Kommenda is member of the Heuristic and Evolutionary Algorithms Laboratory at the University of Applied Sciences Upper Austria, Hagenberg (michael.kommenda@fh-hagenberg.at).

Arthur K. Kordon is Advanced Analytics Leader in the Advanced Analytics Group within the Dow Business Services of the Dow Chemical Company (akordon@dow.com).

Michael F. Korns is Chief Technology Officer at Freeman Investment Management, Henderson, Nevada, USA (mkorns@korns.com).

Mark Kotanchek is a CEO and Founder of Evolved Analytics LLC (mark@evolved-analytics.com).

Gabriel Kronberger is Professor of Data Engineering and Business Intelligence at the University of Applied Sciences Upper Austria, Hagenberg (gabriel.kronberger@fh-hagenberg.at).

Ruowang Li is a graduate student in the Bioinformatics and Genomics Program at the Pennsylvania State University, USA (rvl5032@psu.edu).

Luca Manzoni is a Postdoctoral Researcher at Laboratoire i3S of the Université Nice Sophia Antipolis, France (luca.manzoni@i3s.unice.fr).

Jason H. Moore is the Third Century Professor of Genetics and Director of the Institute for Quantitative Biomedical Sciences at Dartmouth Medical School, USA (Jason.H.Moore@Dartmouth.edu).

Una-May O'Reilly is leader of the Evolutionary Design and Optimization Group and Principal Research Scientist at the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, USA (unamay@csail.mit.edu).

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

Marylyn D. Ritchie is an Associate Professor of Biochemistry and Molecular Biology at the Pennsylvania State University, USA (marylyn.ritchie@psu.edu).

Conor Ryan is an Associate Professor of Machine Learning at the University of Limerick, Ireland, where he is the Director of the Biocomputing Developmental Systems Group.

Andrew Saykin is Raymond C. Beeler Professor of Radiology and Imaging Sciences and Director of the Indiana University Center for Neuroimaging.

Hormoz Shahrzad is Principal Researcher and platform architect at Genetic Finance LLC in San Francisco, CA, USA (hormoz@geneticfinance.net).

Li Shen is Associate Professor of Radiology, Indiana University Center for Neuroimaging.

Sara Silva is Senior Researcher at the Knowledge Discovery and Bioinformatics (KDBIO) group of INESC-ID Lisboa, Portugal, and an Invited Researcher at the Evolutionary and Complex Systems (ECOS) group of CISUC, University of Coimbra, Portugal (sara@kdbio.inesc-id.pt).

Lee Spector is a Professor of Computer Science in the School of Cognitive Science at Hampshire College, Amherst, MA, USA (lspector@hampshire.edu).

Sean Stijven is a PhD student in the research group Computational Modeling and Programming (CoMP) at the University of Antwerp and in the Internet Based Communication Networks and Services (IBCN) research group at Ghent University. His research interests include modeling of complex systems, genetic programming, variable selection, symbolic regression, and surrogate modeling (sean.stijven@uantwerpen.be).

Joe Sullivan is a Lecturer in the Electrical and Electronic Engineering Department at the Limerick Institute of Technology, having worked in the semiconductor industry for almost two decades.

Philip Truscott studied Politics and Social Administration at the University of Edinburgh and has a PhD in Sociology from the University of Surrey. He has taught in universities in the USA, Vietnam, and the Philippines. He is currently a Senior Lecturer in Humanities, Arts, and Social Sciences at the Singapore University of Technology and Design.

Ruben Van den Bossche is a PhD student in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include scheduling mechanisms in grid and cloud computing (Ruben.VandenBossche@uantwerpen.be).

Kurt Vanmechelen is a Postdoctoral Fellow in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include resource management in grid and cloud environments in general and the adoption of market mechanisms in such systems in particular. In 2009 he received his PhD in Computer Science at the University of Antwerp (UA), Belgium (Kurt.Vanmechelen@ua.ac.be).

Leonardo Vanneschi is Assistant Professor at the ISEGI, Universidade Nova of Lisboa, Portugal; Researcher at the DISCo, University of Milano-Bicocca, Italy; and Invited Associate Researcher at the KDBIO group of INESC-ID Lisboa, Portugal (lvanneschi@isegi.unl.pt).

Ekaterina (Katya) Vladislavleva is a Chief Data Scientist and Partner at Evolved Analytics LLC, USA, and Managing Director at Evolved Analytics Europe BVBA, Belgium (katya@evolved-analytics.com).

Stefan Wagner is head architect of HeuristicLab and Professor of Complex Software Systems at the University of Applied Sciences Upper Austria, Hagenberg (stefan.wagner@fh-hagenberg.at).

Robert A. Whelan is a Postdoctoral Research Associate in Psychiatry at the University of Vermont, USA (Robert.Whelan@uvm.edu).

Stephan Winkler is member of the Heuristic and Evolutionary Algorithms Laboratory and head of the Bioinformatics Research Group at the University of Applied Sciences Upper Austria, Hagenberg (stephan.winkler@fh-hagenberg.at).

Chapter 1
Extreme Accuracy in Symbolic Regression

Michael F. Korns

Abstract Although recent advances in symbolic regression (SR) have promoted the field into the early stages of commercial exploitation, the poor accuracy of SR is still plaguing even the most advanced commercial packages today. Users expect to have the correct formula returned, especially in cases with zero noise and only one basis function with minimally complex grammar depth. Poor accuracy is a hindrance to greater academic and industrial acceptance of SR tools.

In a previous paper, the poor accuracy of Symbolic Regression was explored, and several classes of test formulas, which prove intractable for SR, were examined. An understanding of why these test problems prove intractable was developed. In another paper a baseline Symbolic Regression algorithm was developed with specific techniques for optimizing embedded real numbers constants. These previous steps have placed us in a position to make an attempt at vanquishing the SR accuracy problem.

In this chapter we develop a complex algorithm for modern symbolic regression which is extremely accurate for a large class of Symbolic Regression problems. The class of problems, on which SR is extremely accurate, is described in detail. A definition of extreme accuracy is provided, and an *informal argument* of extreme SR accuracy is outlined in this chapter. Given the critical importance of accuracy in SR, it is our suspicion that in the future all commercial Symbolic Regression packages will use this algorithm or a substitute for this algorithm.

Keywords Abstract expression grammars • Grammar template genetic programming • Genetic algorithms • Particle swarm • Symbolic regression

M.F. Korns (✉)
Analytic Research Foundation, 98 Perea Street, Makati 1229, Manila, Philippines
e-mail: mkorns@korns.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_1,

1 Introduction

The discipline of Symbolic Regression (SR) has matured significantly in the last few years. There is at least one commercial package on the market for several years (http://www.rmltech.com/). There is now at least one well documented commercial symbolic regression package available for Mathematica (http://www.evolved-analytics.com/). There is at least one very well done open source symbolic regression package available for free download (http://ccsl.mae.cornell.edu/eureqa). In addition to our own ARC system (Korns 2010), currently used internally for massive (million row) financial data nonlinear regressions, there are a number of other mature symbolic regression packages currently used in industry including Smits and Kotanchek (2004) and Kotanchek et al. (2007). Plus there is another commercially internally deployed regression package which handles 50–10,000 input features using specialized linear learning (McConaghy 2011).

Yet, despite the increasing sophistication of commercial SR packages, there have been serious issues with SR accuracy even on simple problems (Korns 2011). Clearly the perception of SR as a *must use* tool for important problems or as an *interesting heurism* for shedding light on some problems, will be greatly affected by the demonstrable accuracy of available SR algorithms and tools. The depth and breadth of SR adoption in industry and academia will be greatest if a very high level of accuracy can be demonstrated for SR algorithms.

In Korns (2012) we developed a simple, easy to implement, public domain baseline algorithm for modern symbolic regression which is reasonably competitive with current commercial SR packages. This algorithm was meant to be a baseline for further public domain research on provable SR algorithm accuracy. It is called Constant Swarm with Operator Weighted Pruning and is inspired by recent published techniques in pareto front optimization (Kotanchek et al. 2007), age layered population structures (Hornby 2006), age fitness pareto optimization (Schmidt and Lipson 2010), and specialized embedded abstract constant optimization (Korns 2010).

In this chapter we enhance the previous baseline with a complex multi-island algorithm for modern symbolic regression which is extremely accurate for a large class of Symbolic Regression problems. The class of problems, on which SR is extremely accurate, is described in detail. A definition of extreme accuracy is provided, and an *informal argument* of extreme SR accuracy is outlined in this chapter.

Before continuing with the details of our extreme accuracy algorithm, we proceed with a basic introduction to general nonlinear regression. Nonlinear regression is the mathematical problem which Symbolic Regression aspires to solve. The canonical generalization of nonlinear regression is the class of Generalized Linear Models (GLMs) as described in Nelder and Wedderburn (1972). A GLM is a linear combination of $\mathbf{I}$ basis functions B_i; i = 0,1, ...I, a dependent variable y, and an independent data point with M features $x = <x_0, x_1, x_2, \ldots, x_{M-1}>$: such that

- (*E1*) $y = \gamma(x) = c_0 + \Sigma c_i B_i(x) + \mathbf{err}$

As a broad generalization, GLMs can represent any possible nonlinear formula. However the format of the GLM makes it amenable to existing linear regression theory and tools since the GLM model is linear on each of the basis functions B_i. For a given vector of dependent variables, Y, and a vector of independent data points, X, symbolic regression will search for a set of basis functions and coefficients which minimize **err**. In Koza (1992) the basis functions selected by symbolic regression will be formulas as in the following examples:

- (*E2*) $B_0 = x_3$
- (*E3*) $B_1 = x_1 + x_4$
- (*E4*) $B_2 = \text{sqrt}(x_2)/\tan(x_5/4.56)$
- (*E5*) $B_3 = \tanh(\cos(x_2 * .2) * \text{cube}(x_5 + \text{abs}(x_1)))$

If we are minimizing the normalized least squared error, NLSE (Korns 2012), once a suitable set of basis functions B have been selected, we can discover the proper set of coefficients C deterministically using standard univariate or multivariate regression. The value of the GLM model is that one can use standard regression techniques and theory. Viewing the problem in this fashion, we gain an important insight. Symbolic regression does not add anything to the standard techniques of regression. The value added by symbolic regression lies in its abilities as a search technique: how quickly and how accurately can SR find an optimal set of basis functions B. The immense size of the search space provides ample need for improved search techniques. In basic Koza-style tree-based Genetic Programming (Koza 1992) the genome and the individual are the same Lisp s-expression which is usually illustrated as a tree. Of course the tree-view of an s-expression is a visual aid, since a Lisp s-expression is normally a list which is a special Lisp data structure. Without altering or restricting basic tree-based GP in any way, we can view the individuals not as trees but instead as s-expressions such as this depth 2 binary tree s-exp: *(/ (+ x_2 3.45) (∗ x_0 x_2))*, or this depth 2 irregular tree s-exp: *(/ (+ x_4 3.45) 2.0)*.

In basic GP, applied to symbolic regression, the non-terminal nodes are all operators (implemented as Lisp function calls), and the terminal nodes are always either real number constants or features. The maximum depth of a GP individual is limited by the available computational resources; but, it is standard practice to limit the maximum depth of a GP individual to some manageable limit at the start of a symbolic regression run.

Given any selected maximum depth k, it is an easy process to construct a maximal binary tree s-expression U_k, which can be produced by the GP system without violating the selected maximum depth limit. As long as we are reminded that each f represents a function node while each t represents a terminal node, the construction algorithm is simple and recursive as follows.

- (U_0): t
- (U_1): (f t t)
- (U_2): (f (f t t) (f t t))
- (U_3): (f (f (f t t) (f t t)) (f (f t t) (f t t)))
- (U_k): (f U_{k-1} U_{k-1})

The basic GP symbolic regression system (Koza 1992) contains a set of functions F, and a set of terminals T. If we let t ∈ T, and f ∈ F ∪ ξ, where *ξ(a, b)= ξ(a)=a*, then any basis function produced by the basic GP system will be represented by at least one element of U_k. Adding the ξ function allows U_k to express all possible basis functions generated by the basic GP system *to a depth of k*. **Note to the reader**, the ξ function performs the job of a pass-through function. The ξ function allows a *fixed-maximal-depth* expression in U_k to express trees of varying depth, such as might be produced from a GP system. For instance, the varying depth GP expression $x_2+(x_3-x_5)=\xi(x_2, 0.0)+(x_3-x_5)=+(\xi(x_2, 0.0) -(x_3\ x_5))$ which is a *fixed-maximal-depth* expression in U_2.

In addition to the special pass through function ξ, in our system we also make additional slight alterations to improve coverage, reduce unwanted errors, and restrict results from wandering into the complex number range. All unary functions, such as *cos*, are extended to ignore any extra arguments so that, for all unary functions, *cos(a,b)=cos(a)*. The *sqroot* and *ln* functions are extended for negative arguments so that *sqroot(a)=sqroot(abs(a))* and *ln(a)=ln(abs(a))*.

Given this formalism of the search space, it is easy to compute the size of the search space, and it is easy to see that the search space is huge even for rather simple basis functions. For our use in this chapter the function set will be the following functions: F=(+ − ∗**/cos sin tan tanh sqroot square cube quart exp ln** ξ). The terminal set is the features $\mathbf{x}_0$ thru $\mathbf{x}_{M-1}$ and the real constant **c**, which we shall consider to be 2^{18} in size.

During the writing of Korns (2010, 2011, 2012), a high level regression search language was developed called **RQL**. RQL was inspired by the database search language SQL. Therefore RQL is analogous to SQL but not similar to SQL. The algorithm included in this paper is primarily presented in RQL. A very brief, but hopefully sufficient, description of RQL follows.

Regression Query Language **RQL** is a high level Symbolic Regression search language and consists of **one or more search clauses** which together make up a symbolic regression request. Each search clause represents an independent evolutionary island in which a separate symbolic regression search is performed.

- (*A1*) **search** goal **where** island(breeder,strategy,popsize,pool,serial)
  ...constraints...
  ...events...

It is assumed that the champions from each independent search island will be accumulated into a final list of champions from which the best champion will become the answer to the entire search process. The search **goal** specifies the area to be searched. For example, a common goal is *universal(3,1,t)* which searches all single (1) regression champions from all possible basis functions of depth (3) where the terminals are both (t) variables (*containing features*) or abstract constants (*containing real numbers*). The goal *universal(3,1,t)* is also known as $U_3(1)$ throughout this chapter.

Another search goal example might be $f_0(v_0, f_1(v_1, c_0))$ which searches for a function with two arguments where the second argument is also a function with two arguments, the second of which is a constant. The abstract function variables f_0 thru f_K are meant to contain one concrete function from the set $F \cup \xi$ unless otherwise constrained. The abstract feature variables v_0 thru v_J are meant to contain one concrete feature from the set x_0 thru x_{M-1} unless otherwise constrained. The abstract constant variables c_0 thru c_L are meant to contain one real number, of size 2^{cbit}, unless otherwise constrained. The *constraints*, located anywhere after the **where** keyword, are in the form of limitations on variable and function variable coverage such as $f_0(cos,sin,tan,tanh)$ or $v_0(x_0, x_3, x_{10})$ or $c_0(3.45)$.

The **island** keyword sets up the parameters of the evolutionary search island. We use only two **breeders**: *pareto* which implements a typical pareto front algorithm and also understands *onfinal* and *onscore* events, and *smart* which implements a focused elitist algorithm and also understands *onfinal* and *onscore* events. We use only one population operator **strategy** *standard* which implements typical elitist mutation and crossover operators, plus standard swarm operators for optimizing embedded constants, see the baseline algorithm (Korns 2012). The population size **popsize**, constant pool size **pool**, and number of serial iterations per generation **serial** vary with each search specification.

Three other *constraint* and *event* clauses may appear anywhere after the **where** keyword. These are the **isolate** *constraint* clause, and the **onscore** and **onfinal** *events*. Each of these will be explained, with brief descriptions and actual examples, as we detail specific regression search requests required for the extreme accuracy algorithm.

Incidentally any reasonable pareto front implementation, any reasonable elitist implementation, any reasonable standard set of population operators, and any reasonable set of swarm optimizers for embedded constants will work with this extreme accuracy algorithm. The key to implementing this extreme accuracy algorithm lies in the number of independent search island requests and exactly what is searched for in each independent island, which brings us to the core issues involved in the pursuit of extreme accuracy.

When searching for a single regression champion with one simple basis function of depth 3 i.e. *universal(3,1,t)* also known as $U_3(1)$, one encounters a number of difficult problems. Many of the simple forms covered in $U_3(1)$, such as $cos(x_{10})$, cannot be easily discovered by evolutionary methods. This is because getting close to the champion does not necessarily convey a fitness improvement. For instance where $\cos(x_{10})$ is the correct champion, it is not clear that $\cos(x_8)$ $\cos(x_9)$ $\cos(x_{11})$ provide any fitness improvement which evolutionary search methods might exploit. Another easily understood pair of examples can be shown where the correct champion is square($x_{10}+3.427$). Any trial champion such as cube($x_{10}+c_0$) will have its fitness improved as c_0 approaches 3.427. Unfortunately this convenient fitness improvement does not occur when the correct champion is $\cos(x_{10}+3.427)$ and the trial champion is cube($x_{10}+c_0$) or even $\cos(x_{10}+c_0)$.

So the obvious answer is to search *universal(3,1,t)* serially for every possible value of functions, variables, and embedded constants. This is fine when the number of functions and variables are small and when the number of bits (*cbit*) used to represent embedded constants is small. **However Symbolic Regression is of greatest value when the number of functions, features, and cbits is large.** In our work in this chapter we have the number of functions $\|F\| = 15$, the number of features $\|V\| = 100$, and *cbit*=18. The size of *universal(3,1,t)* can be computed with the following formula $\|F\|^7 * (\|V\| + 2^{cbit})^8$. Therefore $15^7 * (100 + 2^{18})^8. = 3.82E+51$ which is larger than the estimated number of stars in our universe.

Since serial search of *universal(3,1,t)* is not possible in reasonable time, pursuit of extreme accuracy requires us to move on to the more complex algorithm presented in this chapter. This extreme accuracy algorithm relies on three strategies. **First**, carving out the smaller subsets of *universal(3,1,t)* which can be shown to require serial search and demonstrating these areas are small enough to be serially searched in practical time. **Second**, carving out the larger subsets of *universal(3,1,t)* which are tractable for evolutionary search and demonstrating these larger areas are responsive to evolutionary search in practical time. **Third**, for those remaining areas too large for serial search and too unresponsive for evolutionary search, we use algebraic manipulations and mathematical regression equivalences to reduce these problems spaces to equivalent spaces which can be solved.

Our core assertion in this chapter is that the algorithm will find extremely accurate champions for all of the problems in $\mathbf{U}_2$**(1)** and in $\mathbf{U}_1$**(3)**.

Example Test Problems

In this section we list the example test problems which we will address. All of these test problems lie in the domain of either $\mathbf{U}_2$**(1)** or $\mathbf{U}_1$**(3)** where the function set F=**(+ − ∗/cos sin tan tanh sqroot square cube quart exp ln** ξ**)**, and the terminal set is the features $\mathbf{x}_0$ thru $\mathbf{x}_{M-1}$ plus the constant **c** with **cbit**=18. Our test will reference 100 features. Our core assertion is that the algorithm will find extremely accurate champions for all of these problems and for **all similar problems** in practical time.

- (*T1*): $y=1.57+(14.3*x_3)$
- (*T2*): $y=3.57+(24.33/x_3)$
- (*T3*): $y=1.687+(94.183*(x_3*x_2))$
- (*T4*): $y=21.37+(41.13*(x_3/x_2))$
- (*T5*): $y=-1.57+(2.3*((x_3*x_0)*x_2))$
- (*T6*): $y=9.00+(24.983*((x_3*x_0)*(x_2*x_4)))$
- (*T7*): $y=-71.57+(64.3*((x_3*x_0)/x_2))$
- (*T8*): $y=5.127+(21.3*((x_3*x_0)/(x_2*x_4)))$
- (*T9*): $y=11.57+(69.113*((x_3*x_0)/(x_2+x_4)))$
- (*T10*): $y=206.23+(14.2*((x_3*x_1)/(3.821-x_4)))$
- (*T11*): $y=0.23+(19.2*((x_3-83.519)/(93.821-x_4)))$

- (*T12*): y=0.283+(64.2∗((x$_3$−33.519)/(x$_0$−x$_4$)))
- (*T13*): y= −2.3+(1.13∗sin(x$_2$))
- (*T14*): y=206.23+(14.2∗(exp(cos(x$_4$))))
- (*T15*): y= −12.3+(2.13∗cos(x$_2$∗13.526))
- (*T16*): y= −12.3+(2.13∗tan(95.629/x$_2$))
- (*T17*): y= −28.3+(92.13∗tanh(x$_2$∗x$_4$))
- (*T18*): y= −222.13+(−0.13∗tanh(x$_2$/x$_4$))
- (*T19*): y= −2.3+(−6.13∗sin(x$_2$)∗x$_3$)
- (*T20*): y= −2.36+(28.413∗ln(x$_2$)/x$_3$)
- (*T21*): y=21.234+(30.13∗cos(x$_2$)∗tan(x$_4$))
- (*T22*): y= −2.3+(41.93∗cos(x$_2$)/tan(x$_4$))
- (*T23*): y=0.913+(62.13∗ln(x$_2$)/square(x$_4$))
- (*T24*): y=13.3+(80.23∗x$_2$)+(1.13∗x$_3$)
- (*T25*): y=18.163+(95.173/x$_2$)+(1.13/x$_3$)
- (*T26*): y=22.3+(62.13∗x$_2$)+(9.23∗sin(x$_3$))
- (*T27*): y=93.43+(71.13∗tanh(x$_3$))+(41.13*sin(x$_3$))
- (*T28*): y=36.1+(3.13∗x$_2$)+(1.13∗x$_3$)+(2.19*x$_0$)
- (*T29*): y=17.9+(2.13∗x$_2$)+(1.99∗sin(x$_3$))+(1.13*cos(x$_3$))
- (*T30*): y= −52.183+(9.13∗tanh(x$_3$))+(−11.13*sin(x$_3$))+(14.3*ln(x$_3$))

For the sample test problems, we will use only statistical best practices out-of-sample testing methodology. A matrix of independent variables will be filled with random numbers between −100 and +100. Then the model will be applied to produce the dependent variable. These steps will create the training data (each matrix row is a *training example* and each matrix column is a *feature*). A symbolic regression will be run on the training data to produce a champion estimator. Next a matrix of independent variables will be filled with random numbers between −100 and +100. Then the model will be applied to produce the dependent variable. These steps will create the testing data. The fitness score is the root mean squared error divided by the standard deviation of Y, NLSE. The estimator will be evaluated against the testing data producing the final NLSE and R-Square scores for comparison.

For the purposes of this algorithm, ***extremely accurate*** will be defined as any champion which achieves a normalized least squares error (NLSE) of **0.0001** or less on the **testing data** under conditions where both the training data and testing data were constructed with zero noise.

All timings quoted in this chapter were performed on a Dell XPS L521X Intel i7 quad core laptop with 16 GB of RAM, and 1 Tb of hard drive, manufactured in Dec 2012 (our test machine). Each test problem was trained against 10,000 training examples with 100 features per example and tested against 10,000 testing examples with 100 features per example. Noise was NOT introduced into any of the test problems, so an exact answer was always theoretically possible.

2 General Search Island

The extremely accurate algorithm begins with an RQL search command which sets up a blanket search of a user specified depth and breadth

- (*S0*) **search** universal(D,B,t) **where** island(pareto,standard,100,100,200) op(ξ, $+$, $-$, $*$,/,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)

For our purposes herein we will set the expression depth D=4 and the number of basis functions B=3. But any user specified depth and number of basis functions can be accommodated.

Search command (S0) assumes that one has an SR system at least as capable as the baseline algorithm in Korns (2012), a reasonably competent implementation of pareto front breeding with *onfinal* and *onscore* event handling, a reasonably competent implementation of standard population operators with mutation, crossover, and swarm optimizers for the constant pools. The survivor population size will be 100. The constant pool size will be 100. Each generation 200 serial iterations will be made in universal(4,3,t). This island search is independent of all other search commands in the algorithm.

Search (S0) will provide the same breadth and depth of search and will be as accurate as any existing commercial package – depending upon the implementation of *pareto* and *standard*. That is the good news. The bad news is that (S0) will not be extremely accurate because of the issues already mentioned. The size of the (S0) search space is $(15^{15} * (100 + 2^{18})^{16})^3 = 10.0\text{E}{+}312$. So the serial search at 200 iterations per generation will take longer than the age of the universe to complete. Meaning that we can't count on serial search and evolutionary search is powerful but not extremely accurate.

Therefore, if we wish to achieve extreme accuracy on $U_2(1)$ and $U_1(3)$, additional search commands will have to be added to the algorithm. These search commands will be executed independently and asynchronously from search (S0). Taken as a whole, general search (S0) together with the specialized searches to be added will constitute the entire extreme accuracy algorithm. The additional specialized search commands will carve out subsets of $U_2(1)$ and $U_1(3)$ which are amenable to serial search in practical time, will carve out the subsets which are tractable for evolutionary search, and using algebraic manipulations and mathematical regression equivalences will carve out the subsets which can be solved with complex search commands.

There will be 24, *which for cloud deployment can be expanded into 80 searches*, of these additional RQL search commands in the algorithm. Each RQL search command sets up a search island independent of all other search islands. This allows the algorithm to be easily distributed across multiple computers or in a cloud environment. The champions from each island are gathered together with the most fit champion being the answer to this RQL query.

The algorithm's claim of extreme accuracy is supported by what might be called an *informal argument* rather than a formal proof. A brief sketch of the informal

arguments will accompany each of the 24 RQL commands with, hopefully, enough information and examples to allow the reader to understand the basic reasoning supporting the claim of extreme accuracy.

3 $U_1(3)$ Search Island

The RQL search command covering the space $U_1(3)$ is thankfully fairly straightforward and the space responds very well to evolutionary search.

- (*S1*) **search** regress($f_0(v_0,v_1)$,$f_1(v_2,v_3)$,$f_2(v_4,v_5)$) **where** island(smart,standard, 10,25,200) op(ξ, +, −, ∗,/,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,inv)

Search command (S1) performs multiple regressions with three basis functions, each of which is in U_1 and looks like f(t,t). Each of the variables v_0 thru v_5 contain a single feature from the set x_0 thru x_{99}. Each of f_0, f_1, and f_2 are function variables contains functions from the set $\mathbf{F} \cup \xi \cup$ inv. From the terms, t, all embedded constants can be eliminated because they cancel out of the basis function and enhance the regression coefficient for the basis function as shown in the following examples.

- (*E6*) regress(c_0+v_0)=a+b*(c_0+v_0)=a+(b*c_0)+b*v_0 = regress(v_0)
- (*E7*) regress(c_0/v_0)=a+b*(c_0/v_0)=a+(b*c_0)/v_0 = regress(inv(v_0))
- (*E8*) regress(cos(c_0))=a+b*cos(c_0)=c_1

Since we can eliminate all of the embedded constants from each term in U_1, we are left with *regress($f_0(v_0,v_1)$,$f_1(v_2,v_3)$,$f_2(v_4,v_5)$)* as our search goal.

4 Search Island S2

The RQL search command covering the space $f_0(f_1(v_0,v_1))$ in U_2 is necessary because large portions of this space do not respond well to evolutionary methods.

- (*S2*) **search** regress($f_0(f_1(v_0,v_1))$) **where** island(smart,standard,10,25,200)
  f_0(ξ,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  f_1(ξ, +, −, ∗, /,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)

Search command (S2) performs single regressions where each of the variables v_0 thru v_1 contain single features from the set x_0 thru x_{99}. The search space size is $12 * 16 * 100 * 100$ =1.92 M. At 200 serial iterations per generation, this search will require a maximum of 9,600 generations. On our test machine, each generation requires 0.00012 h. So the maximum time required for this search island to complete is 1.152 h.

5 Search Island S3

The RQL search command covering the space $f_0(f_1(c_0,v_0))$ in U_2 allows the algorithm to carve out a large space which responds very well to evolutionary search methods.

- (*S3*) **search** regress($f_0(f_1(c_0,v_0))$) **where** island(smart,standard,10,25,200) f_0(inv,sqroot,square,cube,quart,exp,ln) f_1(rdiv,$*$, $+$, $-$,rsub)

Search command (S3) performs single regressions where the variable v_0 contains features from the set x_0 thru x_{99}. The function rdiv is defined as $\text{rdiv}(c_0,v_0)=v_0/c_0$. The function rsub is defined as $\text{rsub}(c_0,v_0)=v_0-c_0$. The search space size is $7 * 5 * 2^{18} * 100 = 917\,\text{M}$. All of the unary functions are mostly monotonic. The binary operators are all monotonic. This very large search space responds well to evolutionary methods.

6 Search Islands S4 Thru S15

The RQL search command also covering the space $f_0(f_1(c_0,v_0))$ in U_2 but where f_0 are all of the unary functions allows the algorithm to isolate the space which do not respond well to evolutionary search methods. With these search islands the algorithm carves out a series of difficult spaces to be searched serially.

- (*S4*) **search** regress($\cos(c_0+v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S5*) **search** regress($\cos(c_0*v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S6*) **search** regress($\cos(c_0/v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S7*) **search** regress($\sin(c_0+v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S8*) **search** regress($\sin(c_0*v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S9*) **search** regress($\sin(c_0/v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S10*) **search** regress($\tan(c_0+v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S11*) **search** regress($\tan(c_0*v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S12*) **search** regress($\tan(c_0/v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S13*) **search** regress($\tanh(c_0+v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S14*) **search** regress($\tanh(c_0*v_0)$) **where** island(smart,standard,10,25,2,000)
- (*S15*) **search** regress($\tanh(c_0/v_0)$) **where** island(smart,standard,10,25,2,000)

Search commands (S4) thru (S15) perform single regressions where the variable v_0 contains features from the set x_0 thru x_{99}. The reverse argument order $f_0(v_0,c_0)$ and the binary operators rdiv and rsub, from S(3), do not have to be searched because these trigonometric functions all share the following properties.

- (*E9*) $\text{regress}(\cos(v_0+c_0))=\text{regress}(\cos(c_0+v_0))$
- (*E10*) $\text{regress}(\cos(v_0-c_0))=\text{regress}(\cos(c_1-v_0))$
- (*E11*) $\text{regress}(\cos(v_0/c_0))=\text{regress}(\cos(c_1*v_0))$

The search space size, for each of these islands, is $2^{18} * 100 = 26.2144$ M. At 2,000 serial iterations per generation, this search will require a maximum of 13,107 generations. On our test machine, each generation requires 0.0001998 h. So the maximum time required for this search island to complete is 2.619 h.

Taken together searches (S3) thru (S15) cover the entire space of $f_0(f_1(c_0,v_0))$ and $f_0(f_1(v_0,c_0))$ where $f_0(+,-,*,/)$ and f_1(inv,cos,sin,tan,tanh,sqroot,square,cube,quart, exp,ln).

7 Search Islands S16

The RQL search command covering the space $f_1(f_0(v_0),f_2(v_1,v_2))$ in U_2 allows the algorithm to carve out a large space with both evolutionary and serial search methods. Search (S5) performs single regressions where the variables v_0, v_1, and v_2 contain single features from the set x_0 thru x_{99}.

- (*S16*) **search** regress($f_1(f_0(v_0),f_2(v_1,v_2))$) **where** island(smart,standard,10,25,400)
  $f_0(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  $f_1(+,-$,rsub,*,/,rdiv)
  $f_2(+,-,*,/)$

This space is so large that, in its cloud version, it must be carved up into 24 separate island searches in order to achieve results in practical time. The **where** clause for each search (S5.1) thru (S5.24) island contains the following.

- **where**
  island(smart,standard,10,25,400)
  $f_0(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)

  The space is carved up by expanding the f_1 and f_2 functions as shown below.

- (*S16.1*) **search** regress($f_0(v_0)/(v_1/v_2)$) **where** ... as above...
- (*S16.2*) **search** regress($f_0(v_0)/(v_1*v_2)$) **where** ... as above...
- (*S16.3*) **search** regress($f_0(v_0)/(v_1+v_2)$) **where** ... as above...
- (*S16.4*) **search** regress($f_0(v_0)/(v_1-v_2)$) **where** ... as above...
- (*S16.5*) **search** regress($f_0(v_0)*(v_1/v_2)$) **where** ... as above...
- (*S16.6*) **search** regress($f_0(v_0)*(v_1*v_2)$) **where** ... as above...
- (*S16.7*) **search** regress($f_0(v_0)*(v_1+v_2)$) **where** ... as above...
- (*S16.8*) **search** regress($f_0(v_0)*(v_1-v_2)$) **where** ... as above...
- (*S16.9*) **search** regress($(v_1/v_2)/f_0(v_0)$) **where** ... as above...
- (*S16.10*) **search** regress($(v_1*v_2)/f_0(v_0)$) **where** ... as above...
- (*S16.11*) **search** regress($(v_1+v_2)/f_0(v_0)$) **where** ... as above...
- (*S16.12*) **search** regress($(v_1-v_2)/f_0(v_0)$) **where** ... as above...
- (*S16.13*) **search** regress($f_0(v_0)+(v_1/v_2)$) **where** ... as above...
- (*S16.14*) **search** regress($f_0(v_0)+(v_1*v_2)$) **where** ... as above...
- (*S16.15*) **search** regress($f_0(v_0)+(v_1+v_2)$) **where** ... as above...

- (*S16.16*) **search** regress($f_0(v_0)+(v_1-v_2)$) **where** ... as above...
- (*S16.17*) **search** regress($f_0(v_0)-(v_1/v_2)$) **where** ... as above...
- (*S16.18*) **search** regress($f_0(v_0)-(v_1*v_2)$) **where** ... as above...
- (*S16.19*) **search** regress($f_0(v_0)-(v_1+v_2)$) **where** ... as above...
- (*S16.20*) **search** regress($f_0(v_0)-(v_1-v_2)$) **where** ... as above...
- (*S16.21*) **search** regress($(v_1/v_2)-f_0(v_0)$) **where** ... as above...
- (*S16.22*) **search** regress($(v_1*v_2)-f_0(v_0)$) **where** ... as above...
- (*S16.23*) **search** regress($(v_1+v_2)-f_0(v_0)$) **where** ... as above...
- (*S16.24*) **search** regress($(v_1-v_2)-f_0(v_0)$) **where** ... as above...

The search space size, for each of these islands, is $100 * 100 * 100 * 12 = 12$ M. At 400 serial iterations per generation, this search will require a maximum of 30,000 generations. On our test machine, each generation requires 0.00021 h. So the maximum time required for this search island to complete is 6.3 h.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target $y = (1.57 + (2.13 * (cos(x99) * (x98/x97))))$, the evolutionary search normally finds the target in less than half of the maximum serial time.

Taken together searches (S16.1) thru (S16.24) cover the entire space of (S16).

8 Search Islands S17

The RQL search command covering the space $f_1(f_0(v_0,v_1),f_2(v_2,v_3))$ in U_2 allows the algorithm to carve out a large space with both evolutionary and serial search methods. Search (S17) performs single regressions where the variables v_0, v_1, v_2, and v_3 contain single features from the set x_0 thru x_{99}. This is by far the largest and most costly search required to cover U_2.

- (*S17*) **search** regress($f_0(f_1(v_0,v_1),f_2(v_2,v_3))$)
  where island(smart,standard,10,25,4,000)
  $f_0(*,/)$
  $f_1(+,-,*,/)$
  $f_2(+,-,*,/)$

The reason that we can eliminate the $+$ and $-$ operators from f_0 is precisely because those expansions are linear in two basis functions and will be solved independently by search island (S1). Therefore we do not have to expand them here. Nevertheless, even this reduced space is so large that it must be carved up into 32 separate island searches in order to achieve results in practical time. The space is carved up by expanding the f_0, f_1, and f_2 functions as shown below.

- (*S17.1*) **search** regress($(v_0+v_1)*(v_2+v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.2*) **search** regress($(v_0+v_1)*(v_2-v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.3*) **search** regress($(v_0+v_1)*(v_2*v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.4*) **search** regress($(v_0+v_1)*(v_2/v_3)$) **where** island(smart,standard,10,25,4,000)

- (*S17.5*) **search** regress((v0+v1)/(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.6*) **search** regress((v0+v1)/(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.7*) **search** regress((v0+v1)/(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.8*) **search** regress((v0+v1)/(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.9*) **search** regress((v0−v1)∗(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.10*) **search** regress((v0−v1)∗(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.11*) **search** regress((v0−v1)∗(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.12*) **search** regress((v0−v1)∗(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.13*) **search** regress((v0−v1)/(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.14*) **search** regress((v0−v1)/(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.15*) **search** regress((v0−v1)/(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.16*) **search** regress((v0−v1)/(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.17*) **search** regress((v0∗v1)∗(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.18*) **search** regress((v0∗v1)∗(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.19*) **search** regress((v0∗v1)∗(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.20*) **search** regress((v0∗v1)∗(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.21*) **search** regress((v0∗v1)/(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.22*) **search** regress((v0∗v1)/(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.23*) **search** regress((v0∗v1)/(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.24*) **search** regress((v0∗v1)/(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.25*) **search** regress((v0/v1)∗(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.26*) **search** regress((v0/v1)∗(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.27*) **search** regress((v0/v1)∗(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.28*) **search** regress((v0/v1)∗(v2/v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.29*) **search** regress((v0/v1)/(v2+v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.30*) **search** regress((v0/v1)/(v2−v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.31*) **search** regress((v0/v1)/(v2∗v3)) **where** island(smart,standard,10,25,4,000)
- (*S17.32*) **search** regress((v0/v1)/(v2/v3)) **where** island(smart,standard,10,25,4,000)

The search space size, for each of these islands, is $100*100*100*100 = 100\,\text{M}$. At 4,000 serial iterations per generation, this search will require a maximum of 25,000 generations. On our test machine, each generation requires 0.009 h. So the maximum time required for this search island to complete is $225\,\text{h} = 9.375$ days.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target *y = (1.57+(2.13∗((x97−x96)/(x98+x99))))*, the evolutionary search normally finds the target in less than a quarter of the maximum serial time.

Taken together searches (S17.1) thru (S17.32) cover the entire space of (S17).

9 Search Island S18

The RQL search command covering the space $f_0(f_1(v_0),f_2(v_1))$ in U_2 allows the algorithm to carve out a large space with both evolutionary and serial search methods. Search (S18) performs single regressions where the variables v_0, and v_1 contain single features from the set x_0 thru x_{99}.

- (*S18*) **search** regress($f_0(f_1(v_0),f_2(v_1))$) **where** island(smart,standard,10,25,200)
  $f_0(+,-,*,/)$
  $f_1(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  $f_2(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)

The search space size, for each of this island, is $100 * 100 * 12 * 12 * 4 = 5.76$ M. At 200 serial iterations per generation, this search will require a maximum of 28,800 generations. On our test machine, each generation requires 0.000135 h. So the maximum time required for this search island to complete is 3.9 h.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target $y = (1.57+(2.13*(ln(x98)/quart(x99))))$, the evolutionary search normally finds the target in less than half of the maximum serial time.

10 Search Island S19

The RQL search command covering the space $f_0(f_1(v_0),f_2(c_0,v_1))$ in U_2 allows the algorithm to carve out a large space where evolutionary search and serial search methods are both intractable. The formal RQL search command is.

- (*E12*) **search** regress($f_1(f_0((v_0),f_2(c_0,v_1)))$ **where** island(smart,standard,10,25,200)
  $f_0(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  $f_1(+,-,*,/$,rsub,rdiv)
  $f_2(+,-,*,/$,rsub,rdiv)

If we are to try E12 by evolutionary search we run into trouble with test problems such as $y = cos(v_0)/(c_0+v_1)$. If we try serial search we see that the size of this space is $100 * 100 * 12 * 6 * 6 * 2^{18} = 1.13$ T which at 200 iterations per generation will require 5,662,310,400 generations. On our test computer each generation requires 0.00021 h. So we will finish testing all possible serial combinations in approximately 1,189,085 h = 49,545 days = 135 years.

Since searching for (E12) is not practical under any approach, we take a giant leap and search for the following.

- (*S19*) **search** regress($f_0(v_0)$,$1/f_0(v_0)$,v_1,$1/v_1$,$f_0(v_0)*v_1$,$f_0(v_0)/v_1$,$v_1/f_0(v_0)$,$1/(v_1*f_0(v_0))$)
  where island(smart,standard,10,25,200)
  $f_0(\xi$,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  onfinal(regress($poly$))

There is a remarkable difference between (E12) and (S19). Search (E12) performs single regression; while, search (S19) performs multiple regressions where the variables v_0, and v_1 contain features from the set x_0 thru x_{99} (*the* **onfinal** *clause simply eliminates all zero or near zero coefficient terms from the final answer and converts to simpler form if possible*). Showing/Learning how these two searches relate to each other will require a bit of simple *regression math* and will help wake us up.

First, notice that search (E12) can be expanded into the following 36 single regression cases with the following equivalent regressions. Notice that all except the three *bolded* cases can be expanded into equivalent simpler regressions.

- (*E12.1*) regress($f_0(v_0)+(c_0+v_1)$) = regress($f_0(v_0)$,v_1)
- (*E12.2*) regress($f_0(v_0)-(c_0+v_1)$) = regress($f_0(v_0)$,v_1)
- (*E12.3*) **regress($\mathbf{f_0(v_0)/(c_0+v_1)}$) = regress($\mathbf{f_0(v_0)/(c_0+v_1)}$)**
- (*E12.4*) regress($f_0(v_0)*(c_0+v_1)$) = regress($f_0(v_0)$,$f_0(v_0)*v_1$)
- (*E12.5*) regress($(c_0+v_1)-f_0(v_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.6*) regress($(c_0+v_1)/f_0(v_0)$) = regress($1/f_0(v_0)$,$v_1/f_0(v_0)$)
- (*E12.7*) regress($f_0(v_0)+(c_0-v_1)$) = regress($f_0(v_0)$,v_1)
- (*E12.8*) regress($f_0(v_0)-(c_0-v_1)$) = regress($f_0(v_0)$,v_1))
- (*E12.9*) **regress($\mathbf{f_0(v_0)/(c_0-v_1)}$) = regress($\mathbf{f_0(v_0)/(c_0-v_1)}$)**
- (*E12.10*) regress($f_0(v_0)*(c_0-v_1)$) = regress($f_0(v_0)$,$f_0(v_0)*v_1$)
- (*E12.11*) regress($(c_0-v_1)-f_0(v_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.12*) regress($(c_0-v_1)/f_0(v_0)$) = regress($1/f_0(v_0)$,$v_1/f_0(v_0)$)
- (*E12.13*) regress($f_0(v_0)+(c_0*v_1)$) = regress($f_0(v_0)$,v_1)
- (*E12.14*) regress($f_0(v_0)-(c_0*v_1)$) = regress($f_0(v_0)$,v_1)
- (*E12.15*) regress($f_0(v_0)/(c_0*v_1)$) = regress($f_0(v_0)/v_1$)
- (*E12.16*) regress($f_0(v_0)*(c_0*v_1)$) = regress($f_0(v_0)*v_1$)
- (*E12.17*) regress($(c_0*v_1)-f_0(v_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.18*) regress($(c_0*v_1)/f_0(v_0)$) = regress($v_1/f_0(v_0)$)
- (*E12.19*) regress($f_0(v_0)+(c_0/v_1)$) = regress($f_0(v_0)$,$1/v_1$)
- (*E12.20*) regress($f_0(v_0)-(c_0/v_1)$) = regress($f_0(v_0)$,$1/v_1$)
- (*E12.21*) regress($f_0(v_0)/(c_0/v_1)$) = regress($f_0(v_0)*v_1$)
- (*E12.22*) regress($f_0(v_0)*(c_0/v_1)$) = regress($f_0(v_0)/v_1$)
- (*E12.23*) regress($(c_0/v_1)-f_0(v_0)$) = regress($f_0(v_0)$,$1/v_1$)
- (*E12.24*) regress($(c_0/v_1)/f_0(v_0)$) = regress($1/(f_0(v_0)*v_1)$)
- (*E12.25*) regress($f_0(v_0)+(v_1-c_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.26*) regress($f_0(v_0)-(v_1-c_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.27*) **regress($\mathbf{f_0(v_0)/(v_1-c_0)}$) = regress($\mathbf{f_0(v_0)/(v_1-c_0)}$)**
- (*E12.28*) regress($f_0(v_0)*(v_1-c_0)$) = regress($f_0(v_0)$,$f_0(v_0)*v_1$)
- (*E12.29*) regress($(v_1-c_0)-f_0(v_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.30*) regress($(v_1-c_0)/f_0(v_0)$) = regress($1/f_0(v_0)$,$v_1/f_0(v_0)$)
- (*E12.31*) regress($f_0(v_0)+(v_1/c_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.32*) regress($f_0(v_0)-(v_1/c_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.33*) regress($f_0(v_0)/(v_1/c_0)$) = regress($f_0(v_0)/v_1$)
- (*E12.34*) regress($f_0(v_0)*(v_1/c_0)$) = regress($f_0(v_0)*v_1$)
- (*E12.35*) regress($(v_1/c_0)-f_0(v_0)$) = regress($f_0(v_0)$,v_1)
- (*E12.36*) regress($(v_1/c_0)/f_0(v_0)$) = regress($v_1/f_0(v_0)$)

Eliminating the three bolded cases and collecting all the equivalent regression terms from the right hand side of Eqs. (E12.1) thru (E12.36) we arrive at *regress($f_0(v_0)$,$1/f_0(v_0)$,v_1,$1/v_1$,$f_0(v_0)*v_1$,$f_0(v_0)/v_1$,$v_1/f_0(v_0)$,$1/(v_1*f_0(v_0))$)* which is equivalent to search (S19). Addressing the bolded cases (E12.3), (E12.9), and (E12.27) is more complicated and will be left to the following section.

The search space size, for island (S19), is $100*100*12 = 120,000$. At 200 serial iterations per generation, this search will require a maximum of 600 generations. On our test machine, each generation requires 0.000435 h. So the maximum time required for this search island to complete is 0.261 h.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target *y=1.57+(2.13*(ln(x98)* (23.583−x99)))*, the evolutionary search normally finds the target in less than a third of the maximum serial time.

11 Search Island S20

The previous search island (S19) addressed the RQL search command covering the space $f_0(f_1(v_0),f_2(c_0,v_1))$ see (E12); however, three more complicated regression cases (E12.3), (E12.9), and (E12.27) were left to this section. If we are to try these three cases by evolutionary search we run into trouble with test problems such as $y=cos(v_0)/(c_0-v_1)$. If we try serial search we see that the size of each of these spaces is $100 * 100 * 12 * 2^{18} = 31.5$ B which at 200 iterations per generation will require 157,286,400 generations. On our test computer each generation requires 0.000135 h. So we will finish testing all possible serial combinations in approximately 21,233 h = 9,830 days = 2.4 years.

Since searching for (E12.3), (E12.9), and (E12.27) is not practical under any approach, we take a giant leap and search for the following.

- (*S20*) **search** regress(v_1*y,v_1,$f_0(v_0)$) **where** island(smart,standard,10,25,200)
  isolate(true)
  f_0(ξ,inv,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln)
  onscore(0.0,600,regress(1.0/f_0(c_0,v_1),f_0(v_0)/f_0(c_0,v_1))
  where
  island(smart,standard,10,25,200)
  f_0(+, −,rsub)
  c_0(1.0/$w0$))

Not only is there is a big difference between (E12.3), (E12.9), (E12.27), and (S20); but, the search goal in (S20) contains the term ***y*** which is the dependent variable in the regression. While this may seem invalid in the domain of any basic regression course taught in university, we will show below why it is perfectly valid in this context.

First, the ***isolate(true)*** clause in (S20) keeps any of the champions from the first **where** clause from reaching the final solution list. While there is nothing wrong with using ***y*** during training (***y*** is used repeatedly during fitness calculation while training), allowing ***y*** to appear in the final solution causes problems because ***y*** will not be available during testing. So only solutions containing the features x_0 thru x_{99} are appropriate for final champions.

Second, the ***onscore*** clause erases all champions except the top champion and proceeds to convert the top champion into the form specified in the ***onscore*** goal and **where** clauses. The ***onscore*** clause is triggered when the first search achieves a fitness score of 0.0 or when the number of training generations reaches 600. This initiates a completely new search for *regress(1.0/$f_0(c_0,\$v_1\$),\$f_0\$(\$v_0\$)/f_0(c_0,\$v_1\$))* which does not contain the inappropriate ***y*** term. Therefore the term ***y*** is used only during training and setup but never in the final solution.

In order to understand how the two cascading searches in (S20) relate to the three difficult cases (E12.3), (E12.9), and (E12.27), we must observe the following regression equivalence chains in the situation where there is zero noise.

- (*E12.9*) **regress($\mathbf{f_0(v_0)/(c_0-v_1)}$)** $->$
- (*E12.9.1*) $y=a0+b0(f_0(v_0)/(c_0-v_1))->$
- (*E12.9.2*) $y(c_0-v_1)=a0(c_0-v_1)+b0f_0(v_0)->$
- (*E12.9.3*) $c_0y - v_1y=a0(c_0-v_1)+b0f_0(v_0)->$
- (*E12.9.4*) $c_0y=a0(c_0-v_1)+b0f_0(v_0)+v_1y->$
- (*E12.9.5*) $y=(a0(c_0-v_1)/c_0)+(b0f_0(v_0)/c_0)+(v_1y/c_0)->$
- (*E12.9.6*) $y=a0 - (a0/c_0)*v_1+(b0/c_0)*f_0(v_0)+(1/c_0)*v_1y->$
- (*E12.9.7*) **regress($\mathbf{v_1*y,v_1,f_0(v_0)}$))** $->$
- (*E12.9.8*) $y=a1+w0*v_1y - w1v_1+w2*f_0(v_0)->$
- (*E12.9.9*) $w0=(1/c_0)->$
- (*E12.9.10*) $\mathbf{c_0=(1/w0)}->$
- (*E12.9.11*) $regress(f_0(v_0)/((1/w0)+v_1))->$
- (*E12.9.12*) **regress($\mathbf{f_0(v_0)/(c_0+v_1)}$)**

Similar equivalence chains are true for (E12.3) and (E12.27). Taken all three together, we see that the answers to (E12.3), (E12.9), and (E12.27) will be either *regress($f_0(v_0)/((1/w0)+v_1)$)*, *regress($f_0(v_0)/((1/w0)-v_1)$)*, or *regress($f_0(v_0)/(v_1-(1/w0))$)* which is exactly what search island (S20) proposes.

Let's use this actual example, suppose the target formula is *$y=0.913+(62.13*ln(x_2)/(x_4-23.451))$*. The first search in (S20) *regress($v_1*y,v_1,f_0(v_0)$))* discovers that the champion *$y=0.913-(0.039*x_4)+(0.0426421*(x_4*y)-(2.65*ln(x_2)))$;* achieves a fitness score of 0.0. This low fitness score triggers the **onscore** clause (otherwise the onscore clause would be triggered by more than 600 generations passing). The **onscore** clause substitutes for the items enclosed in $ sign pairs and searches for the following goal *regress(1.0/$f_0(c_0,x_4)$,$ln(x_2)/f_0(c_0,x_4)$) where $f_0(+,-,rsub)$ $c_0(23.451)$* (**since (1/.0426421)=23.451**). The final answer is *$y=0.913+(62.13*(ln(x_2)/(x_4-23.451)))$* with a final fitness score of 0.0.

The search space size, for island (S20), is $100*100*12 = 120,000$. At 200 serial iterations per generation, this search will require a maximum of 600 generations. On our test machine, each generation requires 0.0003 h. So the maximum time required for this search island to complete is 0.18 h and is followed immediately by a brief search for the onscore goal.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target *$y = 0.913+(62.13*ln(x_2)/(x_4-23.451))$*, the evolutionary search normally finds the target in less than a half of the maximum serial time.

12 Search Island S21

The RQL search command covering the space $f_1(f_0(c_0,v_0),f_2(v_1,v_2))$ in U_2 allows the algorithm to carve out a large space where evolutionary search and serial search methods are both intractable. The formal RQL search command is.

- (*E13*) **search** regress($f_1(f_0(c_0,v_0),f_2(v_1,v_2))$) **where** island(smart,standard,10,25,200)
  $f_0(+,*,/)$
  $f_1(+,-,*,/$,rsub,rdiv)
  $f_2(+,-,*,/)$

If we are to try E13 by evolutionary search we run into trouble with test problems such as $y=(v_1*v_2)/(c_0+v_0)$. If we try serial search we see that the size of this space is $100 * 100 * 100 * 3 * 6 * 4 * 2^{18} = 18.87$ T which at 200 iterations per generation will require 94,371,840,000 generations. On our test computer each generation requires 0.00021 h. So we will finish testing all possible serial combinations in approximately 19,818,086 h = 825,753 days = 2,256 years.

Since searching for (E13) is not practical under any approach, we take a giant leap and search for the following.

- (*S21*) **search** regress(v_0,v_1,v_2,v_1*v_2,v_1/v_2,1.0/v_0,v_0*v_1,v_0*v_2,
  $v_0*v_1*v_2$,(v_0*v_1)/v_2,v_1/v_0,v_2/v_0,(v_1*v_2)/v_0,$v_1/(v_0*v_2)$,
  $1/(v_1+v_2)$,$v_0/(v_1+v_2)$,$1/(v_1-v_2)$,$v_0/(v_1-v_2)$,$1/(v_1*v_2)$,$v_0/(v_1*v_2)$,
  v_2/v_1,(v_0*v_2)/v_1,$1/(v_0*(v_1+v_2))$,$1/(v_0*v_1*v_2)$,$v_2/(v_0*v_1)$)
  where island(smart,standard,10,25,200)
  onfinal(regress($poly$))

Of course there is a big difference between (E13) and (S21). Search (E13) performs single regression; while, search (S21) performs multiple regressions where the variables v_0, v_1, and v_2 contain features from the set x_0 thru x_{99} (*the* **onfinal** *clause simply eliminates all zero or near zero coefficient terms from the final answer and converts to simpler form if possible*). Showing/Learning how these two searches relate to each other will require a bit of simple *regression math* and our close attention.

First, notice that search (E13) can be expanded into the following 72 single regression cases with the following equivalent regressions. Notice that all except the three *bolded* cases can be expanded into equivalent simpler regressions.

- (*E13.1*) regress($(c_0+v_0)+(v_1+v_2)$) = regress(v_0,v_1,v_2)
- (*E13.2*) regress($(c_0+v_0)+(v_1-v_2)$) = regress(v_0,v_1,v_2)
- (*E13.3*) regress($(c_0+v_0)+(v_1*v_2)$) = regress(v_0,v_1*v_2)
- (*E13.4*) regress($(c_0+v_0)+(v_1/v_2)$) = regress(v_0,v_1/v_2)
- (*E13.5*) regress($(c_0+v_0)-(v_1+v_2)$) = regress(v_0,v_1,v_2)
- (*E13.6*) regress($(c_0+v_0)-(v_1-v_2)$) = regress(v_0,v_1,v_2)
- (*E13.7*) regress($(c_0+v_0)-(v_1*v_2)$) = regress(v_0,v_1*v_2)
- (*E13.8*) regress($(c_0+v_0)-(v_1/v_2)$) = regress(v_0,v_1/v_2)
- (*E13.9*) regress($(c_0+v_0)*(v_1+v_2)$) = regress(v_1,v_2,v_0*v_1,v_0*v_2)

- (*E13.10*) regress((c$_0$+v$_0$)∗(v$_1$−v$_2$)) = regress(v$_1$,v$_2$,v$_0$∗v$_1$,v$_0$∗v$_2$)
- (*E13.11*) regress((c$_0$+v$_0$)∗(v$_1$∗v$_2$)) = regress(v$_1$∗v$_2$,v$_0$∗v$_1$∗v$_2$)
- (*E13.12*) regress((c$_0$+v$_0$)∗(v$_1$/v$_2$)) = regress(v$_1$/v$_2$,(v$_0$∗v$_1$)/v$_2$)
- (*E13.13*) regress((c$_0$+v$_0$)/(v$_1$+v$_2$)) = regress(1/(v$_1$+v$_2$),v$_0$/(v$_1$+v$_2$))
- (*E13.14*) regress((c$_0$+v$_0$)/(v$_1$−v$_2$)) = regress(1/(v$_1$+v$_2$),v$_0$/(v$_1$+v$_2$))
- (*E13.15*) regress((c$_0$+v$_0$)/(v$_1$∗v$_2$)) = regress(1/(v$_1$∗v$_2$),v$_0$/(v$_1$∗v$_2$))
- (*E13.16*) regress((c$_0$+v$_0$)/(v$_1$/v$_2$)) = regress(v$_2$/v$_1$,(v$_0$∗v$_2$)/v$_1$)
- (*E13.17*) regress((v$_1$+v$_2$)−(c$_0$+v$_0$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.18*) regress((v$_1$−v$_2$)−(c$_0$+v$_0$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.19*) regress((v$_1$∗v$_2$)−(c$_0$+v$_0$)) = regress(v$_0$,v$_1$∗v$_2$)
- (*E13.20*) regress((v$_1$/v$_2$)−(c$_0$+v$_0$)) = regress(v$_0$,v$_1$/v$_2$)
- (*E13.21*) **regress((v$_1$+v$_2$)/(c$_0$+v$_0$)) = regress((v$_1$+v$_2$)/(c$_0$+v$_0$))**
- (*E13.22*) **regress((v$_1$−v$_2$)/(c$_0$+v$_0$)) = regress((v$_1$−v$_2$)/(c$_0$+v$_0$))**
- (*E13.23*) **regress((v$_1$∗v$_2$)/(c$_0$+v$_0$)) = regress((v$_1$∗v$_2$)/(c$_0$+v$_0$))**
- (*E13.24*) **regress((v$_1$/v$_2$)/(c$_0$+v$_0$)) = regress((v$_1$/v$_2$)/(c$_0$+v$_0$))**
- (*E13.25*) regress((c$_0$∗v$_0$)+(v$_1$+v$_2$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.26*) regress((c$_0$∗v$_0$)+(v$_1$−v$_2$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.27*) regress((c$_0$∗v$_0$)+(v$_1$∗v$_2$)) = regress(v$_0$,v$_1$∗v$_2$)
- (*E13.28*) regress((c$_0$∗v$_0$)+(v$_1$/v$_2$)) = regress(v$_0$,v$_1$/v$_2$)
- (*E13.29*) regress((c$_0$∗v$_0$)−(v$_1$+v$_2$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.30*) regress((c$_0$∗v$_0$)−(v$_1$−v$_2$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.31*) regress((c$_0$∗v$_0$)−(v$_1$∗v$_2$)) = regress(v$_0$,v$_1$∗v$_2$)
- (*E13.32*) regress((c$_0$∗v$_0$)−(v$_1$/v$_2$)) = regress(v$_0$,v$_1$/v$_2$)
- (*E13.33*) regress((c$_0$∗v$_0$)∗(v$_1$+v$_2$)) = regress(v$_0$∗v$_1$,v$_0$∗v$_2$)
- (*E13.34*) regress((c$_0$∗v$_0$)∗(v$_1$−v$_2$)) = regress(v$_0$∗v$_1$,v$_0$∗v$_2$)
- (*E13.35*) regress((c$_0$∗v$_0$)∗(v$_1$∗v$_2$)) = regress(v$_0$∗v$_1$∗v$_2$)
- (*E13.36*) regress((c$_0$∗v$_0$)∗(v$_1$/v$_2$)) = regress((v$_0$∗v$_1$)/v$_2$)
- (*E13.37*) regress((c$_0$∗v$_0$)/(v$_1$+v$_2$)) = regress(v$_0$/(v$_1$+v$_2$))
- (*E13.38*) regress((c$_0$∗v$_0$)/(v$_1$−v$_2$)) = regress(v$_0$/(v$_1$+v$_2$))
- (*E13.39*) regress((c$_0$∗v$_0$)/(v$_1$∗v$_2$)) = regress(v$_0$/(v$_1$∗v$_2$))
- (*E13.40*) regress((c$_0$∗v$_0$)/(v$_1$/v$_2$)) = regress((v$_0$∗v$_2$)/v$_1$)
- (*E13.41*) regress((v$_1$+v$_2$)−(c$_0$∗v$_0$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.42*) regress((v$_1$−v$_2$)−(c$_0$∗v$_0$)) = regress(v$_0$,v$_1$,v$_2$)
- (*E13.43*) regress((v$_1$∗v$_2$)−(c$_0$∗v$_0$)) = regress(v$_0$∗v$_1$∗v$_2$)
- (*E13.44*) regress((v$_1$/v$_2$)−(c$_0$∗v$_0$)) = regress((v$_0$∗v$_1$)/v$_2$)
- (*E13.45*) regress((v$_1$+v$_2$)/(c$_0$∗v$_0$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.46*) regress((v$_1$−v$_2$)/(c$_0$∗v$_0$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.47*) regress((v$_1$∗v$_2$)/(c$_0$∗v$_0$)) = regress((v$_1$∗v$_2$)/v$_0$)
- (*E13.48*) regress((v$_1$/v$_2$)/(c$_0$∗v$_0$)) = regress(v$_1$/(v$_0$∗v$_2$))
- (*E13.49*) regress((c$_0$/v$_0$)+(v$_1$+v$_2$)) = regress(1/v$_0$,v$_1$,v$_2$)
- (*E13.50*) regress((c$_0$/v$_0$)+(v$_1$−v$_2$)) = regress(1/v$_0$,v$_1$,v$_2$)
- (*E13.51*) regress((c$_0$/v$_0$)+(v$_1$∗v$_2$)) = regress(1/v$_0$,v$_1$∗v$_2$)
- (*E13.52*) regress((c$_0$/v$_0$)+(v$_1$/v$_2$)) = regress(1/v$_0$,(v$_1$/v$_2$))
- (*E13.53*) regress((c$_0$/v$_0$)−(v$_1$+v$_2$)) = regress(1/v$_0$,v$_1$,v$_2$)
- (*E13.54*) regress((c$_0$/v$_0$)−(v$_1$−v$_2$)) = regress(1/v$_0$,v$_1$,v$_2$)

- (*E13.55*) regress((c$_0$/v$_0$)−(v$_1$∗v$_2$)) = regress(1/v$_0$,v$_1$∗v$_2$)
- (*E13.56*) regress((c$_0$/v$_0$)−(v$_1$/v$_2$)) = regress(1/v$_0$,(v$_1$/v$_2$))
- (*E13.57*) regress((c$_0$/v$_0$)∗(v$_1$+v$_2$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.58*) regress((c$_0$/v$_0$)∗(v$_1$−v$_2$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.59*) regress((c$_0$/v$_0$)∗(v$_1$∗v$_2$)) = regress(f$_0$(v$_0$),v$_1$)
- (*E13.60*) regress((c$_0$/v$_0$)∗(v$_1$/v$_2$)) = regress(f$_0$(v$_0$),v$_1$)
- (*E13.61*) regress((c$_0$/v$_0$)/(v$_1$+v$_2$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.62*) regress((c$_0$/v$_0$)/(v$_1$−v$_2$)) = regress(v$_1$/v$_0$,v$_2$/v$_0$)
- (*E13.63*) regress((c$_0$/v$_0$)/(v$_1$∗v$_2$)) = regress(1(v$_0$∗v$_1$∗v$_2$))
- (*E13.64*) regress((c$_0$/v$_0$)/(v$_1$/v$_2$)) = regress(v$_2$/(v$_0$∗v$_1$))
- (*E13.65*) regress((v$_1$+v$_2$)−(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$,v$_2$)
- (*E13.66*) regress((v$_1$−v$_2$)−(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$,v$_2$)
- (*E13.67*) regress((v$_1$∗v$_2$)−(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$∗v$_2$)
- (*E13.68*) regress((v$_1$/v$_2$)−(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$/v$_2$)
- (*E13.69*) regress((v$_1$+v$_2$)/(c$_0$/v$_0$)) = regress(v$_0$∗v$_1$,v$_0$∗v$_2$)
- (*E13.70*) regress((v$_1$−v$_2$)/(c$_0$/v$_0$)) = regress(v$_0$∗v$_1$,v$_0$∗v$_2$)
- (*E13.71*) regress((v$_1$∗v$_2$)/(c$_0$/v$_0$)) = regress(v$_0$∗v$_1$∗v$_2$)
- (*E13.72*) regress((v$_1$/v$_2$)/(c$_0$/v$_0$)) = regress((v$_0$∗v$_1$)/v$_2$)

Eliminating the four bolded cases and collecting all the equivalent regression terms from the right hand side of Eqs. (E13.1) thru (E13.72) we arrive at search (S21). Addressing the bolded cases (E13.21) thru (E13.24) is more complicated and will be left to the following section.

The search space size, for island (S21), is $100 * 100 * 100 = 1,000,000$. At 200 serial iterations per generation, this search will require a maximum of 5,000 generations. On our test machine, each generation requires 0.0006 h. So the maximum time required for this search island to complete is 3 h.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target *y = (1.57+(2.13∗(x98∗x67)/(23.583−x99)))*, the evolutionary search normally finds the target in less than half of the maximum serial time.

13 Search Island S22

The previous search island (S21) addressed the RQL search command covering the space f$_1$(f$_0$(c$_0$,v$_0$),f$_2$(v$_1$,v$_2$)) see (E13); however, three more complicated regression cases (E13.21) thru (E13.24) were left to this section. If we are to try these three cases by evolutionary search we run into trouble with test problems such as *y = (v$_0$∗v$_1$)/(c$_0$+v$_2$)*. If we try serial search we see that the size of each of these spaces is $100 * 100 * 100 * 2^{18} = 262.144$ B which at 4,000 iterations per generation will require 65,536,000 generations. On our test computer each generation requires 0.00021 h. So we will finish testing all possible serial combinations in approximately 41,287 h = 1,720 days = 47 years.

Since searching for (E13.21) thru (E13.24) is not practical under any approach, we take a giant leap and search for the following.

- (*S22*) **search** regress(v_0∗y,v_0,v_1,v_2,v_1∗v_2,v_1/v_2) **where** island(smart,standard,10,25,200) isolate(true)
  onscore(0.0,5000,regress(f_0($v_1$$,$$v_2$$)/f_1(c_0,$$v_0$$))
  where
  island(smart,standard,10,25,200)
  $f_0(+,-,*,/)$
  f_1(+, −,rsub)
  c_0(1.0/$w0$))

Clearly there is a big difference between (E13.21) thru (E13.24) and (S22), and also the search goal in (S22) contains the term ***y*** which being the dependent variable in the regression. First, the ***isolate(true)*** clause in (S22) keeps any of the champions from the first **where** clause from reaching the final solution list. While there is nothing wrong with using ***y*** during training (***y*** is used repeatedly during fitness calculation while training), allowing ***y*** to appear in the final solution causes problems because ***y*** will not be available during testing. So only solutions containing the features x_0 thru x_{99} are appropriate for final champions.

Second, the ***onscore*** clause erases all champions except the top champion and proceeds to convert the top champion into the form specified in the ***onscore*** goal and **where** clauses. The ***onscore*** clause is triggered when the first search achieves a fitness score of 0.0 or when the number of training generations reaches 5,000. This initiates a completely new search for *regress(f_0($$v_1$$,$$v_2$$)/f_1(c_0,$$v_0$$))* which does not contain the inappropriate ***y*** term. Therefore the term ***y*** is used only during training and setup but never in the final solution.

In order to understand how the two cascading searches in (S22) relate to the four difficult cases (E13.21) thru (E13.24), we must observe the following regression equivalence chains in the situation where there is zero noise.

- (*E13.23*) **regress((v_1∗v_2)/(c_0+v_0))** − >
- (*E13.23.1*) y=a0+b0((v_1∗v_2)/(c_0+v_0))− >
- (*E13.23.2*) y(c_0+v_0)=a0(c_0+v_0)+b0(v_1∗v_2) − >
- (*E13.23.3*) c_0y+v_0y=a0(c_0+v_0)+b0(v_1∗v_2) − >
- (*E13.23.4*) c_0y=a0c_0+a0v_0+b0(v_1∗v_2) - v_0y − >
- (*E13.23.5*) y=a0+(a0/c_0)v_0+(b0/c_0)(v_1∗v_2) - (1/c_0)v_0y − >
- (*E13.23.6*) **regress(v_0∗y,v_0,v_1∗v_2))** − >
- (*E13.23.7*) y=a1 - w0*v_0y - w1v_0+w2*(v_1∗v_2) − >
- (*E13.23.8*) w0=(1/c_0) − >
- (*E13.23.9*) c_0=**(1/w0)** − >
- (*E13.23.10*) regress((v_1∗v_2)/((1/w0)+v_0))− >
- (*E13.23.11*) **regress((v_1∗v_2)/(c_0+v_0))**

Similar equivalence chains are true for (E13.21), (E13.22) and (E13.24). Taken all three together, we see that the answers to (E13.21), (E13.22), (E13.23) and (E13.24) will be *regress(f_0(v_1,v_2)/f_1((1/w0),v_0))* where $f_0(+,-,*,/)$ and f_1(+, −,rsub), which is exactly what search island (S22) proposes.

Let's use this actual example, suppose the target formula is $y = 1.0+(2.0*((x_1 * x_2)/(23.451+x_4))$. The first search in (S22) *regress(v0*y,v0,v1,v2,v1*v2,v1/v2))* discovers that the champion $y = y = 1-(0.0426421*(x_4*y))+(0*x_4)+(0*x_2)+(0*x_1)+(0.085289*(x_2*x_1))+(0*(x_2/x_1))$ achieves a fitness score of 0.0. This low fitness score triggers the **onscore** clause (otherwise the onscore clause would be triggered by more than 5,000 generations passing). The **onscore** clause substitutes for the items enclosed in $ sign pairs and searches for the following goal *regress($f_0(x_1,x_2)$/$f_1(c_0,x_4)$) where* $f_0(+,-,*,/)$ $f_1(+,-,rsub)$ $c_0(23.451)$ (**since (1/0.0426421)** = **23.451**). The final answer is $y=1.0+(2.0*((x_1*x_2)/(23.451+x_4)))$ with a final fitness score of 0.0.

The search space size, for island (S22), is $100 * 100 * 100 * 12 = 720,000$. At 200 serial iterations per generation, this search will require a maximum of 60,000 generations. On our test machine, each generation requires 0.0003 h. So the maximum time required for this search island to complete is 18 h and is followed immediately by a brief search for the onscore goal.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target $y = 1.0+(2.0*((x_1*x_2)/(23.451+x_4)))$, the evolutionary search normally finds the target in less than a third of the maximum serial time.

14 Search Island S23

The RQL search command covering the space $f_1(f_0(c_1,v_1),f_2(c_0,v_0))$ in U_2 allows the algorithm to carve out a large space where evolutionary search and serial search methods are both intractable. The formal RQL search command is.

- (*E14*) **search** regress($f_1(f_0(c_1,v_1),f_2(c_0,v_0))$) **where** island(smart,standard,10,25,200)
  $f_0(+,-,*,/)$
  $f_1(+,-,*,/)$
  $f_2(+,-,*,/)$

If we are to try E14 by evolutionary search we run into trouble with test problems such as $y= (c_1*v_1)/(c_0-v_0)$. If we try serial search we see that the size of this space is $100*100*4*4*4*2^{18}*2^{18} = 43.9$ Quadrillion which at 200 iterations per generation will require 219,902,325,555,200 generations. On our test computer each generation requires .00021hrs. So we will finish testing all possible serial combinations in approximately 461,794,883,665 h = 19,241,453,486 days = 52,716,310 years.

Since searching for (E14) is not practical under any approach, we take a giant leap and search for the following.

- (*S23*) **search** regress(v_0,v_1,1.0/v_0,1.0/v_1,v_0*v_1,v_0/v_1,v_1/v_0,1/(v_0*v_1))
  where island(smart,standard,10,25,200)
  onfinal(regress($poly$))

Of course there is a big difference between (E14) and (S23). Search (E14) performs single regression; while, search (S23) performs multiple regressions where the variables v_0, and v_1 contain features from the set x_0 thru x_{99} (*the* **onfinal** *clause simply eliminates all zero or near zero coefficient terms from the final answer and converts to simpler form if possible*). Showing/Learning how these two searches relate to each other will require a bit of simple *regression math* and our close attention.

First, notice that search (E14) can be expanded into the following 64 single regression cases with the following equivalent regressions. Notice that all except the three *bolded* cases can be expanded into equivalent simpler regressions.

- (*E14.1*) regress($(c_1+v_1)+(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.2*) regress($(c_1+v_1)+(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.3*) regress($(c_1+v_1)+(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.4*) regress($(c_1+v_1)+(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.5*) regress($(c_1-v_1)+(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.6*) regress($(c_1-v_1)+(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.7*) regress($(c_1-v_1)+(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.8*) regress($(c_1-v_1)+(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.9*) regress($(c_1*v_1)+(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.10*) regress($(c_1*v_1)+(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.11*) regress($(c_1*v_1)+(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.12*) regress($(c_1*v_1)+(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.13*) regress($(c_1/v_1)+(c_0+v_0)$) = regress($1/v_1$,v_0)
- (*E14.14*) regress($(c_1/v_1)+(c_0-v_0)$) = regress($1/v_1$,v_0)
- (*E14.15*) regress($(c_1/v_1)+(c_0*v_0)$) = regress($1/v_1$,v_0)
- (*E14.16*) regress($(c_1/v_1)+(c_0/v_0)$) = regress($1/v_1$,$1/v_0$)
- (*E14.17*) regress($(c_1+v_1)-(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.18*) regress($(c_1+v_1)-(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.19*) regress($(c_1+v_1)-(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.20*) regress($(c_1+v_1)-(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.21*) regress($(c_1-v_1)-(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.22*) regress($(c_1-v_1)-(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.23*) regress($(c_1-v_1)-(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.24*) regress($(c_1-v_1)-(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.25*) regress($(c_1*v_1)-(c_0+v_0)$) = regress(v_1,v_0)
- (*E14.26*) regress($(c_1*v_1)-(c_0-v_0)$) = regress(v_1,v_0)
- (*E14.27*) regress($(c_1*v_1)-(c_0*v_0)$) = regress(v_1,v_0)
- (*E14.28*) regress($(c_1*v_1)-(c_0/v_0)$) = regress(v_1,$1/v_0$)
- (*E14.29*) regress($(c_1/v_1)-(c_0+v_0)$) = regress($1/v_1$,v_0)
- (*E14.30*) regress($(c_1/v_1)-(c_0-v_0)$) = regress($1/v_1$,v_0)
- (*E14.31*) regress($(c_1/v_1)-(c_0*v_0)$) = regress($1/v_1$,v_0)
- (*E14.32*) regress($(c_1/v_1)-(c_0/v_0)$) = regress($1/v_1$,$1/v_0$)
- (*E14.33*) regress($(c_1+v_1)*(c_0+v_0)$) = regress(v_0,v_1,v_0*v_1)
- (*E14.34*) regress($(c_1+v_1)*(c_0-v_0)$) = regress(v_0,v_1,v_0*v_1)

- (*E14.35*) regress((c$_1$+v$_1$)∗(c$_0$∗v$_0$)) = regress(v$_0$,v$_0$∗v$_1$)
- (*E14.36*) regress((c$_1$+v$_1$)∗(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$/v$_0$)
- (*E14.37*) regress((c$_1$−v$_1$)∗(c$_0$+v$_0$)) = regress(v$_0$,v$_1$,v$_0$∗v$_1$)
- (*E14.38*) regress((c$_1$−v$_1$)∗(c$_0$−v$_0$)) = regress(v$_0$,v$_1$,v$_0$∗v$_1$)
- (*E14.39*) regress((c$_1$−v$_1$)∗(c$_0$∗v$_0$)) = regress(v$_0$,v$_0$∗v$_1$)
- (*E14.40*) regress((c$_1$−v$_1$)∗(c$_0$/v$_0$)) = regress(1/v$_0$,v$_1$/v$_0$)
- (*E14.41*) regress((c$_1$∗v$_1$)∗(c$_0$+v$_0$)) = regress(v$_1$,v$_0$∗v$_1$)
- (*E14.42*) regress((c$_1$∗v$_1$)∗(c$_0$−v$_0$)) = regress(v$_1$,v$_0$∗v$_1$)
- (*E14.43*) regress((c$_1$∗v$_1$)∗(c$_0$∗v$_0$)) = regress(v$_0$∗v$_1$)
- (*E14.44*) regress((c$_1$∗v$_1$)∗(c$_0$/v$_0$)) = regress(v$_1$/v$_0$)
- (*E14.45*) regress((c$_1$/v$_1$)∗(c$_0$+v$_0$)) = regress(1/v$_1$,v$_0$/v$_1$)
- (*E14.46*) regress((c$_1$/v$_1$)∗(c$_0$−v$_0$)) = regress(1/v$_1$,v$_0$/v$_1$)
- (*E14.47*) regress((c$_1$/v$_1$)∗(c$_0$∗v$_0$)) = regress(v$_0$/v$_1$)
- (*E14.48*) regress((c$_1$/v$_1$)∗(c$_0$/v$_0$)) = regress(1/(v$_0$∗v$_1$))
- (*E14.49*) **regress((c$_1$+v$_1$)/(c$_0$+v$_0$)) = regress(1/(c$_0$+v$_0$),v$_1$/(c$_0$+v$_0$))**
- (*E14.50*) **regress((c$_1$+v$_1$)/(c$_0$−v$_0$)) = regress(1/(c$_0$+v$_0$),v$_1$/(c$_0$+v$_0$))**
- (*E14.51*) regress((c$_1$+v$_1$)/(c$_0$∗v$_0$)) = regress(1/v$_0$,v$_1$/v$_0$)
- (*E14.52*) regress((c$_1$+v$_1$)/(c$_0$/v$_0$)) = regress(v$_0$,v$_0$∗v$_1$)
- (*E14.53*) **regress((c$_1$−v$_1$)/(c$_0$+v$_0$)) = regress(1/(c$_0$+v$_0$),v$_1$/(c$_0$+v$_0$))**
- (*E14.54*) **regress((c$_1$−v$_1$)/(c$_0$−v$_0$)) = regress(1/(c$_0$+v$_0$),v$_1$/(c$_0$+v$_0$))**
- (*E14.55*) regress((c$_1$−v$_1$)/(c$_0$∗v$_0$)) = regress(1/v$_0$,v$_1$/v$_0$)
- (*E14.56*) regress((c$_1$−v$_1$)/(c$_0$/v$_0$)) = regress(v$_0$,v$_0$∗v$_1$)
- (*E14.57*) **regress((c$_1$∗v$_1$)/(c$_0$+v$_0$)) = regress(v$_1$/(c$_0$+v$_0$))**
- (*E14.58*) **regress((c$_1$∗v$_1$)/(c$_0$−v$_0$)) = regress(v$_1$/(c$_0$+v$_0$))**
- (*E14.59*) regress((c$_1$∗v$_1$)/(c$_0$∗v$_0$)) = regress(v$_1$/v$_0$)
- (*E14.60*) regress((c$_1$∗v$_1$)/(c$_0$/v$_0$)) = regress(v$_0$∗v$_1$)
- (*E14.61*) **regress((c$_1$/v$_1$)/(c$_0$+v$_0$)) = regress(1/(v$_1$∗(c$_0$+v$_0$)))**
- (*E14.62*) **regress((c$_1$/v$_1$)/(c$_0$−v$_0$)) = regress(1/(v$_1$∗(c$_0$+v$_0$)))**
- (*E14.63*) regress((c$_1$/v$_1$)/(c$_0$∗v$_0$)) = regress(1/v$_0$∗v$_1$)
- (*E14.64*) regress((c$_1$/v$_1$)/(c$_0$/v$_0$)) = regress(v$_0$/v$_1$)

Eliminating the eight bolded cases and collecting all the equivalent regression terms from the right hand side of Eqs. (E14.1) thru (E14.64) we arrive at search (S23). Addressing the bolded cases (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), (E14.61), and (E14.62) is more complicated and will be left to the following section.

The search space size, for island (S23), is $100 * 100 = 10,000$. At 200 serial iterations per generation, this search will require a maximum of 50 generations. On our test machine, each generation requires 0.0006 h. So the maximum time required for this search island to complete is 0.03 h.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target $y = (1.57 + (2.13 * (3.23 * x67) * (23.583 - x99)))$, the evolutionary search normally finds the target in less than a third of the maximum serial time.

15 Search Island S24

The previous search island (S23) addressed the RQL search command covering the space $f_1(f_0(c_1,v_1),f_2(c_0,v_0))$ see (E14); however, eight more complicated regression cases (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), (E14.61), and (E14.62) were left to this section. If we are to try these eight cases by evolutionary search we run into trouble with test problems such as $y=(c_1*v_1)/(c_0+v_0)$. If we try serial search we see that the size of each of these spaces is $100 * 100 * 4 * 4 * 4 * 2^{18} * 2^{18} = 43.9$ Quadrillion which at 200 iterations per generation will require 219,902,325,−555,200 generations. On our test computer each generation requires 0.00021 h. So we will finish testing all possible serial combinations in approximately 461,794,883,−665 h = 19,241,453,486 days = 52,716,310 years.

Since searching for (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), (E14.61), and (E14.62) is not practical under any approach, we take a giant leap and search for the following.

- (*S24*) **search** regress(v_0*y,v_0,v_1,1/v_1) **where** island(smart,standard,10,25,200) isolate(true)
  onscore(0.0,50,regress(1/f_0(c_0,$v_0$$),$v_1$$/f_0(c_0,$v_0$$),1/ ($v_1$$*f_0(c_0,$v_0$$)))
  where
  island(10,25,200)
  f_0(+, −,rsub)
  c_0(1.0/$w0$)
  onfinal(regress($poly$)))

Clearly there is a big difference between (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), (E14.61), and (E14.62) and (S24), and also the search goal in (S23) contains the term ***y*** which being the dependent variable in the regression. First, the ***isolate(true)*** clause in (S24) keeps any of the champions from the first **where** clause from reaching the final solution list. While there is nothing wrong with using ***y*** during training (***y*** is used repeatedly during fitness calculation while training), allowing ***y*** to appear in the final solution causes problems because ***y*** will not be available during testing. So only solutions containing the features x_0 thru x_{99} are appropriate for final champions.

Second, the ***onscore*** clause erases all champions except the top champion and proceeds to convert the top champion into the form specified in the ***onscore*** goal and **where** clauses. The ***onscore*** clause is triggered when the first search achieves a fitness score of 0.0 or when the number of training generations reaches 50. This initiates a completely new search for *regress(1/f_0(c_0,$v_0$$),$v_1$$/f_0(c_0,$v_0$$),1/($v_1$$* f_0(c_0,$v_0$$)))* which does not contain the inappropriate ***y*** term. Therefore the term ***y*** is used only during training and setup but never in the final solution.

In order to understand how the two cascading searches in (S24) relate to the eight difficult cases (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), (E14.61), and (E14.62), we must observe the following regression equivalence chains in the situation where there is zero noise.

- (*E14.61*) **regress((c$_1$/v$_1$)/(c$_0$+v$_0$))** $->$
- (*E14.61.1*) y=a+b((c$_1$/v$_1$)/(c$_0$+v$_0$))$->$
- (*E14.61.2*) y(c$_0$+v$_0$)=a(c$_0$+v$_0$)+((bc$_1$)/v$_1$) $->$
- (*E14.61.3*) c$_0$y+v$_0$y=a(c$_0$+v$_0$)+((bc$_1$)/v$_1$) $->$
- (*E14.61.4*) c$_0$y =ac$_0$+av$_0$+((bc$_1$)/v$_1$) - v$_0$y $->$
- (*E14.61.5*) y =a+(a/c$_0$)v$_0$+(bc$_1$/c$_0$)/v$_1$ - (1/c$_0$)v$_0$y $->$
- (*E14.61.6*) **regress(v$_0$y,v$_0$,1/v$_1$)** $->$
- (*E14.61.7*) y=a1+w0v$_0$y+w1v$_0$+w2/v$_1$ $->$
- (*E14.61.8*) w0=(1/c$_0$) $->$
- (*E14.61.9*) **c$_0$ =(1/w0)** $->$
- (*E14.61.10*) regress(1/(v$_1$∗((1/w0)+v$_0$))) $->$
- (*E14.61.11*) **regress((c$_1$/v$_1$)/(c$_0$+v$_0$))**

Similar equivalence chains are true for (E14.49), (E14.50), (E14.53), (E14.54), (E14.57), (E14.58), and (E14.62). Taken all together, we see that the answers to the eight bolded cases will be *regress(1/f$_0$(c$_0$,v_0$$),v_1$$/f$_0$(c$_0$,v_0$$),1/(v_1$$∗ f$_0$(c$_0$,v_0$$)))* where f$_0$(+, −,rsub), which is exactly what search island (S24) proposes.

Let's use this actual example, suppose the target formula is *y=1.0+(2.0∗ ((2.8∗x$_2$)/(23.451+x$_4$))*. The first search in (S24) *regress(v$_0$∗y,v$_0$,v$_1$,1/v$_1$)* discovers that the champion *y=y=1−(0.0426421∗(x$_4$∗y))+(0∗x$_4$)+(2.8∗x$_2$)+(0*x$_1$)+(0∗ (1/x$_2$)))* achieves a fitness score of 0.0. This low fitness score triggers the **onscore** clause (otherwise the onscore clause would be triggered by more than 50 generations passing). The **onscore** clause substitutes for the items enclosed in $ sign pairs and searches for the following goal *regress(1/f$_0$(c$_0$,x$_4$),x$_2$/f$_0$(c$_0$,x$_4$),1/(x$_2$∗f$_0$(c$_0$,x$_4$))) where f$_0$(+, −,rsub) c$_0$(23.451)* (**since (1/0.0426421) = 23.451**). The final answer is *y=1.0+(4.8*(x$_2$/(23.451+x$_4$)))* with a final fitness score of 0.0.

The search space size, for island (S24), is 100*100=10,000. At 200 serial iterations per generation, this search will require a maximum of 50 generations. On our test machine, each generation requires 0.0003 h. So the maximum time required for this search island to complete is 0.015 h and is followed immediately by a brief search for the onscore goal.

Most often the evolutionary search finds the correct answer in far less time. For instance, even in the case of this difficult target *y=1.0+(2.0∗((2.8∗x$_2$)/ (23.451+ x$_4$)))*, the evolutionary search normally finds the target in less than a third of the maximum serial time.

16 Accuracy Measurements

Packaging together RQL search commands from (S0) thru (S24), with searches (S16.1 thru S16.24) and (S17.1 thru S17.32) expanded for cloud deployment, we attack the 30 test problems using 81 processor units. As mentioned, each of the problems were trained and tested on 10,000 training examples with 100 features.

Table 1.1 Results demonstrating extreme accuracy

Test	WFFs	Train-NLSE	Test-NLSE	Extreme-NLSE
T01	1 K	0.0000	0.0000	0.0000
T02	5 K	0.0000	0.0000	0.0000
T03	5 K	0.0000	0.0000	0.0000
T04	5 K	0.0000	0.0000	0.0000
T05	6 K	0.0000	0.0000	0.0000
T06	51 M	0.0000	0.0000	0.0000
T07	6 K	0.0000	0.0000	0.0000
T08	2 B	0.0000	0.0000	0.0000
T09	12 M	0.0000	0.0000	0.0000
T10	139 M	0.0000	0.0000	0.0000
T11	32 M	0.0000	0.0000	0.0000
T12	6 K	0.0000	0.0000	0.0000
T13	3 K	0.0000	0.0000	0.0000
T14	17 K	0.0000	0.0000	0.0000
T15	3 M	0.0000	0.0000	0.0000
T16	1 M	0.0000	0.0000	0.0255
T17	12 M	0.0000	0.0000	0.0000
T18	14 M	0.0000	0.0000	0.0000
T19	729 K	0.0000	0.0000	0.0000
T20	22 M	0.0000	0.0000	0.0000
T21	41 M	0.0000	0.0000	0.0000
T22	61 M	0.0000	0.0000	0.0000
T23	32 M	0.0000	0.0000	0.0000
T24	2 K	0.0000	0.0000	0.0000
T25	6 K	0.0000	0.0000	0.0000
T26	436 K	0.0000	0.0000	0.0000
T27	158 K	0.0000	0.0000	0.0000
T28	2 K	0.0000	0.0000	0.0000
T29	3 M	0.0000	0.0000	0.0000
T30	2 M	0.0000	0.0000	0.0000

Note: (1) The number of individuals evaluated before finding a solution is listed in the Well Formed Formulas (WFFs) column. (2) The fitness score of the champion on the training data is listed in the (Train-NLSE) column. (3) The fitness score of the champion on the testing data is listed in the (Test-NLSE) column. (4) The fitness score of the champion on the extreme range data is listed in the (Extreme-NLSE) column

The maximum time to complete a test problem in our cloud environment is 225 h or 9.375 days. The results in Table 1.1 demonstrate extreme accuracy on the 30 test problems.

Notice the extreme search efficiency which Table 1.1 demonstrates. Our assertion is that the extreme accuracy algorithm is getting the same accuracy on $U_2(1)$ and $U_1(3)$ as if each and every single element of those sets were searched serially; and yet we are never evaluating more than a few billion candidates. Notice also the high variance in WFFs evaluated per test problem. This is the result of the random nature

of evolutionary search and how much of the search burden must be carried by the serial search and mathematical treatments.

Obviously *extreme accuracy* is not the same as *absolute accuracy* and is therefore fragile under some conditions. Extreme accuracy will stop at the first estimator which achieves an NLSE of **0.0** on the training data and *hope* that the estimator will achieve an NLSE of **0.0001** or less on the testing data. Yes, an extremely accurate algorithm is guaranteed to find a perfect champion (*estimator training fitness of 0.0*) if there is one to be found; but, this perfect champion may or may not be the estimator which was used to create the testing data. For instance in the target formula $y = 1.0 + (100.0 * sin(x_0)) + (0.001 * square(x_0))$ we notice that the final term $(0.0001 * square(x_0))$ is less significant at low ranges of x_0; but, as the absolute magnitude of x_0 increases, the final term is increasingly significant. And, this does not even cover the many issues with problematic training data ranges and poorly behaved target formulas within those ranges. For instance, creating training data in the range −1,000–1,000 for the target formula $y = 1.0 + exp(x_2 * 34.23)$ runs into many issues where the value of **y** exceeds the range of a 64 bit IEEE real number. So as one can see the concept of *extreme accuracy* is just the beginning of the attempt to conquer the accuracy problem in SR.

In an attempt to further explore the behavior we have labeled *extreme accuracy*, an extreme training matrix of independent variables was filled with random numbers in the range [0,1]. Then an extreme testing matrix of independent variables was filled with random numbers in the range [−1,0]. The champion, which was trained on the [0,1] range, had never seen this data before and had never seen data in the range [−1,0] before. The champion's results against the extreme testing data are shown in the *Extreme-NLSE* column of Table 1.1.

It should be noted that the end user has no knowledge of RQL searches (S0) thru (S24). These searches are applied, behind the veil, when the user submits a test problem. Similarly, the end user had no knowledge of the details of the cloud deployment – nor is it necessary or desirable that the end user have such involvement.

All the extreme algorithm timings and tables of results in this paper have been performed, in a modest cloud deployment. In a modest cloud deployment, searches (S0) thru (S15), (S16.1) thru (S16.24), (S17.1) thru (S17.32), and (S18) thru (S24) are distributed across 81 processor units. On our test machine, if one allows a maximum time to complete of 9.375 days running in this modest cloud configuration, the maximum number of features which can be attempted is 100 features.

The extreme algorithm can also be delivered, in a single thread deployment, on a laptop for scientists who want to run nonlinear regression problems in the background as they perform their normal tasks. In a single thread deployment, searches (S0) thru (S24) are packaged together as a unit and run in a single process on the laptop. On our test machine, if one allows a maximum time to complete of 3.25 days running in the background, the maximum number of features which can be attempted is 25 features. If one allows a maximum time to complete of 12.5 days running in the background, the maximum number of features which can be attempted is 35 features.

The extreme algorithm can also be delivered, as a multi-thread deployment, on a workstation for scientists who want to run nonlinear regression problems in their laboratory. In a multi-thread deployment, searches (S0–S3), (S4–S15), (S16–S17), and (S18–S24) are packaged together as four units. Each unit is run on a single thread on the workstation and assigned to a single core cpu. On our test machine, if one allows a maximum time to complete of 13.02 days running on the workstation, the maximum number of features which can be attempted is 50 features.

The extreme algorithm can also be delivered, on a large cloud deployment, for scientists who want to run very large nonlinear regression problems and who have a large number of computation nodes. In a large cloud deployment, searches (S0) thru (S15), (S16.1) thru (S16.24), and (S18) thru (S24) are distributed across 49 processor units. Searches (S17.1) thru (S17.32) are further broken out, at run time, into $32 \times \mathrm{M}$ separate searches where $\|\mathrm{V}\| = \mathrm{M}$. This is done by expanding each of the searches (S17.1) thru (S17.32) into M separate searches by setting the variable v_0 into all possible M concrete values, as shown in the examples below.

- (*S17.1.1*) **search** regress($(x_0+v_1)*(v_2+v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.1.2*) **search** regress($(x_1+v_1)*(v_2+v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.1.3*) **search** regress($(x_2+v_1)*(v_2+v_3)$) **where** island(smart,standard,10,25,4,000)
- (*S17.1.M*) **search** regress($(x_M+v_1)*(v_2+v_3)$) **where** island(smart,standard,10,25,4,000)

On our test machine, if one allows a maximum time to complete of 11.719 days running in this large cloud configuration, the maximum number of features which can be attempted is 500 features, and one would require 16,049=(49+(32*500)) computation nodes. Furthermore, at 500 features, the size of the search space for which we are asserting extreme accuracy is larger than $15^7*(500+2^{18})^8 = 3.86\mathrm{E}{+}051$.

17 Conclusion

In a previous paper Korns (2011), significant accuracy issues were identified for state of the art SR systems. It is now obvious that these SR accuracy issues are due primarily to the poor surface conditions of specific subsets of the problem space. For instance, if the problem space is exceedingly choppy with little monotonicity or flat with the exception of a single point with fitness advantage, then no amount of fiddling with evolutionary parameters will address the core issue.

In this paper we lay the ground work for an enhanced algorithmic approach to SR which achieves a level of extreme accuracy. This enhanced algorithm contains a search language and an *informal argument*, suggesting a priori, that extreme accuracy will be achieved on any single isolated problem within a broad class of basic SR problems. Furthermore, maximum resource allocations and maximum timings are given for achieving extreme accuracy.

The new extreme accuracy algorithm introduces a hybrid view of SR in which advanced evolutionary methods are deployed in the extremely large spaces where

serial search is impractical, and in which the intractable smaller spaces are first identified and then attacked either serially or with mathematical treatments. All academics and SR researchers are heartily invited into this newly opened *playground*, as a plethora of intellectual work awaits. Increasing SR's demonstrable range of extreme accuracy will require that new intractable subspaces be identified and that new mathematical treatments be devised.

Finally, to the extent that the reasoning in this *informal argument*, of extreme accuracy, gain academic and commercial acceptance, a climate of *belief* in SR can be created wherein SR is increasingly seen as a "**must have**" tool in the scientific arsenal.

References

Hornby GS (2006) ALPS: the age-layered population structure for reducing the problem of premature convergence. In: Keijzer M, Cattolico M, Arnold D, Babovic V, Blum C, Bosman P, Butz MV, Coello Coello C, Dasgupta D, Ficici SG, Foster J, Hernandez-Aguirre A, Hornby G, Lipson H, McMinn P, Moore J, Raidl G, Rothlauf F, Ryan C, Thierens D (eds) GECCO 2006: proceedings of the 8th annual conference on genetic and evolutionary computation, Seattle, vol 1. ACM, pp 815–822. doi:10.1145/1143997.1144142, http://www.cs.bham.ac.uk/~wbl/biblio/gecco2006/docs/p815.pdf

Korns MF (2010) Abstract expression grammar symbolic regression. In: Riolo R, McConaghy T, Vladislavleva E (eds) Genetic programming theory and practice VIII, Ann Arbor. Genetic and evolutionary computation, vol 8. Springer, chap 7, pp 109–128. http://www.springer.com/computer/ai/book/978-1-4419-7746-5

Korns MF (2011) Accuracy in symbolic regression. In: Riolo R, Vladislavleva E, Moore JH (eds) Genetic programming theory and practice IX, Ann Arbor. Genetic and evolutionary computation. Springer, chap 8, pp 129–151. doi:10.1007/978-1-4614-1770-5-8

Korns MF (2012) A baseline symbolic regression algorithm. In: Genetic programming theory and practice X. Springer

Kotanchek M, Smits G, Vladislavleva E (2007) Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models. In: Riolo RL, Soule T, Worzel B (eds) Genetic programming theory and practice V, Ann Arbor. Genetic and evolutionary computation. Springer, chap 12, pp 201–220. doi:10.1007/978-0-387-76308-8-12

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

McConaghy T (2011) FFX: fast, scalable, deterministic symbolic regression technology. In: Riolo R, Vladislavleva E, Moore JH (eds) Genetic programming theory and practice IX, Ann Arbor. Genetic and evolutionary computation. Springer, chap 13, pp 235–260. doi:10.1007/978-1-4614-1770-5-13

Nelder J, Wedderburn R (1972) Generalized linear models. J R Stat Soc Ser A 135:370–384

Schmidt M, Lipson H (2010) Age-fitness pareto optimization. In: Riolo R, McConaghy T, Vladislavleva E (eds) Genetic programming theory and practice VIII, Ann Arbor. Genetic and evolutionary computation, vol 8. Springer, chap 8, pp 129–146. http://www.springer.com/computer/ai/book/978-1-4419-7746-5

Smits G, Kotanchek M (2004) Pareto-front exploitation in symbolic regression. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice II, Ann Arbor. Springer, chap 17, pp 283–299. doi:10.1007/0-387-23254-0-17

Chapter 2
Exploring *Interestingness* in a Computational Evolution System for the Genome-Wide Genetic Analysis of Alzheimer's Disease

Jason H. Moore, Douglas P. Hill, Andrew Saykin, and Li Shen

Abstract Susceptibility to Alzheimer's disease is likely due to complex interaction among many genetic and environmental factors. Identifying complex genetic effects in large data sets will require computational methods that extend beyond what parametric statistical methods such as logistic regression can provide. We have previously introduced a computational evolution system (CES) that uses genetic programming (GP) to represent genetic models of disease and to search for optimal models in a rugged fitness landscape that is effectively infinite in size. The CES approach differs from other GP approaches in that it is able to learn how to solve the problem by generating its own operators. A key feature is the ability for the operators to use expert knowledge to guide the stochastic search. We have previously shown that CES is able to discover nonlinear genetic models of disease susceptibility in both simulated and real data. The goal of the present study was to introduce a measure of *interestingness* into the modeling process. Here, we define interestingness as a measure of non-additive gene-gene interactions. That is, we are more interested in those CES models that include attributes that exhibit synergistic effects on disease risk. To implement this new feature we first pre-processed the data to measure all pairwise gene-gene interaction effects using entropy-based methods. We then provided these pre-computed measures to CES as expert knowledge and as one of three fitness criteria in three-dimensional Pareto optimization. We applied this new CES algorithm to an Alzheimer's disease data set with approximately 520,000 genetic attributes. We show that this approach discovers more interesting models with the added benefit of improving classification accuracy. This study demonstrates the applicability of CES to genome-wide genetic analysis using expert knowledge derived from measures of interestingness.

J.H. Moore (✉) • D.P. Hill • A. Saykin • L. Shen
The Geisel School of Medicine at Dartmouth, One Medical Center Drive, HB7937, Lebanon, NH 03756, USA
e-mail: Jason.H.Moore@Dartmouth.edu; douglas.hill@Dartmouth.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_2,

Keywords Computational evolution • Genetic epidemiology • Epistasis • Gene-gene interactions

1 Introduction

The genetic analysis of Alzheimer's disease has had mixed results despite the availability of genome-wide measures of genetic variation, reviewed by Bertram and Tanzi (2012). The single best genetic risk factor is variation in the Apolipoprotein E (ApoE) gene with odds ratios (OR) between 3 and 20 depending on the particular combination of alleles. Aside from this strong genetic risk factor, approximately 9 others have been found with much smaller OR between 0.8 and 1.3. Some of the unexplained heritability of this common disease is likely due to complex gene-gene interactions or epistasis. This type of genetic effect cannot be predicted by the effects of single genetic variants and has been largely ignored by genetic and epidemiological studies (Moore and Williams 2009). Several recent studies have highlighted the importance of gene-gene interactions in Alzheimer's disease and thus provide a foundation for further investigation using data mining and machine learning methods that are ideally suited to detecting nonlinear effects of attribute combinations (Combarros et al. 2009; Lehmann et al. 2012; Bullock et al. 2013). Although promising, these studies only explored pairwise gene-gene interactions. The search for higher-order gene-gene interactions in an effectively infinite search space is a significant statistical and computational problem (Moore et al. 2010).

The overarching goal of this study is to explore data mining and machine learning alternatives to parametric statistical methods for the genetic analysis of complex human diseases. In particular, we are interested in computational intelligence methods that are able to learn how to solve genetic analysis problems as a human would. This general approach involves pre-processing the data to identify useful information, implementing a machine learning algorithm that is able to model nonlinear effects, implementing a stochastic search algorithm that is able to exploit expert knowledge and implementing post-processing methods that are able to enhance statistical and biological interpretation of the results. We have previously developed computational intelligence methods for genetic analysis that are based on genetic programming (GP). Genetic programming is an automated computational discovery tool that is inspired by Darwinian evolution by natural selection (Koza 1992; Banzhaf et al. 1998). The goal of GP is to 'evolve' computer programs to solve complex problems. This is accomplished by first generating or initializing a population of random computer programs that are composed of the basic building blocks needed to solve or approximate a solution to the problem. Genetic programming and its many variations have been applied successfully in a wide range of different problem domains including bioinformatics (Fogel and Corne 2003) and genetic analysis (Moore et al. 2010). GP is an attractive approach to the genetic analysis problem because it is inherently flexible, stochastic, parallel and easily adapted to exploit expert knowledge. The goal of the present study was to

build on a GP-based computational evolution strategy (CES) for genetic analysis. We introduce here a measure of interestingness into the modeling process. Here, we define interestingness as a measure of nonlinear gene-gene interactions. That is, we are more interested in those CES models that include attributes that exhibit synergistic effects on disease. To implement this new feature we first pre-processed the data to measure all pairwise gene-gene interaction effects using entropy-based methods. We then provided a small subset of these pre-computed measures to CES as expert knowledge and as one of the fitness criteria in three-dimensional Pareto optimization. We applied this new CES algorithm to an Alzheimer's disease data set with approximately 520,000 genetic attributes.

2 Computational Evolution

It has been suggested that the incorporation of greater biological realism into GP may improve its ability to solve complex, real-world problems. Specifically, Banzhaf et al. (2006) have called for the development of open-ended computational evolution systems (CES) that attempt to emulate, rather than ignore, the complexities of biotic systems. With this in mind, we have recently developed a hierarchical, spatially-explicit CES that allows for the evolution of arbitrarily complex solutions and solution operators, and includes population memory via archives, feedback loops between archives and solutions, and environmental sensing (Moore et al. 2008; Moore and Williams 2009; Greene et al. 2009a,b; Payne et al. 2010). Analyses of this system have demonstrated its ability to identify complex disease-causing genetic architectures in simulated data, and to recognize and exploit useful sources of expert knowledge. Specifically, we have shown that statistical expert knowledge, in the form of ReliefF scores (Moore and White 2007), can be incorporated via environmental sensing (Greene et al. 2009b) and population initialization (Payne et al. 2010) to improve system performance. In addition, we recently showed that biological expert knowledge in the form of protein-protein interactions could be used to guide CES toward valid gene-gene interaction models (Pattin et al. 2010). We also showed how visualization of CES results could improve the modeling process (Moore et al. 2011). More recently, we have demonstrated how two-dimensional Pareto optimization can be used to help guide the search (Moore et al. 2013). Here, we introduce a measure of *interestingness* and show how it can be incorporated as expert knowledge and into a three-way Pareto optimization. We briefly introduce both of these in turn below.

Pre-processing Measure of Interestingness

A central goal of this study is to identify new genetic models of Alzheimer's disease that have not been revealed in prior studies. We are particularly interested in those

models that are comprised of nonlinear gene-gene interactions that are not predicted from the independent effects of single genetic variants. We used here a measure of gene-gene interaction introduced to the genetics community by Moore et al. (2006) that is based on information theory or entropy. Specifically, we measure *interaction information* as the information gain due to the synergistic effects of two genetic variants after the individual effects have been subtracted out. This provides a measure of interestingness that can be used to help guide the proposed CES methodology toward models that are new and different from what has been previously discovered.

Pareto Optimization

A common approach for addressing overfitting in data mining and machine learning is to use cross-validation as an estimate of the generalizability of a model. Unfortunately, implementation of cross-validation methods in conjunction with stochastic methods such as GP can be complex given these algorithms are likely to find different models in each division of the data. Pareto optimization (reviewed by Lamont and VanVeldhuizen (2002)) offers a viable alternative and has been shown to be quite effective in the context of GP (Smits and Kotanchek 2004). Pareto optimization balances several different model objectives that are each treated equally. We have previously used classification accuracy and model size as our two objectives (Moore et al. 2013). Here, we add a third dimension defined by the interaction information measure of interestingness. For a given GP population, models for which there are no better as measured by accuracy, model size and interestingness are selected. This subset of Pareto-optimal models is referred to as the Pareto front. The goal of the present study was to introduce the use of interestingness as a third dimension in Pareto optimization of CES. This allows CES to explore models that might have good interestingness but poor accuracy or error.

Post-processing of CES Models

We have previously demonstrated that post-processing CES results can improve model discovery (Moore et al. 2011, 2013). In other words, there is value in analyzing the results of a CES run and extracting knowledge from that analysis that can be used to improve interpretation of CES models. Here, we use network analysis and visualization to help interpret the genetic effects in CES models (Hu et al. 2013).

3 Methods

In this section, we first present a summary of our computational evolution system (CES) for open-ended genetic analysis of complex human diseases. We then discuss our implementation of the pre-processing methods, Pareto optimization and post-processing extensions and their application to Alzheimer's disease.

Computational Evolution System

In Fig. 2.1, we provide a graphical overview of CES, which is both hierarchically organized and spatially explicit. The bottom level of the hierarchy consists of a lattice of solutions (Fig. 2.1D), which compete with one another within

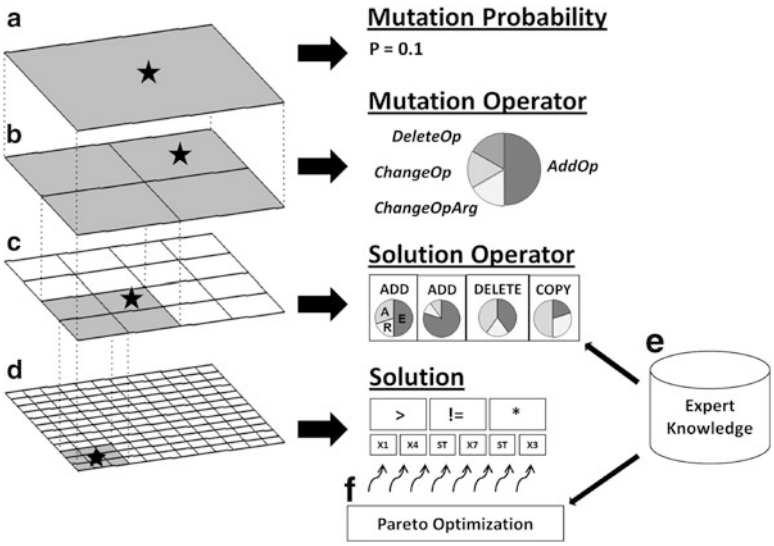


Fig. 2.1 Visual overview of our computational evolution system for discovering symbolic discriminant functions that differentiate disease subjects from healthy subjects using information about single nucleotide polymorphisms (SNPs). The hierarchical structure is shown on the *left* while some specific examples at each level are shown in the *middle*. At the lowest level (*D*) is a grid of solutions. Each solution consists of a list of functions and their arguments (e.g. X1 is an attribute or SNP) that are evaluated using a stack (denoted by ST in the solution). The next level up (*C*) is a grid of solution operators that each consists of some combination of the ADD, DELETE and COPY functions each with their respective set of probabilities that define whether attributes are added, deleted or copied randomly, using an attribute archive (memory) or just randomly. In this implementation of CES, we use pre-processed expert knowledge (*E*) with Pareto optimization (*F*) to help reduce overfitting. The top two levels of the hierarchy (*A* and *B*) exist to generate variability in the operators that modify the solutions. This system allows operators of arbitrary complexity to modify solutions. A 12 × 12 grid is shown here as an example. A 36 × 36 grid was used in the present study

spatially-localized, overlapping neighborhoods. The second layer of the hierarchy contains a lattice of arbitrarily complex solution operators (Fig. 2.1C), which operate on the solutions in the lower layer. The third layer of the hierarchy contains a lattice of mutation operators (Fig. 2.1B), which modify the solution operators in the second layer, and the highest layer of the hierarchy governs the rate at which the mutation operators are modified (Fig. 2.1A). CES includes a source of expert knowledge (Fig. 2.1E) that can be used to with the solution operators and as part of Pareto optimization (Fig. 2.1F). CES also possesses an attribute archive, which stores the frequencies with which attributes are used. The solution operators can then exploit these data to bias the construction of solutions toward frequently utilized attributes. We did not use the attribute archive in the present study.

Solution Representation, Fitness Evaluation, Selection, and Pareto Optimization

Each solution represents a classifier, which takes a set of SNPs as input and produces an output that can be used to assign diseased or healthy status. These solutions are represented as stacks, where each element in the stack consists of a function and two operands (Fig. 2.1). The function set contains $+, -, *, /, \%, <, <=, >, >=, ==, !=$, where % denotes protected modulus. Operands are either SNPs, constants, or the output of another element in the stack.

Each solution produces a discrete output S_i when applied to an individual i. Symbolic discriminant analysis (Moore et al. 2002) is then used to map this output to a classification rule, as follows. The solution is independently applied to the set of diseased and healthy individuals to obtain two separate distributions of outputs, $S^{diseased}$ and $S^{healthy}$, respectively. A classification threshold S_0 is then calculated as the arithmetic mean of the medians of these two distributions. Each of the possible relationships between S_0 and S_i $(<, <=, >=, >)$ is tested across all individuals, and the one with best overall accuracy is chosen to classify whether individuals are healthy or diseased.

Solution accuracy is assessed through a comparison of predicted and actual clinical endpoints. Specifically, the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) are used to calculate accuracy as:

$$A = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

Solution length can be assessed in several ways. The number of elements in the classifier is the most straightforward. Since many solutions leave results on the stack that do not contribute to the classification, we can define "number of relevant elements" as only those contributing to the result. Finally we can count the number of unique SNPs in the relevant elements. We have chosen to use this as the measure of length in the present study as it makes the resulting solutions easier to analyze.

The population is organized on a two-dimensional lattice with periodic boundary conditions. Each solution occupies a single lattice site, and competes with the solutions occupying the eight spatially adjacent sites. In all previous CES implementations election has been both synchronous and elitist, such that the solution of highest fitness within a given neighborhood was always selected to repopulate the focal site of that neighborhood. In the present study, selection proceeds in two stages modeled after Pareto domination tournaments and fitness sharing described by Horn et al. (1994). We used classification accuracy, number of attributes in the model and interaction information as the axes in the Pareto optimization. Here, the sum of the interaction information for all pairs of attributes in a model is the measure of interestingness. First all dominated solutions and solutions evaluating to a constant are removed from competition. A solution is dominated if there exists any solution of lesser or equal length with lesser classification error, or lesser length and equal error. If no solution survives this stage, one of the nine is chosen with equal probability. If more than one solution survives, each is assigned a probability and a roulette wheel selection is made. Higher selection probability is assigned to a solution if there are relatively fewer solutions of that length in the lattice, in order to prevent convergence on solutions of a single length. For the present results we made the probability inversely proportional to the square of the number of existing solutions of the same length. Reproduction is either sexual or asexual, as dictated by the evolvable solution operators that reside in the next layer of the hierarchy.

The population is initialized by randomly generating solutions with 1–15 elements subject to the constraint that they produce a valid output that is not constant for all input. The functions are selected at random with uniform probability from the function set.

Solution Operators

CES allows for the evolution of arbitrarily complex variation operators used to modify solutions. This is achieved by initializing the solution operator lattice (Fig. 2.1C) with a set of basic building blocks which can be recombined in any way to form composite operators throughout the execution of the program. The action of some of these operators is influenced by any of several types of expert knowledge (EK) that CES recognizes. In this study we have used one type of EK, Association EK. Association EK is used to help CES to more quickly find solutions using specific combinations of attributes or SNPs. Here, we used a measure of interaction information as the expert knowledge. Adding and altering attributes is based on a lookup table that is constructed from the strength of interactions between pairs of attributes. Because 521,028 attributes have over 2.7×2^{11} pairs, it was impossible to pre-compute and store all pairs in the memory available. We pre-computed all pairs but stored only the 93,606 most strongly interacting pairs, a tiny fraction of the total. The following are the building blocks and the way they are influenced by Association EK.

1. ADD: Inserts a randomly generated element into the solution at a randomly selected position. If the element immediately before this position is one of the pairs of strongly interacting elements, preferentially chooses one of these interacting elements.
2. ALTER: Modifies either the function or an argument of a randomly selected element. If it chooses an attribute as the new argument, that attribute is selected as above in ADD.
3. COPY: Within a randomly selected neighboring solution, randomly selects an element and inserts it into a randomly selected position in the focal solution.
4. DELETE: Removes an element from a randomly selected position or a position that is probabilistically selected by the lookup table.
5. REPLACE: Within a randomly selected neighboring solution, randomly selects a source position. In the focal solution, randomly selects a destination position. Replaces everything between the destination position and the end (root) of the focal solution with everything between the source position and the end of the source solution.

The solution operators reside on a periodic, toroidal lattice of coarser granularity than the solution lattice (Fig. 2.1C). Each site is occupied by a single solution operator, which is assigned to operate on 3×3 sub-grid of solutions. These operators compete with one another in a manner similar to the competition among solutions, and their selection probability is determined by the fitness changes they evoke in the solutions they control. For this purpose we assigned the fitness of a solution as we have done in previous studies: balanced accuracy with a small penalty for number of elements. We did not adapt a Pareto tournament to the selection of solution operators.

Mutation Operators

The third level of the hierarchy contains the mutation operators, which are used to modify the solution operators (Fig. 2.1B). These reside on a toroidal lattice of even coarser granularity, and are assigned to modify a subset of the solution operators below. The mutation operators are represented as three-element vectors, where each element corresponds to the probability with which a specific mutation operator is used. These three mutation operators work as follows. The first (DeleteOp) deletes an element of a solution operator; the second (AddOp) adds an element to a solution operator, and the third (ChangeOp) mutates an existing element in a solution operator. The probabilities with which these mutation operators are used undergo mutation at a rate specified in the highest level of the hierarchy (Fig. 2.1A).

Alzheimer's Disease Data

The data used in this study came from the Alzheimer's Disease Neuroimaging Initiative (ADNI) that began on October 1, 2004. The study takes functional MRIs every 6–12 month of patients in three categories: those who are neuro-typical, those with mild cognitive impairment, and those with Alzheimer's disease. A total of 521,028 single-nucleotide polymorphisms (SNPs) were measured across the human genome in a total of 740 subjects. Here, we used neuro-typical patients as the control subjects and those with mild cognitive impairment or Alzheimer's disease as the case subjects creating a binary class or outcome. The goal of the modeling exercise is to identify the optimal subset of SNPs along with the optimal mathematical model that is predictive of the binary class.

Pre-processing, Experimental Design and Post-processing

The goal of this study was to apply CES to the genetic analysis of Alzheimer's disease. We first pre-processed the data by estimating the interaction information for all pairs of SNPs as described by Moore et al. (2006). We considered pairs of SNPs that have higher interaction information more interesting. This pre-processed interestingness measure was used as expert knowledge (Attribute EK) in the CES solution modifiers and as an additional axis in a three-way Pareto optimization.

Each CES run was conducted with a 36×36 grid of solutions for 2,000 generations. CES was implemented in a hierarchical framework inspired by the age-layered population structure algorithm or ALPS (Hornby 2006). Here, we implemented a depth six binary tree where each node represents a CES run with the leaves of the tree representing the initial runs. Each higher node run is initialized with Pareto optimal solutions from the lower nodes. This was performed 10 times with different sets of random seeds. This analysis was repeated with and without expert knowledge and with two or three-way Pareto optimization where the two-way Pareto had only classification accuracy and model size as axes. The three-way Pareto optimization included interaction information as the measure of interestingness. Thus, we had 4 treatment groups each with 10 runs. We used two-way analysis of variance (ANOVA) to compare the mean classification accuracy and interaction information among the two expert knowledge groups and the two Pareto groups. We also considered the interaction effect of both expert knowledge and Pareto. All results were considered statistically significant at the 0.05 significance level.

We reported the best models discovered from each set of runs under each of the four experimental conditions. We used the visualization of statistical epistasis networks (ViSEN) method developed by Hu et al. (2013) to visualize the two-way and three-way gene-gene interactions among SNPs in the best CES models. This allows us to interpret the nature of the genetic effects in a visual manner.

4 Results

Figure 2.2 summarizes the average classification error (1-accuracy, y-axis) of the most accurate models, model size measured by number of attributes (x-axis) and the interaction information or interestingness summed across all pairs of SNPs in a model (size of circle). The results are shown according to whether two-way or three-way Pareto optimization was used with or without expert knowledge. As expected, three-way Pareto with expert knowledge showed the highest interaction information and thus the highest interestingness. An unexpected result was that these models also had the lowest average classification error. Thus, the most interesting models were also the ones that classified disease best. The ANOVA results confirm this. We found that two-way vs. three-way Pareto had a significant effect on both classification error ($p = 0.006$) and interaction information ($p < 0.001$). We also found that whether we included expert knowledge had a significant effect on classification error ($p < 0.001$) and interaction information ($p < 0.001$). In addition, Pareto and expert knowledge had a strong interaction effect on classification error ($p < 0.001$) and interaction information ($p < 0.001$).

Figure 2.3 illustrates the overall best model discovered by CES. This model had a classification accuracy of 0.738 (error = 0.262) and consisted of 7 attributes or SNPs. We selected this model as a compromise between model size, accuracy and interestingness. It was discovered by CES run with the three-way Pareto

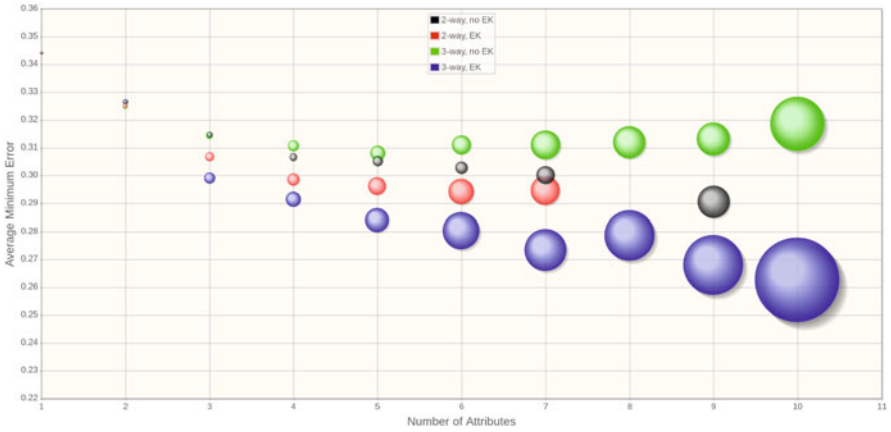


Fig. 2.2 Visual summary of the CES results. Shown are the average interaction information scores (*size of circle*) among the best models on the Pareto front for the 10 CES runs. Results are shown by the average minimum error (1-accuracy – y-axis) and the number of attributes in the model (x-axis). Results are also shown according to the CES method used. *Black* indicates two-way Pareto optimization with no expert knowledge from pre-processing the data for interaction information scores. *Red* indicates two-way Pareto optimization with expert knowledge. *Green* indicates three-way Pareto optimization with no expert knowledge. *Blue* indicates three-way Pareto optimization with expert knowledge. Note that three-way Pareto optimization with expert knowledge achieves the lowest error across model sizes and has the highest interestingness

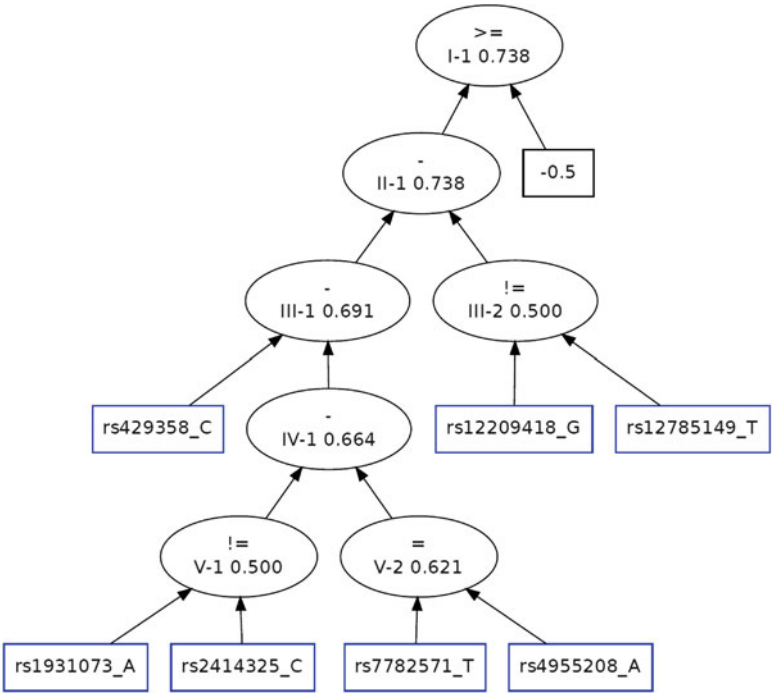


Fig. 2.3 The overall best Pareto-optimal model discovered by CES using three-way Pareto optimization and expert knowledge. The model includes attributes (*rectangles*), constants (*squares*) and mathematical functions or nodes (*ovals*). Each model outputs a discriminant score for each subject in the data set. These scores are then used for classification in a discriminant analysis. The numbers shown within each oval are the classification accuracies at each level in the tree

optimization and with using expert knowledge to guide the search. Figure 2.4 shows a statistical interpretation of the model. Note the large independent effect of SNP rs429358 which by itself has an accuracy of about 0.656. Each of the other six SNPs in the model has weak independent effects and strong pairwise interactions with at least one other SNP. Together they improve the accuracy of the classifier from 0.656 for a model with just the large independent effect to 0.738 for the collection of all seven attributes or SNPs.

5 Summary and Discussion

Alzheimer's disease is likely the result of complex interactions among many genetic and environmental factors. We have demonstrated here how a computational evolution system (CES) can be used to identify new models of disease susceptibility in genome-wide genetic studies with hundreds of thousands of attributes. Despite the size and complexity of the search space, CES was able to find models with high accuracy.

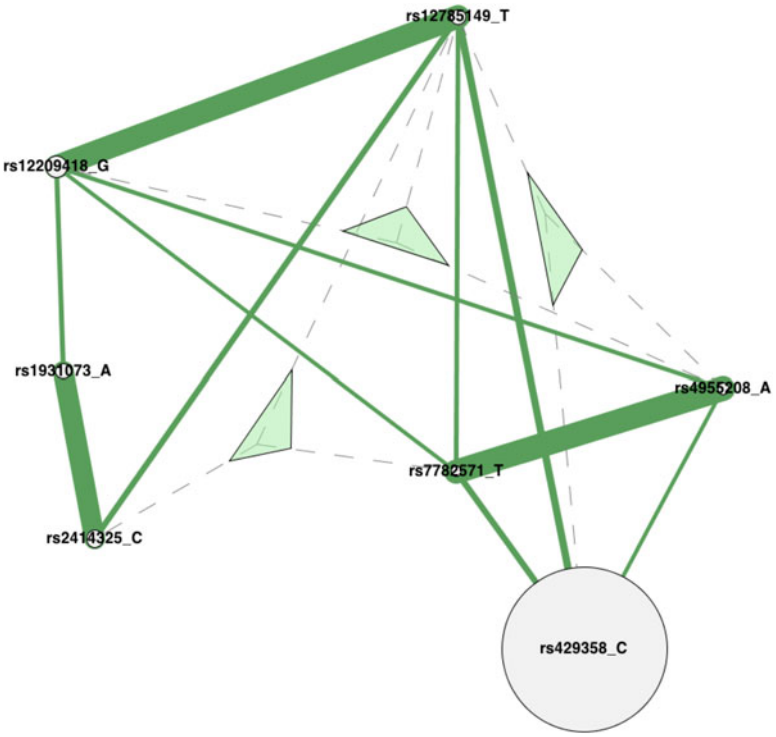


Fig. 2.4 Visualization of the statistical interaction network among the attributes in the overall best CES model. The size of the circle is proportional to the independent effects of each genetic variant on disease risk. The lines connecting each node are proportional to the interaction information for that pair of variants. The *triangles* represent the three-way information gain. Note that the genetic variants with the strongest pairwise interactions are children of the same functions in the CES classification model

The overall best model discovered consisted of seven attributes or SNPs achieving an accuracy of 0.738. The SNP with the large independent effect (rs429358) is located in the ApoE gene that is a known strong risk factor for Alzheimer disease. CES was able to improve on this local minimum by adding six additional SNPs to the model that each is part of a relatively strong gene-gene interaction pair. Interestingly, each pair of strong gene-gene interactions was co-located in the model as children of the same function. SNP rs1931073 is in an intergenic region near the PPAP2B gene that is involved with cell adhesion and cell-cell interactions. SNP rs2414325 is in a gene called UNC13C for which not much is known about its function. SNP rs7782571 is near the ISPD gene that is known to be mutated in rare diseases such as Walker-Warburg syndrome that is known to have brain anomalies. SNP rs4955208 is in the OSBPL10 gene that codes for an intracellular lipid receptor. SNP rs12209418 is in the PKIB gene, a protein kinase inhibitor, and is associated with neuronitis or inflammation of the neurons.

SNP rs12785149 is in the FAM107B gene whose function is not well known. Of these seven genes, only ApoE is present in the Alzgene database (Bertram et al. 2012) that provides an unbiased catalog of known genetic risk factors for Alzheimer's disease. The other six could represent new and novel discoveries. Based on their biology, it is plausible that all of them could be related to the disease process. For example, OSBPL10 is involved with lipid metabolism which is a known component of Alzheimer's disease pathobiology. At this point they remain novel hypotheses that will need to be tested in other data.

We introduced here the concept of interestingness defined as the degree of gene-gene interaction in a given CES model. Interestingness was introduced into CES in two different ways. First, pre-computed pairwise interaction information scores were used as expert knowledge during the model building process. Second, we used interaction information as an additional axis in a three-way Pareto optimization that also included model error and size. This allowed CES to explore models with high interestingness but low accuracy thus promoting diversity. Interestingness has been explored previously for use with data mining (see survey by Geng and Hamilton (2006)). Geng and Hamilton (2006) review nine specific criteria for determining whether a model or result is interesting. The first is conciseness or parsimony. The second is coverage (i.e. applies to a large subset of the data). The third is reliability that is measured by the accuracy or error of a classifier. The fourth is peculiarity that measures how far away a finding is from others. The fifth is diversity that measures how different the elements of a model are. The sixth is novelty (i.e. the result is new). The seventh is *surprisingness* that measures how unexpected the result is based on prior knowledge. The eighth is utility that measures how useful the result is. The final criterion is actionability that measures how applicable a result is to a particular domain. Each of these criteria can be grouped into objective and subjective categories. For example, conciseness, coverage, reliability peculiarity and diversity are all objective measures because they can be computed using an algorithm or mathematical function. On the other hand, novelty, *surprisingness*, utility and actionability are all subjective and dependent on the experience and knowledge of the particular domain expert. In this study we used interaction information as a pre-processed measure of interestingness that could be used to guide the CES. This is an objective measure because we are computing a specific measure from the data. However, it can also be seen as a subjective measure because some in the field do not think gene-gene interactions are important.

Human genetics research has been focused almost exclusively on detecting single attribute effects on disease risk that completely ignore the complexity of the genotype-phenotype relationship. Our primary objective here was to further develop a computational evolution system or CES for the discovery of new and novel genetic associations that embrace the complexity of the problem. We were encouraged by the results of this study and think measures of interestingness have a very important role to play as we attempt to bring expert knowledge back into the modeling process. Our future studies will further explore the role of interestingness for improving computational intelligence modeling strategies in this domain.

Acknowledgements This work was supported by NIH grants LM011360, LM009012, LM010098 and AI59694. We would like to thank the participants of present and past Genetic Programming Theory and Practice Workshops (GPTP) for their stimulating feedback and discussion that helped formulate some of the ideas in this paper.

References

Banzhaf W, Francone FD, Keller RE, Nordin P (1998) Genetic programming: an introduction on the automatic evolution of computer programs and its applications. Morgan Kaufmann, San Francisco

Banzhaf W, Beslon G, Christensen S, Foster J, Képès F, Lefort V, Miller J, Radman M, Ramsden J (2006) From artificial evolution to computational evolution: a research agenda. Nat Rev Genet 7:729–735

Bertram L, Tanzi RE (2012) The genetics of Alzheimer's disease. Prog Mol Biol Transl Sci 107:79–100. doi:10.1016/B978-0-12-385883-2.00008-4

Bullock JM, Medway C, Cortina-Borja M, Turton JC, Prince JA, Ibrahim-Verbaas CA, Schuur M, Breteler MM, van Duijn CM, Kehoe PG, Barber R, Coto E, Alvarez V, Deloukas P, Hammond N, Combarros O, Mateo I, Warden DR, Lehmann MG, Belbin O, Brown K, Wilcock GK, Heun R, Kolsch H, Smith AD, Lehmann DJ, Morgan K (2013) Discovery by the epistasis project of an epistatic interaction between the GSTM3 gene and the HHEX/IDE/KIF11 locus in the risk of Alzheimer's disease. Neurobiol Aging 34(4):1309.e1–1309.e7. doi:10.1016/j.neurobiolaging.2012.08.010

Combarros O, van Duijn CM, Hammond N, Belbin O, Arias-Vasquez A, Cortina-Borja M, Lehmann MG, Aulchenko YS, Schuur M, Kolsch H, Heun R, Wilcock GK, Brown K, Kehoe PG, Harrison R, Coto E, Alvarez V, Deloukas P, Mateo I, Gwilliam R, Morgan K, Warden DR, Smith AD, Lehmann DJ (2009) Replication by the epistasis project of the interaction between the genes for IL-6 and IL-10 in the risk of Alzheimer's disease. J Neuroinflammation 6:22. doi:10.1186/1742-2094-6-22

Fogel GB, Corne DW (eds) (2003) Evolutionary computation in bioinformatics. Morgan Kaufmann, San Francisco

Geng L, Hamilton HJ (2006) Interestingness measures for data mining: a survey. ACM Comput Surv 38(3). doi:10.1145/1132960.1132963, http://doi.acm.org/10.1145/1132960.1132963

Greene CS, Hill DP, Moore JH (2009a) Environmental noise improves epistasis models of genetic data discovered using a computational evolution system. In: Proceedings of the 11th annual conference on genetic and evolutionary computation, GECCO'09, Montreal. ACM, New York, pp 1785–1786. doi:10.1145/1569901.1570160, http://doi.acm.org/10.1145/1569901.1570160

Greene CS, Hill DP, Moore JH (2009b) Environmental sensing of expert knowledge in a computational evolution system for complex problem solving in human genetics. In: Riolo RL, O'Reilly UM, McConaghy T (eds) Genetic programming theory and practice VII. Genetic and evolutionary computation. Springer, Ann Arbor, chap 2, pp 19–36

Horn J, Nafpliotis N, Goldberg DE (1994) A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence, Orlando, vol 1, pp 82–87. doi:10.1109/ICEC.1994.350037, http://dx.doi.org/10.1109/ICEC.1994.350037

Hornby GS (2006) ALPS: the age-layered population structure for reducing the problem of premature convergence. In: Proceedings of the 8th annual conference on genetic and evolutionary computation, GECCO'06, Seattle. ACM, New York, pp 815–822. doi:10.1145/1143997.1144142, http://doi.acm.org/10.1145/1143997.1144142

Hu T, Chen Y, Kiralis JW, Moore JH (2013) ViSEN: methodology and software for visualization of statistical epistasis networks. Genet Epidemiol 37(3):283–285. doi:10.1002/gepi.21718

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection (complex adaptive systems), 1st edn. A Bradford Book. MIT Press, London. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262111705

Lamont GB, VanVeldhuizen DA (2002) Evolutionary algorithms for solving multi-objective problems. Kluwer Academic, Norwell

Lehmann DJ, Schuur M, Warden DR, Hammond N, Belbin O, Kolsch H, Lehmann MG, Wilcock GK, Brown K, Kehoe PG, Morris CM, Barker R, Coto E, Alvarez V, Deloukas P, Mateo I, Gwilliam R, Combarros O, Arias-Vasquez A, Aulchenko YS, Ikram MA, Breteler MM, van Duijn CM, Oulhaj A, Heun R, Cortina-Borja M, Morgan K, Robson K, Smith AD (2012) Transferrin and HFE genes interact in Alzheimer's disease risk: the epistasis project. Neurobiol Aging 33(1):202.e1–202.e13. doi:10.1016/j.neurobiolaging.2010.07.018

Moore JH, White BC (2007) Tuning ReliefF for genome-wide genetic analysis. In: Proceedings of the 5th European conference on evolutionary computation, machine learning and data mining in bioinformatics, EvoBIO'07, Valencia. Springer, Berlin/Heidelberg, pp 166–175. http://dl.acm.org/citation.cfm?id=1761486.1761502

Moore JH, Williams SM (2009) Epistasis and its implications for personal genetics. Am J Hum Genet 85(3):309–320. doi:10.1016/j.ajhg.2009.08.006, http://dx.doi.org/10.1016/j.ajhg.2009.08.006

Moore JH, Parker JS, Olsen NJ, Aune TM (2002) Symbolic discriminant analysis of microarray data in autoimmune disease. Genet Epidemiol 23(1):57–69

Moore JH, Gilbert JC, Tsai CT, Chiang FT, Holden T, Barney N, White BC (2006) A flexible computational framework for detecting, characterizing, and interpreting statistical patterns of epistasis in genetic studies of human disease susceptibility. J Theor Biol 241(2):252–261. doi:10.1016/j.jtbi.2005.11.036, http://dx.doi.org/10.1016/j.jtbi.2005.11.036

Moore JH, Andrews PC, Barney N, White BC (2008) Development and evaluation of an open-ended computational evolution system for the genetic analysis of susceptibility to common human diseases. In: Marchiori E, Moore JH (eds) EvoBIO'08, Naples. Lecture notes in computer science, vol 4973. Springer, pp 129–140

Moore JH, Asselbergs FW, Williams SM (2010) Bioinformatics challenges for genome-wide association studies. Bioinformatics 26(4):445–455. doi:10.1093/bioinformatics/btp713

Moore JH, Hill DP, Fisher JM, Lavender N, Kidd LC (2011) Human-computer interaction in a computational evolution system for the genetic analysis of cancer. In: Riolo R, Vladislavleva E, Moore JH (eds) Genetic programming theory and practice IX. Genetic and evolutionary computation. Springer, Ann Arbor, chap 9, pp 153–171. doi:10.1007/978-1-4614-1770-5-9

Moore JH, Hill DP, Sulovary A, Kidd L (2013) Genetic analysis of prostate cancer using computational evolution, pareto-optimization and post-processing. In: Riolo RL, Moore JH, Ritchie MD, Vladislavleva K (eds) Genetic programming theory and practice X. Genetic and evolutionary computation. Springer, Ann Arbor, pp 87–101

Pattin KA, Payne JL, Hill DP, Caldwell T, Fisher JM, Moore JH (2010) Exploiting expert knowledge of protein-protein interactions in a computational evolution system for detecting epistasis. In: Riolo R, McConaghy T, Vladislavleva E (eds) Genetic programming theory and practice VIII. Genetic and evolutionary computation, vol 8. Springer, Ann Arbor, chap 12, pp 195–210. http://www.springer.com/computer/ai/book/978-1-4419-7746-5

Payne J, Greene C, Hill D, Moore J (2010) Sensible initialization of a computational evolution system using expert knowledge for epistasis analysis in human genetics. In: Exploitation of linkage learning in evolutionary algorithms. Springer, Ann Arbor, chap 10, pp 215–226

Smits G, Kotanchek M (2004) Pareto-front exploitation in symbolic regression. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice II. Springer, Ann Arbor, chap 17, pp 283–299. doi:10.1007/0-387-23254-0-17

Chapter 3
Optimizing a Cloud Contract Portfolio Using Genetic Programming-Based Load Models

Sean Stijven, Ruben Van den Bossche, Ekaterina Vladislavleva, Kurt Vanmechelen, Jan Broeckhove, and Mark Kotanchek

Abstract Infrastructure-as-a-Service (IaaS) cloud providers offer a number of different tariff structures. The user has to balance the flexibility of the often quoted pay-by-the-hour, fixed price ("on demand") model against the lower-cost-per-hour rate of a "reserved contract". These tariff structures offer a significantly reduced cost per server hour (up to 50 %), in exchange for an up-front payment by the consumer. In order to reduce costs using these reserved contracts, a user has to make an estimation of its future compute demands, and purchase reserved contracts accordingly. The key to optimizing these cost benefits is to have an accurate model of the customer's future compute load – where that load can have a variety of trends and cyclic behaviour on multiple time scales. In this chapter, we use genetic programming to develop load models for a number of large-scale web sites based on real-world data. The predicted future load is subsequently used by a resource manager to optimize the amount of IaaS servers a consumer should allocate at a cloud provider, and the optimal tariff plans (from a cost perspective) for that allocation. Our results illustrate the benefits of load forecasting for cost-efficient IaaS

S. Stijven (✉)
Universiteit Antwerpen, Antwerp, Belgium

Ghent University–iMinds, Ghent, Belgium
e-mail: sean.stijven@uantwerpen.be

R. Van den Bossche • K. Vanmechelen • J. Broeckhove
Universiteit Antwerpen, Antwerp, Belgium
e-mail: Ruben.VandenBossche@uantwerpen.be; Kurt.Vanmechelen@uantwerpen.be; jan.broeckhove@uantwerpen.be

E. Vladislavleva
Evolved Analytics Europe, Beerse, Belgium
e-mail: katya@evolved-analytics.com

M. Kotanchek
Evolved Analytics L.L.C, Midland, MI, USA
e-mail: mark@evolved-analytics.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_3,

portfolio selection. They also might be of interest for the Genetic Programming (GP) community as a demonstration that GP symbolic regression can be successfully used for modelling discrete time series and has a tremendous potential for time lag identification and model structure discovery.

Keywords Cloud computing • Symbolic regression • Time series • Load prediction • Variable selection • Forecasting

1 Introduction

Cloud computing (Armbrust et al. 2010; Buyya et al. 2009) rapidly became an established IT outsourcing model, in which access to scalable IT services can flexibly be acquired and released in a pay-as-you-go manner. These services are hosted by big cloud providers, including Amazon EC2[1] and Rackspace[2] as protagonists. Factors that drive the success of this approach are the potential for cost reductions through economies of scale and the transformation of capital IT expenses into operational ones.

In Mell and Grance (2011), cloud services are categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS), depending on their focus of delivering respectively IT infrastructure, frameworks for software development and deployment, or a finished software product.

This contribution focuses on the IaaS service class in which a consumer can acquire access to compute capacity by launching server *instances* on the cloud provider's infrastructure. The instances' characteristics, such as processing power, memory and I/O capacity, are typically predefined by each provider as a number of *instance types*. Unique for these services are the *scalability* and *elasticity* of the system: most cloud providers offer tools to automatically increase or decease the number of servers instances quickly, based on the current load. This solves the over- or underprovisioning problems with which traditional IT infrastructure tends to struggle.

A diverse range of product offerings in terms of tariff structure is available. Amazon, for example, offers three pricing plans with a different trade-off between cost and quality-of-service (QoS):

On-demand Each instance type has a fixed *hourly* price, which is charged for each (partial) hour that an instance is running. This plan allows for the highest elasticity: sudden load peaks or drops can easily and automatically be settled by launching or killing instances, and one only has to pay for the actual runtime of each server instance.

[1] http://aws.amazon.com/ec2.

[2] http://www.rackspace.com.

Table 3.1 Amazon EC2 prices for `m1.small` in the US-East region

Usage level	Up-front cost	Hourly price	Period	Utilization optimum (%)	Cost reduction w.r.t. on demand (%)
Heavy	$169	$0.014	1 year	83–100	33–45
Medium	$139	$0.021	1 year	69–82	27–33
Light	$61	$0.034	1 year	27–68	0–26
On demand	n/a	$0.06	n/a	0–26	0

Reserved In exchange for an up-front commitment, these contracts offer a reduced hourly rate over a period of 1 or 3 years. Depending on the expected utilization time of an instance, one can choose between heavy, medium or light utilization instances. Each of these reserved contracts is associated with a different up-front and hourly cost. Reserved contracts are only beneficial if the reservation is used for a significant amount of time.

Spot Amazon's Spot market is a low-QoS market, on which excess data center capacity is sold at a much lower rate. The price fluctuates as it depends on the data center's current load. When the spot price for an instance type rises, all instances below that bid are killed immediately. The spot market is particularly interesting for non-real-time applications such as batch-type processing, which can be paused when the cost rises and resumed when the cost is low.

Consumers face an increasing complexity in making cost-efficient decisions when matching their infrastructure requirements to the services currently available. Depending on the utilization, the amount of time an instance is running compared to the period of the contract, it will be better to choose an on-demand instance or make an up-front purchase for a light, medium or heavy reserved instance. As an illustration, the prices for the default `m1.small` instance with a 1-year period are shown in Table 3.1. A reserved contract for an `m1.small` instance with a medium usage level and a period of 1 year will for example be the best choice for an instance that is running between 69 and 82 % of the time and will allow for a cost reduction of $96 (26 %) up to $141 (33 %).

It should be clear that the potential cost savings when using reserved contracts instead of or in addition to on demand contracts for an organization with hundreds of server instances can be huge. Essential to this approach however is the estimation of the future load, as this has an impact on the purchases made and real cost savings achieved. In this contribution, we focus on the optimization problem of minimizing over- and underprediction of load in the process of acquiring reserved contracts in a cost-efficient manner, under uncertainty on future load. We analyze how this optimization problem can be tackled by using forecasts of loads to build optimal contract portfolios and reduce cost.

To build and validate models, we use real world data of web applications for which we assume they are deployed using IaaS. We define load in terms of the number of visits per day. These data are taken from publicly available sources[3] and

[3] http://www.quantcast.com.

consist of discrete uni-variate time-series. The forecasting goal in this chapter is to predict loads at least 90 days into the future and do it with symbolic regression via genetic programming. The problem poses a challenge for the genetic programming domain, as the training data in all cases conceal a non-stationary nonlinear time series requiring long-term multi-step predictions.

The rest of this contribution is structured as follows: Sect. 2 introduces the problem domain of this paper. Section 3 discusses related work in the cloud computing and the genetic programming field. In Sect. 4 an algorithm is outlined that allows one to calculate a cost-efficient allocation of server instances to pricing plans given a predicted load, after which the experimental setup is presented in Sect. 5. The forecasting and scheduling results can be found in Sect. 6. Conclusions of this work are summarized in Sect. 7.

2 Problem Domain

Within the HICCAM (Hybrid Cloud Construction and Management) project (Van den Bossche et al. 2010, 2011, 2013), we investigate the design of software components and algorithms for building and deploying hybrid clouds efficiently, and for automated resource provisioning and management at the level of an entire organization. Within the project's software architecture outlined in Fig. 3.1, the organization's Cloud Procurement Endpoint (CPE) is responsible for managing the procurement of resources from public cloud providers. Requests for executing applications are sent to the CPE by the different decision support systems (DSS) that assist the users in making a trade-off between quality of service levels and costs.

In this contribution, the focus is on the long term planning problem of purchasing server instances with different service levels and prices from a public cloud provider in a cost-efficient manner, based on predictions of future load by using Genetic Programming.

The tariff structure of a cloud provider is represented as a generalization of Amazon's on-demand and reserved pricing plans. We assume that a cloud provider proposes a number of offerings, with an associated *up-front cost* (≥ 0) in exchange for a cheaper *hourly rate* during a given *period* ρ (in days). An offering includes one of the two *charging methods*:

As-you-go After paying the upfront cost, the customer is charged only for the actual usage (hours) of the concerned instance. If an instance runs for h hours, the customer is billed the upfront cost and h times the hourly rate.

Every hour The customer makes a commitment to use the instance continuously. Even if the instance is not running, the hourly rate is charged. If an instance type offering with this charging method is purchased, the cost at the end of the period is always the same, regardless of the number of hours the instance has actually been running.

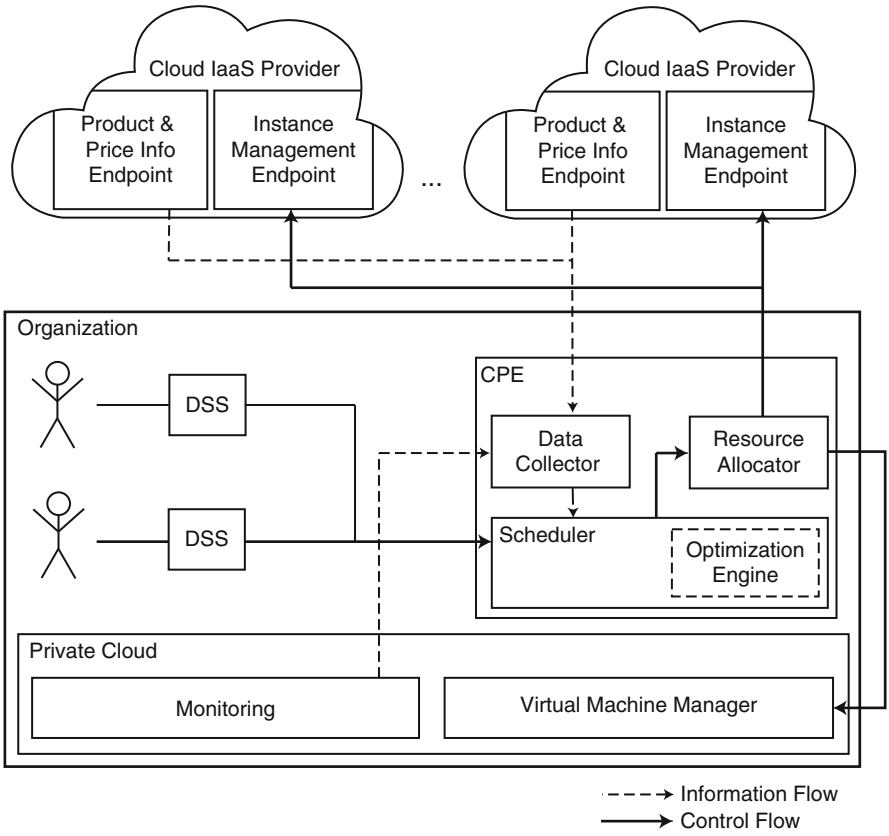


Fig. 3.1 Component view of the HICCAM model

An "on-demand" plan, in which no up-front cost is charged and server hours are charged as-you-go can be modeled by setting the up-front cost $= 0$ and the period $\rho = \infty$.

The contract portfolio management approach in this chapter supposes that the purchase decision is taken only once each period, and that all purchases are made at the first day of that period. The expansion of the scheduling algorithms to a "sliding" approach in which previous ongoing purchases are taken into account is left to future work.

Divergent user requirements and heterogeneous workload characteristics complicate the problem of identifying the best cloud provider for a specific application. The approach presented here is confined to only some of the characteristics and requirements, which we believe constitute a solid base to demonstrate the impact and performance of the prediction and scheduling mechanics.

Our current application model focuses on web application workloads, because they have a long lifetime which makes the use of prediction methods relevant.

3 Related Work

Cloud Computing

In Chaisiri et al. (2011), a stochastic programming approach is applied to the problem of server provisioning under price and demand uncertainty. They present both a short-term and long-term planning algorithm. Khatua et al. (2010) utilize an autoregressive model to predict load peeks in an application to automatically launch on-demand instances at the right time. In Lopes et al. (2010), a web application's load is classified as high, normal and low, and the authors assume that administrators have an idea about the amount of time their applications are in each of these load classes. In Tian et al. (2012), a linear program is used to minimize the provisioning cost for deploying a web application on both reserved and on-demand instances. They follow the same load classification as Lopes et al. (2010), and assume that the long term load is constant: last year's utilization (expressed as the number of hours in each load class) is used as the prediction for the upcoming year.

Genetic Programming for Discrete Time Series

Symbolic regression via genetic programming (GP) has been used for time series prediction since the seminal book by Koza (1992). In a majority of cases GP is given a task to learn the time series trend and reconstruct a dynamic system that generated time series using lagged (delayed) vectors. Given m lagged vectors $v_{t-1}, v_{t-2}, \ldots v_{t-m}$ induce a function f, such that

$$v_t = f(\mathbf{v}) = f(v_{t-1}, v_{t-2}, \ldots v_{t-m}). \tag{3.1}$$

Parameter m is often called an embedding dimension, lag parameter or a sliding time window. Sometimes another *delay* parameter τ is taken into account, and delayed lagged vectors $v_{t-\tau}, v_{t-2\tau}, \ldots v_{t-m\tau}$ are considered for model induction.

With few exceptions, like Yu et al. (2004) and Nikolaev and Iba (2001), GP employs a standard symbolic regression representation. Each individual is a superposition of primitives (being mathematical operators) with terminals sampled from a set of lagged vectors as well as constants.

Many use GP to complement autoregressive and moving average models to capture the nonlinearity of time series. Santini and Tettamanzi (2001) created a separate GP individual for each sample from the prediction horizon, but the vast majority of publications focus on generating single step predictions (predicting the next value) and iterating them over the test set. The problem with iterative predictions (when predicted values are used as inputs for subsequent evaluations) is that they tend to magnify very quickly (see Panyaworayan and Wuetschner (2002)).

In this work we attempt to generate accurate load predictions over a horizon of at least 90 unseen samples (days). Furthermore, all 90 samples are required to construct an optimal contract portfolio, i.e. the target prediction has to be in a form

$$(v_T, \ldots, v_{T+\rho})^T \in \mathbb{R}^\rho, \quad \rho = 90, \tag{3.2}$$

given the training data $(v_1, \ldots, v_{T-1})^T$.

As pointed out by Santini and Tettamanzi (2001) and Nikolaev and Iba (2001), the main advantage of GP for discrete time series prediction lies in its ability to discover dependencies among samples and automatically select only relevant variables for the models.

We have not found any GP references which used more than 20 consecutive samples (lags) as candidate inputs. One, four and ten lags are the most popular (Agapitos et al. 2008; Nikolaev and Iba 2001; Rodriguez-Vazquez and Fleming 1999). Schwaerzel and Bylander (2006) use a time window of the previous 20 data points, but no variable selection results are reported and only predictive accuracy of models rather than their interpretability was measured.

We believe that recent algorithmical improvements in the effectiveness of the variable selection capability of GP should allow exploration of much larger spaces of lagged variables. In this work we push GP towards $m = 365$ inputs and let driving variables be automatically discovered.

4 Algorithm Design for Contract Portfolio Optimization

In this section, an algorithm is outlined to find the optimal allocation of server instances over the available pricing plans, given predicted (or real) load data for the upcoming period. The algorithm should try to minimize cost and output a purchase suggestion in the form of a number of server instances for each pricing plan.

The inputs to the algorithm consist of the required instances with an associated instance type for each hour in the scheduling period, and the set of available pricing plans. The algorithm first calculates the utilization (in hours) for each of the instances running in the period, after which the best pricing plan for this utilization level is calculated based on the cost function described in Sect. 2. The cheapest plan is added to the suggested purchases.

This approach gives preference to the instances with the highest utilization throughout the relevant period. As a consequence, the result set is first filled with pricing plans that offer the highest cost reduction in return for the biggest upfront cost, and falls back on the plans with a lower or zero upfront cost to cope with temporary load surges. This results in a very cost-efficient result set.

5 Data and Experimental Setup

Data

Server utilization data in cloud infrastructures is unfortunately not made available by cloud providers. In our experiments, we therefore use the publicly available traces[4] of the daily number of page views for a number of well-known websites. We assume that the number of page views relates directly to the load that gets generated and hence to the number of server instances required to handle that load. As the page view data consists of daily averages, we assume the load is also constant through the day, that the web application is deployed on a number of homogeneous instances of the same instance type, and that number of instances is scaled linearly according to the *page views per instance* ratio.

In order to evaluate the performance and robustness of the predictions and of the scheduling algorithm, we use the daily page view data from four well-known web applications with different characteristics (see also Fig. 3.2):

1. **Stackoverflow.com:** The page view data shows robust growth with a lower number of views in the weekends and during the Christmas period.
2. **Imgur.com:** For this website we used the visitor data as the page view data contained odd properties, likely due to implementation details. The data shows robust growth until the middle of 2012, after which the growth seems to stagnate.
3. **Linkedin.com:** The page view data shows a strong seasonal pattern with drops in the weekends and holidays and a robust growth until the beginning of 2012, which diminishes in the last 14 months.
4. **Tinypic.com:** The page view data shows rapid decline with lower number of views in the weekends, but no strong seasonal pattern.

The data is in many cases non-stationary (means, variances and covariances change over time) and in all cases heteroscedastic (of different variability, i.e. not allowing constant error variance when modeled using ordinary regression[5]), which poses a good challenge to genetic programming, see Fig. 3.2.

Experiment Setup

We use the algorithm outlined in Sect. 4 with different inputs of expected future load to assess the added value of using predicted future loads to purchase a portfolio of reserved contracts and minimize infrastructure costs. The results of the following predicted input scenarios are compared in Sect. 6.

[4] http://www.quantcast.com.

[5] As in a model $Y_t = \mathcal{M}(t) + \sigma(t)\epsilon_t$.

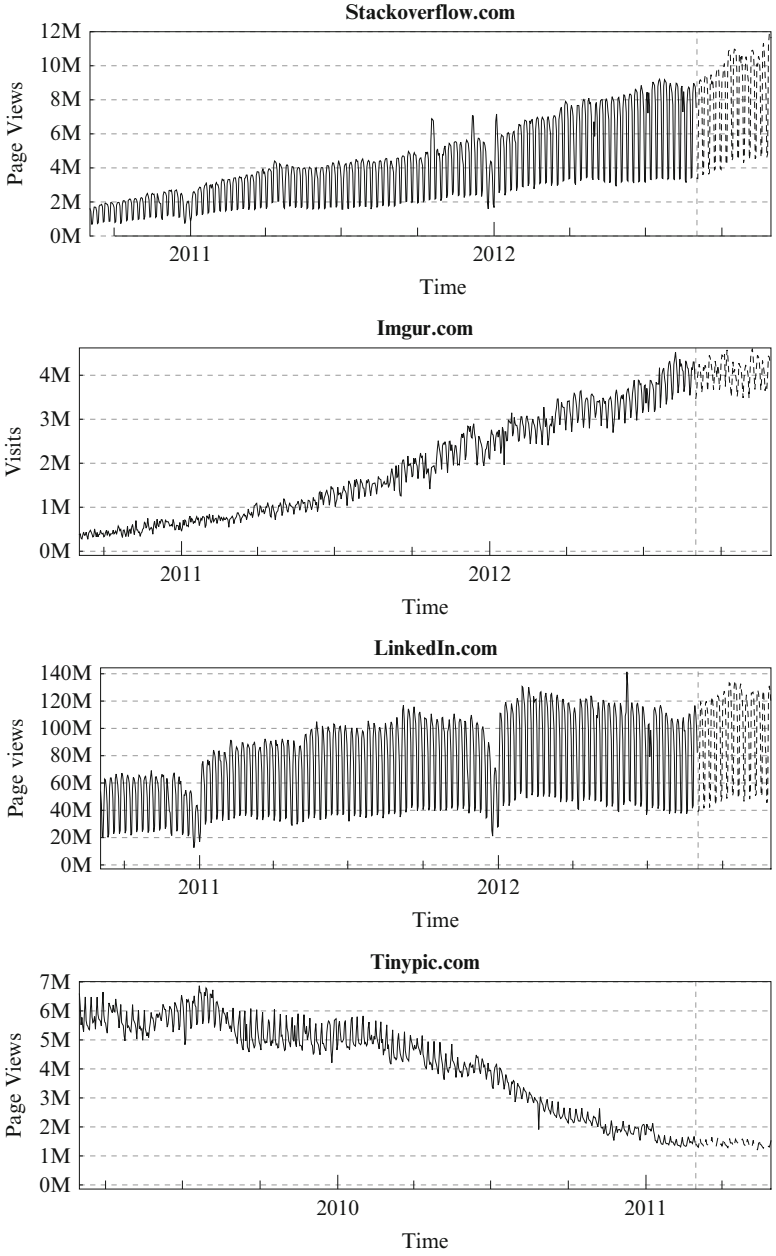


Fig. 3.2 The data of the four websites plotted over time. *Solid lines* indicate the part of the data that was used as training set. *Dashed lines* indicate the part of the data that was used as test set

1. **Optimal:** This scenario is the "reference" scenario, in which the customer has full knowledge of the future and selects an optimal portfolio based on the real future load.
2. **Prediction:** The customer purchases reserved contracts by taking into account the GP predicted load for the scheduling period of ρ days.
3. **One period back:** The customer purchases the same instances as were needed in last ρ days.
4. **One year back:** The customer purchases the same instances as were needed in the same period of ρ days exactly 1 year ago (assuming that utilization levels would be the same as in the period 1 year ago).

In all experiments of this contribution, we use a prediction and scheduling period of $\rho = 90$ days. Although not veracious at the time of writing, it can be argued that a cloud provider could be interested in providing shorter-time reservation periods in addition to the 1 and 3-year periods currently available at Amazon EC2. This would enable customers with highly varying load patterns, which are currently exclusively using on-demand instances, to also make up-front commitments for shorter periods. The support for multiple offerings with different periods and the incorporation of longer-term predictions is left for future work.

The inputs for the prediction scenario are calculated using symbolic regression via GP implemented in *DataModeler* (Evolved Analytics LLC 2011). We use Pareto-aware symbolic regression to optimize both accuracy and simplicity of the models. The fitness function for prediction accuracy was defined as $1 - R^2$ where R^2 is the square of the scaled correlation between prediction and response. As complexity measure we use the sum of subtree sizes of the model expression, which is equivalent to the total number of links traversed starting from the root to each of the terminal nodes of the parse tree.

In practice, data transformation and normalization methods precede analysis of time series (Nikolaev and Iba 2001; Agapitos et al. 2008). We, however, applied symbolic regression to raw data to prepare for online modeling of streamed series.

To let GP automatically discover significant lags, we convert the original load vector to a data matrix in the following way: for a fixed length of a sliding time window m, each load value v_t was augmented to a row vector $(v_t, vt-1, \ldots, v_{t-m}) \in \mathbb{R}^{1\times m}$. Omitting the first m samples of the response vector $\mathbf{v} = (v_1, \ldots, v_T)^T$ (for which not all lags are available) leads to a data matrix with $T - m$ rows and m columns.

The lag parameter m is fixed to $m = 365$ days in all experiments.

To generate predictions 90 days ahead without iteratively accumulating prediction errors, we exclude lags $(v_{t-1}, \ldots, v_{t-90})$ from the modeling and use the first column v_t of the augmented data table as the response. The modeling task for the lag window m and the scheduling horizon ρ is now formulated as a search for a model or a model ensemble f, such that:

$$v_t = f(v_{t-\rho-1}, v_{t-\rho-2}, \ldots v_{t-m}) + \epsilon, \quad m > \rho. \tag{3.3}$$

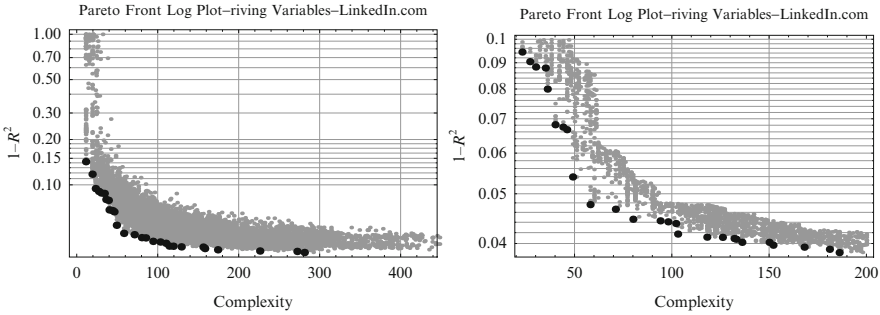


Fig. 3.3 Pareto front plots showing the complexity accuracy profiles of the LinkedIn.com models. The *left plot* shows all models of the SR run. The *right plot* shows only the models using the driving variables for a quality box of complexity 200 and accuracy 0.1

Each experiment presented here consists of two modeling stages of 40 independent symbolic regression runs, with each run executing for 900 s. The results of stage 1 experiments define the driving variables and variable combinations (optimal time lags). Stage 2 experiments use only driving variables as inputs. Final solution ensembles were constructed from stage 2 models. Prediction is then evaluated on the test time period (see Fig. 3.2), used for constructing an optimal contract portfolio, and compared with a baseline on-demand strategy applied to real loads in the test period.

6 Results

SR Results

Exclusion of the 90 most recent lags from the modeling allowed us to forecast the loads 90 days into the future using the available training data without the need to iterate predicted loads and accumulate prediction error. The quality of predictions was evaluated with respect to $1 - R^2$, the Normalized Root Mean Square Error (NRMSE) and the Hit percentage (HIT) (Agapitos et al. 2008).

The forecasts for the testing horizon for all four websites are depicted in Fig. 3.4. The *stackoverflow.com* predictions are of high quality as the previously observed growth in page views (from Fig. 3.2) continues in the test period. The *imgur.com* predictions do well for the first month of the predicted period, but afterwards the growth stagnates and the prediction overshoots because this behavior is not observed in the training data. The *linkedin.com* predictions follow the growth curve as observed in the training data and are therefore very close to the actual data. The *tinypic.com* data stagnates at the end of the training period and continues to do so in the test period. SR predictions follow the declining trend of the training set and

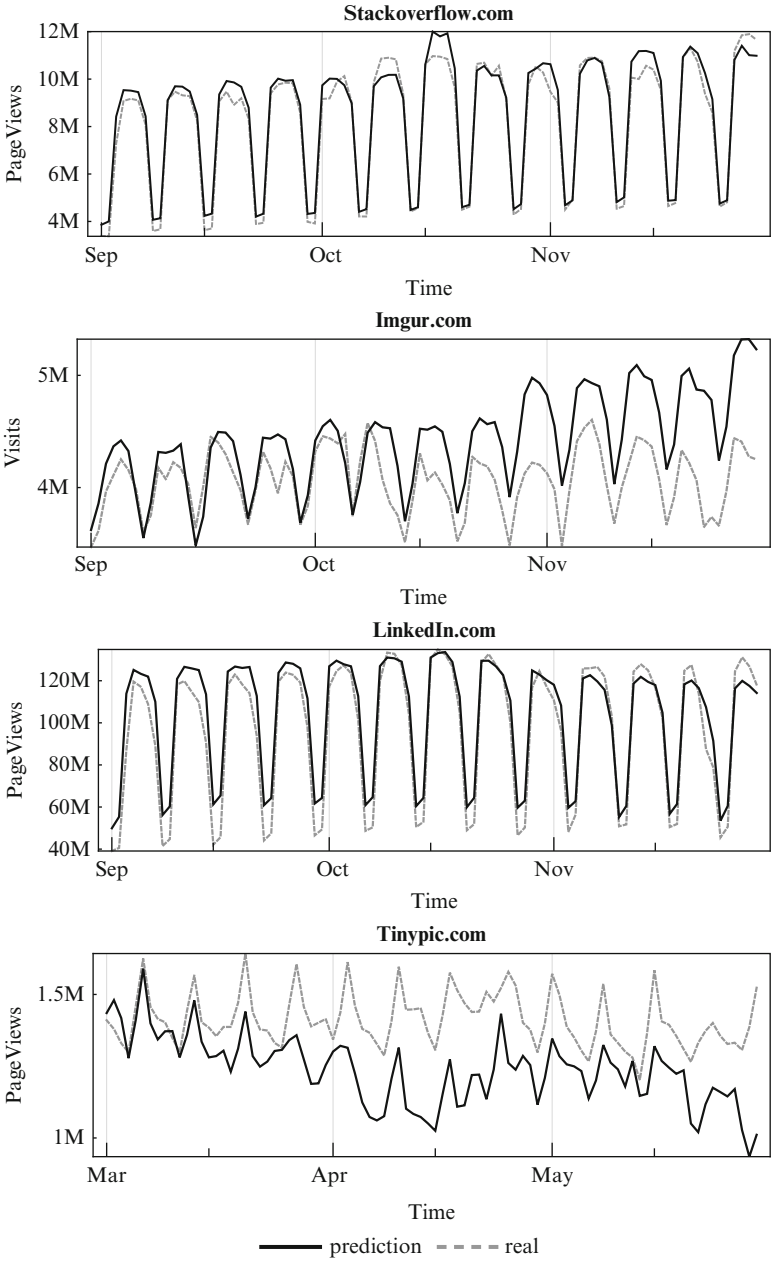


Fig. 3.4 The GP prediction results for the four websites (shown in *solid black*). The actual page view data is indicated by the *dashed gray lines*

Table 3.2 Overview of the prediction errors for the data sets. The prediction errors are shown for the training data, the test data and the first, second and third 30 day period of the test data

Data set		$1 - R^2$	NRMSE	HIT
Stackoverflow.com				
Training data		0.0325	0.0509	0.838
Test data	Day 01–30	0.0073	0.0714	0.793
	Day 31–60	0.0334	0.0726	0.826
	Day 61–90	0.0331	0.0700	0.931
	Day 01–90	0.0254	0.0576	0.843
Imgur.com				
Training data		0.0499	0.0496	0.764
Test data	Day 01–30	0.2350	0.1941	0.862
	Day 31–60	0.5354	0.3506	0.828
	Day 61–90	0.3680	0.6109	0.828
	Day 01–90	0.5258	0.4123	0.843
LinkedIn.com				
Training data		0.0573	0.0612	0.942
Test data	Day 01–30	0.0276	0.1643	0.966
	Day 31–60	0.0102	0.0886	0.897
	Day 61–90	0.0401	0.0944	0.966
	Day 01–90	0.0453	0.1080	0.944
Tinypic.com				
Training data		0.0233	0.0431	0.805
Test data	Day 01–30	0.7009	0.3407	0.724
	Day 31–60	0.7132	0.8235	0.621
	Day 61–90	0.8601	0.5604	0.862
	Day 01–90	0.8291	0.4756	0.741

therefore predict a page view count which is lower than the observed count. On the other hand, the predictions are still closer to the test data than the data of the previous period. Table 3.2 shows the prediction errors for both the training and test data.

For the sake of completeness, Fig. 3.3 depicts a Pareto front of all models generated for *linkedin.com* data as well as a subset of models that use the driving variables. Fig. 3.5 shows an overview of the algebraic expressions of the Pareto front models.

Validation Through Simulation

The main goal of this chapter is to explore the benefits of forecasting the computational load to make data-driven decisions about purchasing IaaS contracts. To achieve this goal, we compare three data-driven load forecasting scenarios to a straightforward paid on-demand service. The reference cost of an on-demand

LinkedIn.com – Model Selection Report

	Complexity	$1–R^2$	Function
1	23	0.094	$4.666 + \frac{-28.093}{7 + delay_{238} + delay_{364}}$
2	30	0.088	$1.657 + -0.098\ (-3.150 + delay_{238} + delay_{364})^2$
3	34	0.088	$0.733 + -0.123\ (delay_{238} + delay_{364})\ (-5 + delay_{287} + delay_{364})$
4	36	0.080	$6.325 + \frac{-58.134}{10 + delay_{217} + delay_{238} - delay_{294} + delay_{364}}$
5	39	0.079	$0.617 + -0.147\ (delay_{238} + delay_{364})\ (-5 + delay_{287} + delay_{364}^2)$
6	40	0.071	$2.040 + -0.244\ (delay_{116} + delay_{275} + (2.021 - delay_{364})^2)$
7	46	0.066	$1.646 + -0.103\ (delay_{116} + delay_{275} + (-2.848 + delay_{238} + delay_{364})^2)$
8	49	0.063	$2.702 + -0.183\ (delay_{116} + delay_{296} + delay_{301}2 + (3 - delay_{364})^2)$
9	53	0.062	$2.934 + -0.205\ (delay_{116} + delay_{275} + (3 - delay_{364})^2 + delay_{364} + delay_{364}^2)$
10	62	0.059	$1.012 + 0.260\ (-delay_{109} + delay_{238} - delay_{294}2 - delay_{296}\,3 + delay_{364})$
11	63	0.052	$3.580 + -0.261\ (delay_{116} + delay_{296} + delay_{301}2 + (3 - delay_{364})^2 + 2\ delay_{364})$
12	72	0.048	$3.000 + -0.215\ (delay_{116} - delay_{238} + delay_{275} + delay_{301}2 + (2.999 - delay_{364})^2 + 2\ delay_{364})$
13	77	0.048	$1.058 + 0.215\ (-delay_{116} + delay_{238} - delay_{296} - delay_{301}2 + 4\ delay_{364} - delay_{364}2)$
14	115	0.048	$1.009 + 0.101\ (-2\ delay_{109} + delay_{217} + delay_{231} + delay_{238} - delay_{294}2 - 2\ delay_{296} - 2\ delay_{301}2 + 8\ delay_{364} - delay_{364}2)$
15	129	0.046	$1.018 + 0.101\ (-delay_{109} - delay_{116} + delay_{231} + 2\ delay_{238} - delay_{275} - delay_{296}2 - 2\ delay_{301}^2 + 8\ delay_{364} - delay_{364}2)$

Fig. 3.5 Pareto front of models that use the driving variables – for the LinkedIn data

Table 3.3 Assumed cloud offerings for instance type `m1.small`

Usage level	Up-front cost	Hourly price	Period ρ	Charging method
Heavy	$42.25	$0.014	90 days	Every hour
Medium	$34.75	$0.021	90 days	As-you-go
Light	$15.25	$0.034	90 days	As-you-go
On demand	N.A.	$0.06	N.A.	As-you-go

service for the scheduling period is estimated using real load from the test data. The three prediction scenarios employ loads predicted by GP, actual loads of the previous scheduling period, and actual loads of the same period 1 year back.

To map page views to the computational cost, we assume that the *page views per instance* ratio is known in advance for a certain instance type and that the ratio is constant for different page requests.

The prices for the different offerings are derived from the Amazon EC2 reserved prices for the `m1.small` instance type, in which the period is reduced to 90 days and the upfront cost is proportionally reduced with three quarters. These prices and conditions are listed in Table 3.3.

For each of the forecasting scenarios we construct an optimal contract portfolio and compare the cost of this portfolio to the baseline cost of the optimal portfolio constructed for real data. The results for all four websites are presented in Fig. 3.6.

Figure 3.6 illustrates that optimal contracts generated using GP-based load forecasts (bars labeled as "Prediction") provide superior results with respect to cost benefits compared to the actual optimal purchasing decisions for all four websites. In three out of four cases GP-based contracts are significantly better than the contracts which anticipate the same loads as in the previous period of ρ days.

For clarity reasons, the cost when using only on demand instances is not shown in Fig. 3.6, but they exceed the cost in the "optimal" reserved scenario by 52, 72, 51, and 73% for Stackoverflow.com, Imgur.com, LinkedIn.com, and

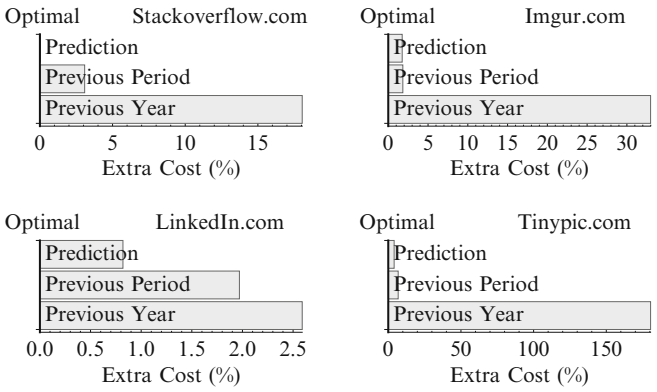


Fig. 3.6 The extra cost percentage compared to the optimal, for hosting the four websites as calculated by the cloud scheduling simulator

tinypic.com respectively. For the LinkedIn website the on-demand costs under our assumptions and prices from Table 3.3 would be $117,094, the contract portfolio in the optimal scenario would cost $77,338, and the prediction based contract for any GP-enthusiast would cost $77,975.

These figures show that it is beneficial for IaaS customers to take into account reserved contracts and consider using prediction models of future load to make the right purchases. Genetic programming proves to be able to build such load prediction models fairly well.

7 Conclusion

In this chapter we illustrated the value of data analysis for taking smarter data-driven decisions for the application of purchasing cloud computing services. We showed that the most successful strategy for selecting contract portfolios is based on upfront reservation of compute instances using forecasted utilization levels obtained by genetic programming.

We showed that symbolic regression via genetic programming (implemented in DataModeler) routinely filters out handfuls of driving variables out of several hundreds of candidate inputs. We also demonstrated the competitive advantage of running multi-objective symbolic regression to evolve not only accurate but also 'short' highly interpretable relationships.

For future work we will avoid the limiting decisions taken to design experiments in this chapter – fixing the sliding time window and maximal number of lags m, and fixing the length of the analysis window T. Wagner et al. (2007) proposed a good heuristic to estimate the optimal values for these parameters (albeit no lagged variables were used as potential inputs). We believe that both the analysis window as well as the sliding time window can be evolved in the same symbolic regression process.

References

Agapitos A, Dyson M, Kovalchuk J, Lucas SM (2008) On the genetic programming of time-series predictors for supply chain management. In: Keijzer M, Antoniol G, Congdon CB, Deb K, Doerr B, Hansen N, Holmes JH, Hornby GS, Howard D, Kennedy J, Kumar S, Lobo FG, Miller JF, Moore J, Neumann F, Pelikan M, Pollack J, Sastry K, Stanley K, Stoica A, Talbi EG, Wegener I (eds) GECCO'08: proceedings of the 10th annual conference on genetic and evolutionary computation, Atlanta. ACM, pp 1163–1170. doi:10.1145/1389095.1389327, http://www.cs.bham.ac.uk/~wbl/biblio/gecco2008/docs/p1163.pdf

Armbrust B, Griffith R, Joseph AD, KatzR, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. Commun ACM 53(4):50–58. http://dl.acm.org/citation.cfm?id=1721672

Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616. http://dx.doi.org/10.1016/j.future.2008.12.001

Chaisiri S, Kaewpuang R, Lee BS, Niyato D (2011) Cost minimization for provisioning virtual servers in Amazon elastic compute cloud. In: 2011 IEEE 19th annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems, Singapore. IEEE, pp 85–95. doi:10.1109/MASCOTS.2011.30, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6005371

Evolved Analytics LLC (2011) DataModeler release 8.0 documentation. Evolved Analytics LLC. www.evolved-analytics.com

Khatua S, Ghosh A, Mukherjee N (2010) Optimizing the utilization of virtual resources in cloud environment. In: 2010 IEEE international conference on virtual environments, human-computer interfaces and measurement systems, Taranto. IEEE, pp 82–87. doi:10.1109/VECIMS.2010.5609349, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5609349

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Lopes R, Brasileiro F, Maciel P (2010) Business-driven capacity planning of a cloud-based it infrastructure for the execution of web applications. In: 2010 IEEE international symposium on parallel and distributed processing, workshops and Phd forum (IPDPSW), Atlanta, pp 1–8. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5470726

Mell P, Grance T (2011) The NIST definition of cloud computing. NIST Spec Publ 800(145). http://pre-developer.att.com/home/learn/enablingtechnologies/The_NIST_Definition_of_Cloud_Computing.pdf

Nikolaev N, Iba H (2001) Genetic programming of polynomial harmonic models using the discrete fourier transform. In: Proceedings of the 2001 Congress on evolutionary computation, 2001, vol 2, Samseong-dong, pp 902–909. doi:10.1109/ CEC.2001.934286

Panyaworayan W, Wuetschner G (2002) Time series prediction using a recursive algorithm of a combination of genetic programming and constant optimization. Facta Universitatis Series: Electron Energ 15(2):265–279. http://factaee.elfak.ni.ac.yu/fu2k22/11wp.pdf

Rodriguez-Vazquez K, Fleming PJ (1999) Genetic programming for dynamic chaotic systems modelling. In: Angeline PJ, Michalewicz Z, Schoenauer M, Yao X, Zalzala A (eds) Proceedings of the Congress on evolutionary computation, vol 1, Washington, DC. IEEE, pp 22–28

Santini M, Tettamanzi A (2001) Genetic programming for financial time series prediction. In: Proceedings of the 4th European conference on genetic programming, EuroGP'01, London. Springer, pp 361–370. http://dl.acm.org/citation.cfm?id=646809.704093

Schwaerzel R, Bylander T (2006) Predicting currency exchange rates by genetic programming with trigonometric functions and high-order statistics. In: Proceedings of the 8th annual conference on genetic and evolutionary computation, GECCO'06, Seattle. ACM, New York, pp 955–956. doi:10.1145/1143997.1144167, http://doi.acm.org/10.1145/1143997.1144167

Tian C, Wang Y, Qi F, Yin B (2012) Decision model for provisioning virtual resources in Amazon EC2. In: Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), Las Vegas, NV, U.S.A pp 159–163. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6380006

Van den Bossche R, Vanmechelen K, Broeckhove J (2010) Cost-Optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: 2010 IEEE 3rd international conference on cloud computing, Miami, pp 228–235. doi:10.1109/CLOUD.2010.58, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5557990

Van den Bossche R, Vanmechelen K, Broeckhove J (2011) Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In: 2011 IEEE third international conference on cloud computing technology and science, Athens, pp 320–327. doi:10.1109/CloudCom.2011.50, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6133159

Van den Bossche R, Vanmechelen K, Broeckhove J (2013) Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Gener Comput Syst 29(4):973–985. doi:10.1016/j.future.2012.12.012, http://linkinghub.elsevier.com/retrieve/pii/S0167739X12002324

Wagner N, Michalewicz Z, Khouja M, McGregor R (2007) Time series forecasting for dynamic environments: the DyFor genetic program model. IEEE Trans Evol Comput 11(4):433–452. doi:10.1109/TEVC.2006.882430

Yu T, Chen SH, Kuo TW (2004) Discovering financial technical trading rules using genetic programming with lambda abstraction. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice II. Springer, Ann Arbor, chap 2, pp 11–30. doi:10.1007/0-387-23254-0-2

Chapter 4
Maintenance of a Long Running Distributed Genetic Programming System for Solving Problems Requiring Big Data

Babak Hodjat, Erik Hemberg, Hormoz Shahrzad, and Una-May O'Reilly

Abstract We describe a system, ECStar, that outstrips many scaling aspects of extant genetic programming systems. One instance in the domain of financial strategies has executed for extended durations (months to years) on nodes distributed around the globe. ECStar system instances are almost never stopped and restarted, though they are resource elastic. Instead they are interactively redirected to different parts of the problem space and updated with up-to-date learning. Their non-reproducibility (i.e. single "play of the tape" process) due to their complexity makes them similar to real biological systems. In this contribution we focus upon how ECStar introduces a provocative, important, new paradigm for GP by its sheer size and complexity. ECStar's scale, volunteer compute nodes and distributed hub-and-spoke design have implications on how a multi-node instance is managed. We describe the set up, deployment, operation and update of an instance of such a large, distributed and long running system. Moreover, we outline how ECStar is designed to allow manual guidance and re-alignment of its evolutionary search trajectory.

Keywords Learning classifier system • Cloud scale • Distributed • Big data

B. Hodjat (✉) • H. Shahrzad
Genetic Finance, San Francisco, CA 94105, USA
e-mail: babak@geneticfinance.net; hormoz@geneticfinance.net

E. Hemberg • U.-M. O'Reilly
ALFA Group, CSAIL, MIT, Cambridge, MA, USA
e-mail: hembergerik@csail.mit.edu; unamay@csail.mit.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_4,

1 Introduction

Despite executing a variant of island-based evolution with migration, ECStar, previously described in O'Reilly et al. (2012), Hemberg et al. (2013b), and Hodjat and Shahrzad (2012), is a genetic programming (GP) system which is remarkably different from other distributed evolutionary algorithms (EA). First, it demonstrates a number of aspects of scaling which outstrip those of extant GP or EA systems. For example, one ECStar system instance learning financial trading strategies executes on nodes which are spatially distributed across the globe and another instance of ECStar is learning forecasting rules in the medical domain. Further, a widely deployed ECStar system instance is very seldom stopped and restarted. This avoids incurring the overhead of deploying and retracting its many compute nodes. It is designed to be able to simultaneously work on different sub-parts of a problem and over time it synthetically integrates partial solutions. Simultaneously, it is able to shift to new aspects of the problem by accepting, without interruption, new learning data or parameter settings.

Another way of contextualizing ECStar is to compare it to artificial life (ALIFE) systems (Bedau 2003). Both share an aspect of longevity in that they are run without a goal of termination. Both share the nature that a snapshot of current state, at any time in the extended duration of their execution, is insightful. ALIFE systems are studied for their life-like behavior in a digital setting (i.e. they offer explanations of a biological process or system). In contrast, ECStar is executing toward solving a problem. It can be polled at anytime, at its Evolutionary Coordinator, to provide its latest best evolved solution. In some commonality with ALIFE systems, there is a continuous interactive relationship between ECStar and its developers. A developer is able to "tinker" with the evolutionary direction of the system. The interactions are indirect (e.g. changes to resources, modifications in parameters) and intentionally light handed: they nudge rather than interfere. This guidance is motivated by problem solving, whereas in ALIFE triggering reactions to perturbations are often the goal.

One further analogy is helpful, ECStar's developer's goal and means of interaction, from the perspective of making a system evolve toward something, places the developer in the role of a bench scientist using directed evolution to isolate a biological property of interest in a cell line. The bench scientist aims to coax evolution to biologically generate a collection of cells with a specific property. A starting cell population is allowed to proliferate and evolve, then, iteratively it is filtered (and perhaps divided for concurrent evolution) to focus its trajectory with an end point in mind. Evolution accomplishes the work of search and adaptation with in-vivo selection while the bench scientist adds an external selection pressure to the process.

In this contribution we augment existing descriptions of ECStar which focus on its distributed design and application, see O'Reilly et al. (2012), Hemberg et al. (2013b), and Hodjat and Shahrzad (2012). We comment on the way in which ECStar's scale, resource choice of nodes offering idle cycles and distributed

hub-and-spoke design have implications on how a very large, multi-node instance is managed. We describe the entirely new proposition of setting up, deploying, operating, monitoring, harvesting, securing and even software updating an instance of such a large, distributed and long running system. Broadly, we describe how it is possible to minimize the installation of new Evolutionary Engines while enabling wide ranging, tunable behavior that a developer can oversee and control. We describe how ECStar is designed to allow manual guidance and re-alignment of its evolutionary "trajectory" through an assortment of mechanisms.

We proceed as follows. In Sect. 2 we compare ECStar in terms of distributed model, size and use of volunteer compute node idle cycles to related work. Section 3 provides a description of ECStar sufficient to visit its design motivation. In Sect. 4 the long running and large scale system features are explained. The operation and direction of ECStar are explained in Sect. 5. In Sect. 6 there is a discussion of how ECStar influences the GP experimental paradigm. Finally, a summary and future work are in Sect. 7.

2 Previous Work

There are other distributed GP systems documented but none has the same architecture, size, execution duration or particular use of volunteer nodes' idle cycles as ECStar.

In terms of architecture, typically distribution follows two basic models:

- Master-slave where the fitness function evaluation is distributed and a single server executes the main evolutionary algorithm.
- Islands with migration where a number of independent EA algorithms each execute on a node, and exchange best solutions regularly (though often asynchronously) using a fixed neighbor topology (Cantu-Paz 2000; Tomassini 2005; Crainic and Toulouse 2010; Scheibenpflug et al. 2012).

ECStar and just a few others use the model where a pool of individuals are coordinated centrally while a set of client nodes evolve them and pass their best to the central server. Merelo et al. (2012) report an initial exploration examining pool versus island based models. Their algorithm IslandSofEA uses separate clients and a pool and scales best of their tested methods.

In terms of size, i.e. number of nodes deployed, a dated but valuable description of larger GP systems and how to build a parallel system can be found in Bennett III et al. (1999). This source reports how to build a parallel computer system for $18,000 (1999 prices) that performs a half peta-flop per day. It employed 10 nodes with 533 MHz Alpha processors. Langdon (2012) talks about running GP on the Emerald GPU super computer which has 1008 ×86 CPU cores and 372 nVidia M2090 Tesla (in total Emerald has 190,464 stream processors), providing an average of 33 $GPopS^{-1}$. An instance of ECStar has been run on more nodes than either of these two systems. However it is difficult to quantify a comparison because

ECStar uses idle cycles of many different types of CPUs. ECStar's Evolutionary Engine software has not been ported to execute on a GPU because, at the time, GPU computing offers too small a fraction of its resource pool. A GPU, because it is SIMD, is best deployed to execute a lot of data simultaneously. A good strategy is for the GP individual to be evaluated after compilation or with a compiled evaluator downloaded into the GPU with its fitness evaluation cases distributed across the GPU. Empirical design questions as to the breakpoints where this strategy works for ECStar would have to be resolved first. ECStar does not exclude the possibility that individuals from Evolutionary Engines with GPUs can be reported to the Evolutionary Coordinator. One other evolutionary algorithm system also volunteer based has a large number of hosts, Desell et al. (2010) execute a distributed genetic algorithm for the MilkyWay@home project which, at time of this article being written, lists 316,902 hosts (see http://boincstats.com/).

In terms of systems that employ volunteer resources, the ways in which they are used is diverse. de Vega et al. (2012) present a customizable execution environment for evolutionary computation using BOINC plus virtualization, by running VMWare in the BOINC wrapperC – see Anderson (2004). The computing model is focused on institutions, which own their computers, and an end-user application that administers the BOINC resources of an institution and allows existing EAs to run multiple executions. The scale is smaller than ECStar and the system is used for conventional EA runs, with no mention of long running evolution or any description of how to maintain it. Smaoui and Garbey (2013) study how to improve volunteer computing scheduling for evolutionary algorithms. They only use the volunteer compute node clients for computation of fitness function, not for running independent EAs per the master-slave model.

Desell et al. (2010), who run a distributed genetic algorithm for the Milky-Way@home project, also use clients for fitness evaluation in the master-slave model. Fault tolerance in a master-slave environment for evolutionary algorithms is discussed in Gonzalez et al. (2012). The authors find the parallel EAs to be robust to faults, i.e. loss of fitness evaluations on the slave nodes. In ECStar the Evolutionary Engine executes an EA and sends solutions to the Evolutionary Coordinator, thus the fault tolerance considers churn of Evolutionary Engines instead of only single fitness evaluations.

We next provide an overview of the ECStar system architecture.

3 ECStar System Architecture

ECStar is a distributed EC system, see Fig. 4.1, that employs a modified decision list representation (Rivest 1987). It uses a large number of volunteer compute nodes as "Evolutionary Engines". The individual solutions from the Evolutionary Engines are coordinated by an Evolutionary Coordinator, and data is distributed to an Evolutionary Engine by a Data Server.

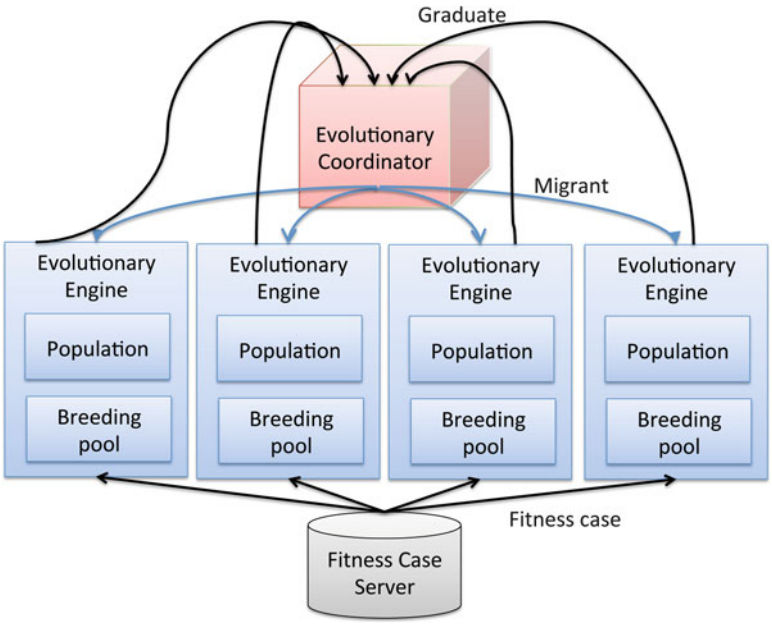


Fig. 4.1 The hub and spoke model of ECStar

Evolutionary Engine

Each Evolutionary Engine runs a completely independent evolutionary algorithm in its client's spare cycles. It has a fixed population size and initially generates the population randomly. In the evolutionary loop, it requests fitness cases (training data) from the fitness case server in the form of a *data package*. Each individual in the population is evaluated, and after a fixed number of data packages, selection and breeding take place before replacement and the next generation. Periodically local individuals become graduates and are dispatched to the Evolutionary Coordinator and *migrants* are received from the Evolutionary Coordinator.

Evolutionary Engines are unpredictably available with an unknown work and communication channel capacity. These constraints (deriving from the volunteer "contract") mean that Evolutionary Engines are designed to occupy a modest CPU, RAM, storage and network footprint, plus devote added computational time to security. Hodjat and Shahrzad (2012) and Hemberg et al. (2013a,b) describe in detail ECStar's age layering model for training data access and usage.

Evolutionary Engine Representation

An individual in ECStar is represented as a set of conjunctive rules. Operating on logical predicates based on expressing conditions of tests on the problem domain features gives the ECStar solutions the advantage of being human readable.

An individual (a.k.a. *classifier*) has a header with id, age, fitness and a body with a set of rules. Each rule is a variable length conjunctive set of conditions with an associated action which is a class in a classification problem. Each condition acts as a propositional variable, which is then applied to the discretized real-valued training environment. Apart from conjunction operators each condition can have, a *complement* operator, negating the truth value and a *lag* which refers to "past" values of an attribute.

```
<rules> ::= <rule> | <rule> <rules>
<rule> ::= <conditions> => <action>
<conditions> ::= <condition> | <condition> & <conditions>
<action> ::= prediction label
<condition> ::= <predicate> | !<condition>  | <condition> [lag]
<predicate> ::= truth value on a feature indicator
```

Each individual in an Evolutionary Engine is evaluated on a number of fitness cases every generation. Each fitness case is in a data package and consists of a number of rows, called events. For each event, the variables in the classifiers rules' conditions are bound to the features of the event. For each rule in the individual, the rule's conditions are evaluated. If all are true, the rule is added to a *Match Set*. Finally, a voting mechanism elects from the *Match Set* a single rule's action as a prediction for the fitness case. When no rule fires, no prediction is made. We track $activity$ and use it as a basis for selection into the breeding population. Predictions for all events in a fitness case are verified against the true class labels. A correct prediction increases the fitness and incorrect decreases. Each individual records its fitness in two ways: relative fitness and absolute fitness. (Relative fitness is the fitness per data package evaluated, i.e. fitness normalization.)

Evolutionary Coordinator

The Evolutionary Coordinator coordinates migration among the Evolutionary Engines. It maintains a layered *archive* – a sorted set of individuals that are currently the best from all Evolutionary Engines. When a *migrant* query is received, *migrants* are randomly picked from its archive and sent to the requesting Evolutionary Engine. Migration serves two purposes: it allows individuals to be sent out to each Evolutionary Engine in order to be evaluated on more fitness cases (because Evolutionary Engines appraise an individual's fitness on only a fraction of the fitness cases before selecting graduates) and it allows the arriving migrants to mix their genetic material with the host Evolutionary Engine's local population. When a *migrant* message is received, the returning individuals compete for the space in the

archive if it is full. The comparisons will be asymmetric, since different individuals have been evaluated on different data packages but the layers maintain individuals evaluated on approximately the same number of data packages to minimize noisy comparison. More details of archive management are mentioned in section "Archive Selection Pressure Through Alive/Dead State".

4 ECStar Design Rationale

We split our discussion of the reasons behind ECStar's design choices into two subsections. In section "Idle Cycles, EA Requirements and Design Choices" we discuss how general properties of volunteer computing and using ECStar's hub and spoke model impose requirements on its distributed evolutionary algorithm and these requirements, in turn, lead to its design choices. In section "Scale Related Design Rationale" we next discuss how requirements imposed by the sheer large size and running time of ECStar have prompted design choices.

Idle Cycles, EA Requirements and Design Choices

Cost is arguably a primary driver for using idle cycles. Using volunteer compute nodes makes an ECStar system instance inexpensive relative to owning equivalent hardware capacity, or contracting cloud services. The decision to use idle cycles leads to a number of requirements that, in turn, lead to ECStar's design decisions. In this section, we connect the consequences to the requirements then to the design decisions. We accomplish this in a space efficient manner by cross referencing Tables 4.1–4.3. Table 4.1, numerically itemized, enumerates properties associated with using large quantities of volunteer compute nodes in a hub-and-spoke model with a many to one relationship between a volunteer client (a.k.a. node) and a dedicated server. Recall that in the ECStar system, servers support Evolutionary Coordinators and Data Servers. Table 4.2, alphabetically itemized, presents design

Table 4.1 Properties associated with using many volunteer compute nodes

1. Large scale geographic distribution of nodes implies, at a macro-scale, that resource availability follows diurnal cycles across time zones. At a micro-scale, volunteer clients are available unpredictably
2. RAM and permanent storage at each client (a.k.a. Evoloutionary Engine) are small
3. Clients are not allowed to communicate with each other to ensure privacy of each volunteer client
4. Extended time is required to enlist, deploy or shut down large quantities of clients
5. Communication channels are insecure and noisy and their capacity is modest
6. In many cases, clients should run while the host machines are in use

Table 4.2 EA Requirements imposed by volunteer compute nodes' properties. () indicates row in Table 4.1

A. The EA must handle irregular compute availability, be able to cope with clients suddenly becoming ready or unavailable and be able to cope with asynchronous data communication (1,4)
B. The Evolutionary Engine footprint must be small (2)
C. The EA must be able to incorporate new islands while executing. It must continue to execute and utilize available nodes while not blocking to wait for unavailable others (3)
D. Migration of information between EA islands (Evoloutionary Engines in ECStar) must occur without the need for island to island identification (1,5)
E. The EA must robustly run with many islands exchanging information
F. The EA must not rely on running iterated experiments (4)
G. The EA, while still using all fitness cases, must only reference a few at a time and all transactions with the servers should be lightweight (5)
H. The EA should minimize its CPU (and memory) utilization so as not to disrupt host machine's main processing priorities (6)

requirements of an evolutionary algorithm system which ensue from various properties in the Properties table. It cross references requirements to properties using the Property table's item numbers. Finally, Table 4.3, the Design Choice table, enumerated with roman numerals, cross references design choices to multiple requirements in the Requirements table.

Scale Related Design Rationale

We now discuss the features which are key in allowing ECStar to execute continuously at a very large scale.

Resources Federation

The Evolutionary Coordinator is a source of two potential bottlenecks. The archive may become too large to manage if there are a lot of Evolutionary Engines and the bandwidth of the channels into the Evolutionary Coordinator used by the Evolutionary Engines may become over-subscribed. While it would be possible to distribute the Evolutionary Coordinator itself to address load balance and archive management, this solution would create another bottleneck due to numerous reads and writes to the distributed archive.

Instead, ECStar *federates* the functions of the Evolutionary Coordinator, see Fig. 4.2. An Evolutionary Coordinator can be down-chain to another Evolutionary Coordinator, acting as an Evolutionary Engine minus the evolutionary process itself. Down-chain Evolutionary Coordinators control and interact as Evolutionary Coordinators with their down-chains, which can be Evolutionary Engines, or

Table 4.3 ECStar's design choices following from its requirements. () indicates row in Table 4.2

I.	The Evolutionary Engines, coded in C, occupy a small footprint. Their stats can be quickly saved and restored (B)
II.	The Evolutionary Engines save their state frequently, they obtain fitness cases from the fitness data server in small quantities and execute the entire population on them before requesting more (A,B)
III.	There is no sequence related bottleneck in the Evolutionary Coordinator logic. It never blocks which allows it to always respond to any Evolutionary Engine graduate or migrant report. If it is busy, it harmlessly rejects the report and the reporter backs off before retrying (A)
IV.	The Evolutionary Coordinator uses a relative fitness metric when comparing individuals during archive management to address the fact that individuals are evaluated at different speeds. It also uses archive layering to impose a (reasonably) fair comparison between migrants in terms of fitness case exposure (A)
V.	The Evolutionary Engines are booted with configuration information that informs them of the Evolutionary Coordinator's and Data Server's IPs. They use this information to integrate with the current system. They do not need IPs of the other Evolutionary Engines because they communicate with them through the Evolutionary Coordinator (C,D)
VI.	Random migration is directed by the Evolutionary Coordinator (C,D)
VII.	Distributed, random fitness case evaluation defines interim solutions in terms of partial fitness as well as evolutionary progress. Neither the Evolutionary Engine or Evolutionary Coordinator have to maintain exact knowledge of which fitness cases an individual has been evaluated (E)
VIII.	No restart is required to run parallel experiments and they are integrated using federated resources (F)
IX.	The Evolutionary Engines use age layering and an elite pool to handle migrant guests of different ages. This prevents a migrant from being eliminated from providing genetic material too soon (D)
X.	Evolutionary Engines run at lower priority than host processes. The Evolutionary Engine can be stopped, paused, or restarted, and state files can be deleted from a container runner (GFRunner) or remotely by the server

Evolutionary Coordinators themselves, but they act as Evolutionary Engines to their up-chains. Each down-chain Evolutionary Coordinator maintains its own local archive, reducing the load on the top Evolutionary Coordinator's archive, as well as reducing bandwidth requirements. Note that Evolutionary Engines and down-chains of Evolutionary Coordinators need to exclusively communicate with their assigned up-chains. This is because the archives are federated themselves, and material received from these archives needs to be reported back to them once aged, and would be missing from other archives.

Evolutionary Engine State Preservation with State Migration

A large computation executing over a long duration needs to be able to preserve its state if interrupted. This state must be quick to restore (and even modify before a restart). It needs to withstand a reboot or even prolonged shut-down of its host. While results are inevitably lost in these circumstances, they should be minimized.

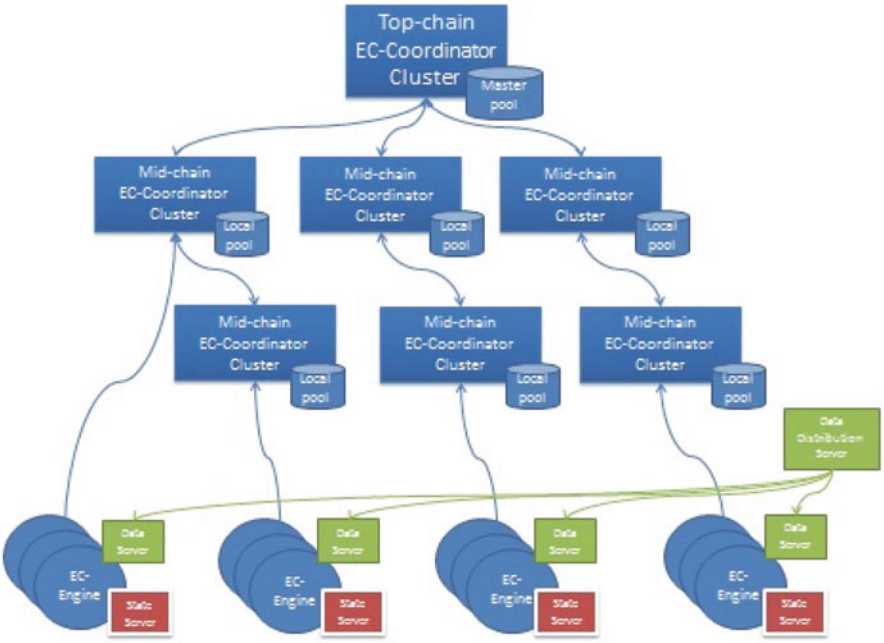


Fig. 4.2 Resource federation in ECStar

In this respect, ECStar can serialize and save the Evolutionary Engine population and the whole state of the evolution at intervals that are specified in an Evolutionary Engine's configuration in what is called the *state file*. Further, this state can be remote from the Evolutionary Engine, a local state-server, making it *stateless*. This makes the large, long computation robust to lost nodes. When a Evolutionary Engine comes online for the first time, or if it is ordered to restart by its up chain Evolutionary Coordinator, it can request a state file from the local state-server and continue processing from that state onwards. This exploitation of a state-server also provides more management control of the Evolutionary Engines, especially for adding or removing them.

Unique IDs and Snapshots

Recall that the Evolutionary Coordinator cannot send an individual to only one Evolutionary Engine in case that Evolutionary Engine never responds (it is a volunteer after all). Therefore, it sends a copy of an individual while keeping it in the archive in order to maintain selection pressure (more details on this follow in section "Archive Selection Pressure Through Alive/Dead State"). Scaling introduces the issue of tracking these replicated individuals as they return in an unpredictable order. To address this, the Evolutionary Coordinator, for each

individual, creates unique Evolutionary Engine identifiers and prefixes them to unique identifiers assigned by the original Evolutionary Engine. This makes all individuals distinguishable both locally and globally. When an Evolutionary Engine receives a migrant, it creates a copy called a *snapshot*. Then after evaluating the original individual on a number of fitness cases, it reports back both the individual with updated fitness and the snapshot to the Evolutionary Coordinator. This allows the Evolutionary Coordinator to easily recognize and consolidate the differences between the two with its own current version of the individual. As a result, when ten Evolutionary Engines evaluate a gene on ten fitness cases, the Evolutionary Coordinator merges those reports so that the individual's fitness is based on the hundred cases.

Fitness-Age Normalization and Layering

No ECStar Evolutionary Engine fully evaluates an individual on all the fitness cases. This partial evaluation implies a need to compare individuals that more or less match in terms of being evaluated on the same number of fitness cases. This is addressed through age-layering, see Hodjat and Shahrzad (2012) and Hemberg et al. (2013b). Layering (at either Evolutionary Engine or Evolutionary Coordinator) allows individuals to only compete with others of the same level of fitness case evaluation, based on their relative fitness.

Archive Selection Pressure Through Alive/Dead State

It is important for the Evolutionary Coordinator to direct selection pressure among the individuals it receives. From Hemberg et al. (2013a) we know that the coverage of the training fitness cases and the symmetry of the training fitness cases of individuals due to the evaluation of random training fitness cases increases as the individual solution is exposed to more cases. Parameterization supports changing the size of the layers on the Evolutionary Coordinator, to tune the pressure and address the issue of asymmetric comparisons (noisy) between individual solutions that have seen only a few training fitness cases. These parameters can be adjusted as ECStar executes.

Individuals that are dispatched by the Evolutionary Coordinator as migrants to Evolutionary Engines are not deleted from its archive. Retaining them maintains selection pressure in the context of returning migrants. However, these individuals may be displaced by new migrants with superior fitness. This creates a timing glitch: when the migrants of an individual return, it might have been displaced. The fitness of the displaced individual must be updated and it must be allowed to compete for entry into the archive. To resolve the glitch, displaced individuals are retained but marked as "dead" and called "ghosts". Ghosts are kept in the archive until the ratio of "dead" genes is too high, then the archive is flushed of them.

Application Specific Interpreter in Evolutionary Engines

Each Evolutionary Engine is deployed with a very small, application specific, language interpreter. This allows instructions to be passed, via a configuration message passed by the Evolutionary Coordinator, to be executed and avoids the need to replace the software on the Evolutionary Engine. It makes the Evolutionary Engine a general executable which in turn is 100 % configurable by the Evolutionary Coordinator via the configuration messages. The interpreter supports signals to restart, flush the Evolutionary Engine's population, or stop before a backward incompatible software upgrade is loaded. A configuration message can also carry:

- New evolutionary algorithms settings, such as mutation rate, population size, maturity age, etc.
- A meta description of a new fitness objective
- A definition of new discrete intervals for feature discrimination
- A definition of new features in data packages

Integrated Data Feature Server

Because it executes uninterrupted for a long time, ECStar incorporates the ability to perform online feature selection. This function is controlled by the Data Feature Server.

Two feature sets are managed by a database table which counts the number of handshakes (migrants reported to the Evolutionary Coordinator to be added or merged) which were requested and the number of handshakes which were fulfilled. The Data Feature Server looks at these ratios, an absolute threshold and improvement in fitness. When the improvement rate declines and the number of requested handshakes is more than C_1, the feature sets are ready to merge and a new set will be created. The merged feature set uses the union of features of both sets. The individuals are merged in the Evolutionary Coordinator and marked as coming from the merged set. The new set uses a fixed collection of features and chooses others randomly. Merging and new feature set creation is coordinated via a properties file referenced by the Evolutionary Coordinator.

The Evolutionary Engines receive the definitions of the new feature set from the Evolutionary Coordinator via a configuration message. This minimizes the effort spent on updating the Evolutionary Engine. Configuration messages propagate through the normal Evolutionary Engine Evolutionary Coordinator communication and take a while to completely reach all Evolutionary Engines.

Security

An ECStar system instance is running on public networks and the integrity of Evolutionary Engines is not assured. Ensuring security increases the computational cost of data transfers but is vital. Detecting injection of erroneous solutions is

handled by consulting the identifiers on solutions and by executing out-of-sample tests. Sensitive data is encrypted and serialized. The Evolutionary Coordinator, Data Server and client-state servers are also required to reside behind firewalls.

Summary

Some of these scaling design choices function below the level of the attention of the engineer supervising an ECStar system instance. Others are exposed via parameterization or for provisioning. This becomes clear in the next section on System Instance Management.

5 ECStar System Instance Management

For an example of a use case for a deployment and maintenance of a run with 10,000 Evolutionary Engines, see Fig. 4.3. ECStar executes over three phases:

Phase 0: Local experiments The initial step:

Local experiments On a cluster local experiments are completed before in preparation to scale. This includes iterating to decide upon an efficient format for data packages, features, representation parameters, objective function and

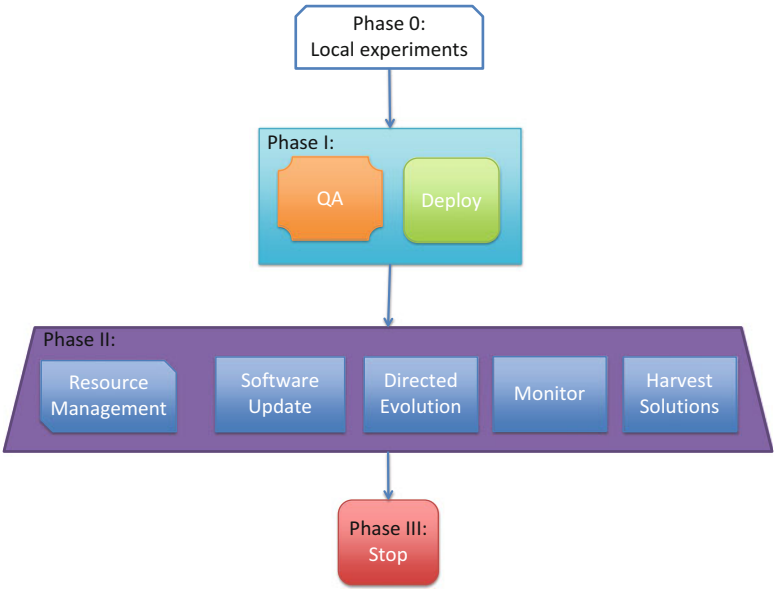


Fig. 4.3 ECStar system management phases

execution parameters in Evolutionary Engine and Evolutionary Coordinator, see O'Reilly et al. (2012). The output of the process are source and configurations for the Evolutionary Coordinator and Evolutionary Engine.

Phase I: Large-scale deployment preparation There are two preparation steps:

Quality Assurance The binaries for the Evolutionary Engines must pass a Quality Assurance (QA) checking whether it is able to run within the limitations of the client's footprint, and not disrupt the volunteer while it is not idle. This includes 24–48 h execution and profiling of the binary.

Deployment After QA, the deployment is initiated to a controlled environment of 1,000 Evolutionary Engines for a couple of days. The Evolutionary Coordinator Servers and Data Servers are configured to handle the initial Evolutionary Engine load. Finally, the Evolutionary Engine code is distributed and deployed globally. This can take a week. The management of resources continues during Phase II.

Phase II: Long-term operation As soon as an Evolutionary Engine has received the binary code and has idle cycles it can start to compute. When the network capacity allows, it can request training cases from the data server. After enough time processing with idle cycles it tries to report results as snapshots of individual solutions to the Evolutionary Coordinator server. If the network capacity allows the Evolutionary Coordinator receives them. When there is high throughput the Evolutionary Coordinator is usually configured to tolerate a wave of 10,000 requests per minute. There are a number of steps:

Software update In order to update the Evolutionary Engine a new binary or parameter settings would need to be distributed. This is discouraged since it could take a week. Moreover, when updates are done on the servers it must be taken into account that there are Evolutionary Engines which are using data from previous versions, thus the turn around for updates is the same as for a new deployment. See section "Updating a Deployed ECStar System Instance" for more details on how this is nonetheless possible.

Resources management By adding or removing Evolutionary Engines the resources can be managed during the run.

Monitor We are continuously monitoring the state of the hardware and the network as well as the solutions at the evolutionary coordinator.

Harvesting The harvesting is performed by filtering the solutions at the Evolutionary Coordinator or out-of-sample tests. See section "Updating a Deployed ECStar System Instance" for more details.

Directed Evolution The design of ECStar allows a directed evolution of the system. Promising combinations within the system can be allowed more resources, while worse can be terminated.

Phase III: Stop The final step:

Stop Halting the run takes a while as well, e.g., a couple of days. It is important to make sure that all the Evolutionary Engines stop sending solutions to the Evolutionary Coordinator.

Table 4.4 Components of ECStar and their access and maintenance

Component	Access	Maintenance
Evolutionary Engine	Unpredictable	Sporadic & delayed
Evolutionary Coordinator	Predictable	Continuous & "hot swap"
Data Server	Predictable	Continuous & "hot swap"
Data Feature Server	Predictable	Continuous & "hot swap"

Monitoring a Deployed ECStar System Instance

When ECStar is up and running there are multiple automatic monitors in action. There is basic IT monitoring of database load and connections. This is mainly preventive and can identify where the Evolutionary Engines have slowed down.

Even more important is the monitoring of the Evolutionary Coordinators. The improvement of fitness is constantly monitored to warn when it trails off. In addition, individuals are constantly filtered and harvested from the Evolutionary Coordinator and tested on out-of-sample data for generalization. The convergence of the clients are monitored by looking at the ratio of the number of handshakes between the Evolutionary Engine and the Evolutionary Coordinator to the accepted handshakes (individuals) at the Evolutionary Coordinator. The fitness function is used to drive the evolution, but there can also be other domain specific properties in the individual which are interesting. The monitoring can be configured to find individuals with these properties in the Evolutionary Coordinator archive as well.

Updating a Deployed ECStar System Instance

The state of a Evolutionary Coordinators, Evolutionary Engine, Data Server and Data Feature Server are: inactive (Off) and active (On). When a component is active the versions of all the components need to be compatible. Changing the active version of the components can be done interactively during runtime or with a restart. If the component is restarted, a new parameter file or a new binary can be used. The runtime changes require the fewest steps, but they will still result in the system being in a state with hetereogenous versions.

Table 4.4 shows the accessibility and the mode of maintenance of each component. Each of the components in ECStar has a configuration file. This reduces the need to recompile the components and instead a restart of the component is sufficient to make changes.

The Evolutionary Coordinator, Data Server and Data Feature Server are the components which are the most accessible, since they are currently not part of the volunteer compute. When there are changes made that requires feedback from the Evolutionary Engines then there will always be the turn around of all the clients being available and updated. The simplest change is when it is pushed out via the servers without stopping the run. It becomes slightly more complex when the clients need to be stopped and updated.

ECStar's design deals with the appearance of Evolutionary Engines after long periods of idleness. This situation is handled by the check for version comparability between Evolutionary Engine and Evolutionary Coordinator when the Evolutionary Engine restarts. When incompatible Evolutionary Engines contact a Evolutionary Coordinator they are terminated by a kill signal sent from the Evolutionary Coordinator.

The federation of Evolutionary Coordinators allows the archive size to be scaled according to the number of Evolutionary Engines. In addition, federation allows a tiered evolution where different tiers control the selection pressure and supply of individuals. Finally, the federation allows more control for ECStar, since the Evolutionary Coordinators are under direct control.

The servers can be "hot swapped" by simply redirecting the IP of the server to a duplicate, while the original server is updated. The Evolutionary Engines are not so easy to change, since they cannot be accessed at will, but there is access to the state server. The loss of the Evolutionary Engine local data is therefore limited to the state file save interval.

The multiple objectives in the fitness function can be altered when ECStar is deployed. This has to be done with care because of the interaction of the old population under the new fitness objectives. It could wipe out the old population with no genetic legacy left behind or it may allow the population to converge to local minima.

6 Directed Evolution

The properties which imply ECStar is impressive and exemplary by contemporary measures of scale and computational evolutionary complexity also make ECStar a square peg in the evolved round hole of GP research expectations. This is because it is practically prohibitive to repeatedly execute ECStar at least 30 times to obtain performance characterization which carries statistical significance. Like nature, ECStar won't pass a t-test; its scale, inherent asynchronicity and complexity dictate that its "tape" can only be played once.

Instead, a directed evolution approach can be taken when studying ECStar. This involves changing parameter settings during the run, adding and removing Evolutionary Engines and updating the software. The reset is never complete as with a standard GP system, it is more of a nudge of the current population. This requires either the population to be sufficiently diverse, or possess the ability to inject diversity, see Hemberg et al. (2013a).

In addition, due to its robust setup ECStar allows investigation of what happens in the system when a change is not yet propagated to all the parts of the system, i.e. the components are hetereogenous. In other words, there will be intermediate effects of system heterogeneity.

7 Summary

In summary, we believe it inevitable that more and more GP (and EC) systems will become distributed on much larger scale than is presently the case. This transition will be driven by operational costs of clouds and grids decreasing plus increasingly bigger applications with training data from so called "Big Data" repositories. This move from somewhat small-scale distribution in the order of ten's or hundred's to thousands through to millions is around three orders of magnitude in terms of computing nodes. ECStar addresses this scale up in the following ways:

- It has a hierarchical federated resources design which creates a hierarchy of Evolutionary Coordinators to handle the scale of the volunteer compute node and uses the Evolutionary Engine as the base. (Section "Resources Federation")
- It explicitly deals with the inevitable and higher risk, introduced by running so many nodes, of losing computed information by means of Evolutionary Engine state preservation, state migration and a strategy of polling for required data in frequent but small quantities. (Section "Evolutionary Engine State Preservation with State Migration")
- It has an explicit means of resolving asynchronously computed work on the same datum (i.e. rule set) that occurs because of the unpredictable availability of its Evolutionary Engines by employing snapshots and unique IDs. (Section "Unique IDs and Snapshots")
- It uses fitness normalization and layering to deal with the different types and value of fitness cases that have been evaluated by each solution. (Section "Fitness-Age Normalization and Layering")
- It maintains archive selection pressure by means of ghosts in the archive and flushing periodically. (Section "Archive Selection Pressure Through Alive/Dead State")
- It integrates a meta language to simplify deployment to the Evolutionary Engines. (Section "Application Specific Interpreter in Evolutionary Engines")
- It incorporates the ability to perform feature selection on the data. (Section "Integrated Data Feature Server")
- It deals with security via encoding data, compiled binaries, firewalls and recalculation of fitness at the Evolutionary Coordinator, as well as out-of-sample evaluation of the top solutions. (Section "Security")

ECStar also introduces explicit operational means of addressing its execution complexity which arises from using so many resources in an asynchronous way the time cost to deploy so many resources the need to update deployed resources. It has facilities for: monitoring, see section "Updating a Deployed ECStar System Instance", harvesting, see section "Monitoring a Deployed ECStar System Instance" and software updating, see section "Updating a Deployed ECStar System Instance".

ECStar offers a digital version of directed evolution. It has strong commonality with Artificial Life systems because, like them, it is a single execution process, rather than a restart-rerun platform. It is NOT an ALIFE system because it is applied

to solving a problem, for purposes much more typical of evolutionary computation. It is applied and GP in practice rather than a digital experiment being studied for its behavior to validate and explicate a biological system.

We embrace a new "zen" of design with this unique evolutionary system. ECStar allows us to try to do direct evolution in a system which you do not completely turn off.

Acknowledgements We acknowledge the generous support of Li Ka-Shing Foundation.

References

Anderson D (2004) BOINC: a system for public-resource computing and storage. In: Proceedings of fifth international workshop on grid computing, Pittsburg, 2004. IEEE/ACM, pp 4–10. doi:10.1109/GRID.2004.14

Bedau MA (2003) Artificial life: organization, adaptation and complexity from the bottom up. Trends Cogn Sci 7(11):505–512

Bennett III FH, Koza JR, Shipman J, Stiffelman O (1999) Building a parallel computer system for $18,000 that performs a half peta-flop per day. In: Proceedings of the genetic and evolutionary computation conference, Shanghai, vol 2, pp 1484–1490

Cantu-Paz E (2000) Efficient and accurate parallel genetic algorithms, vol 1. Kluwer, Boston

Crainic TG, Toulouse M (2010) Parallel meta-heuristics. In: Handbook of metaheuristics. Springer, Berlin/New York, pp 497–541

Desell T, Anderson DP, Magdon-Ismail M, Newberg H, Szymanski B, Varela CA (2010) An analysis of massively distributed evolutionary algorithms. In: 2010 IEEE World Congress on computational intelligence, Barcelona, pp 18–23

de Vega FF, Olague G, Trujillo L, Lombraña González D (2012) Customizable execution environments for evolutionary computation using boinc+ virtualization. Nat Comput 1–15

Gonzalez DL, Laredo JLJ, Vega FF, Guervas JJM (2012) Characterizing fault-tolerance in evolutionary algorithms. In: Fernandez de Vega F, Hidalgo Perez JI, Lanchares J (eds) Parallel architectures and bioinspired algorithms, studies in computational intelligence, vol 415. Springer, Berlin/Heidelberg, pp 77–99. doi:10.1007/978-3-642-28789-3-4, http://dx.doi.org/10.1007/978-3-642-28789-3-4

Hemberg E, Wagy M, Dernoncourt F, Veeramachaneni K, O'Reilly UM (2013a) Efficient training set use for blood pressure prediction in a large scale learning classifier system. In: Sixteenth international workshop on learning classifiers systems, Amsterdam. ACM, New York

Hemberg E, Wagy M, Dernoncourt F, Veeramachaneni K, O'Reilly UM (2013b) Imprecise selection and fitness approximation in a large-scale evolutionary rule based system for blood pressure prediction. In: Proceedings of the fifthteenth international conference on genetic and evolutionary computation conference – GBML, GECCO'13, Amsterdam. ACM, New York

Hodjat B, Shahrzad H (2012) Introducing an age-varying fitness estimation function. In: Genetic programming theory and practice x. Springer, New York

Langdon WB (2012) Distilling genechips with GP on the emerald GPU supercomputer. ACM SIGEVOlution 6(1):16–22

Merelo J, Mora A, Fernandes C, Esparcia-Alcazar AI, Laredo JL (2012) Pool vs. island based evolutionary algorithms: an initial exploration. In: 2012 seventh international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC), Victoria. IEEE, pp 19–24

O'Reilly UM, Wagy M, Hodjat B (2012) Ec-Star: a massive-scale, hub and spoke, distributed genetic programming system. In: Genetic programming theory and practice x. Springer, New York

Rivest R (1987) Learning decision lists. Mach Learn 2(3):229–246
Scheibenpflug A, Wagner S, Kronberger G, Affenzeller M (2012) Heuristiclab hive – an open source environment for parallel and distributed execution of heuristic optimization algorithms. In: 1st Australian conference on the applications of systems engineering ACASE'12, Sydney, p 63
Smaoui M, Garbey M (2013) Improving volunteer computing scheduling for evolutionary algorithms. Future Gener Comput Syst 29(1):1–14
Tomassini M (2005) Spatially structured evolutionary algorithms. Springer, Berlin/New York

Chapter 5
Grounded Simulation: Using Simulated Evolution to Guide Embodied Evolution

Conor Ryan, Joe Sullivan, and Barry Fitzgerald

Abstract Flash memory is a type of non-volatile memory that is being used at an ever-increasing rate in enterprise level storage devices. It is extremely fast (compared to magnetic disks), requires low power and generates very little heat. Unfortunately, it has a relatively short life span, as it wears out over time, compromising its ability to retain data. Previous work has used GAs to tune the control parameters of Flash to trade *retention* for *endurance* and, although successful, was prohibitively costly in terms of time, as all testing had to be done in hardware. This chapter describes the next stages in this work, which use a combination of simulated (faster, but less reliable) and embodied (slower, but more reliable) evolution to produce internal parameter sets for Flash memory that increase the endurance by an order of magnitude while still maintaining an industry accepted level of retention.

Keywords Flash memory • Modeling

1 Introduction

Flash Memory, long associated with consumer electronics such as cameras, memory sticks and MP3 players, is now shaping up to replace magnetic disks as the medium of choice for enterprise level storage. Disks that use Flash memory are known as Solid State Disks (SSDs).

Flash is extremely fast (compared to magnetic disks), requires low power and generates very little heat. Unfortunately, Flash has a relatively short life time

C. Ryan (✉)
University of Limerick, Limerick, Ireland
e-mail: Conor.Ryan@ul.ie

J. Sullivan • B. Fitzgerald
Limerick Institute of Technology, Limerick, Ireland
e-mail: Joe.Sullivan@lit.ie; Barry.Fitzgerald@lit.ie

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_5,

(*endurance*), measured in *program/erase (p/e) cycles*, i.e. the number of times that a device can be written to, because each cycle slightly damages the device by placing charge on the storage locations. Lowering the current reduces the damage and increases the lifetime, but this has an impact on the non-volatility *retention* of the device, as the lower the charge, the more volatile the memory is.

This is rarely an issue at the consumer level, where Flash (often in the form of USB memory sticks or camera memory cards) is mainly read from rather then written to, meaning that manufacturers can reliably specify the retention of devices as 5–10 years. However, at the enterprise level, *endurance* is more important, as the rate with which the devices are written can cause them to wear out extremely quickly. Flash chips generally of interest in Enterprise level SSDs have an endurance of less than 5,000 cycles per location, with data retention of just 1 year. Successive generations of Flash increase the number of bits stored by each cell to increase storage density, but this comes at the expense of reduced endurance. The next generation of Flash, known as triple-level cell (*TLC*) (see Sect. 2) devices, have endurance ratings of just 1,000 cycles. The endurance issue is so grave that it is common for SSD vendors to have upwards of 100 % redundancy in their disks, so that, as parts wear out, their wear leveling software uses other parts.

Because of the way Flash works, see Sect. 2, endurance can be increased by reducing retention, but this is a delicate balancing act, as too little endurance means that one of the most attractive features of Flash, the non-volatility, will be compromised too much. However, if the endurance is too low, then the cost of disks that employ Flash will be prohibitively high, as they will have to resort to redundancy. This is a difficult and time consuming task, and it is generally accepted that no optimal method currently exists.

The trade off is not a simple one, however. Modern Flash memory chips are controlled by a large number of control registers, which govern aspects such as voltage levels, write times, etc. It is not uncommon to see 50+ registers controlling a device. The manner in which these registers is programmed directly affects both the endurance and the retention. Clever manipulation of the registers can increase the endurance without compromising the retention too much. Setting the registers is something of a black art, and it is not uncommon for engineers to simply use similar values to their last model, without trying to tune them to any great extent. As if the enormity of the search space wasn't daunting enough, the only way to test register settings is by exercising a location to destruction, which is a slow process, often taking days or even weeks.

Previous work by Sullivan and Ryan (2011, 2007) used a Genetic Algorithm to set these parameters for an earlier type of Flash (NOR Flash), and, although a three- to five-fold increase in endurance was reported, the experiments were prohibitively long because the only way to reliably test the quality of solutions is to run the hardware to destruction. This chapter describes the next stages in this work, which use a combination of simulated (faster, but less reliable) and embodied (slower, but more reliable) evolution to produce internal parameter sets for Flash memory that increase the endurance by an order of magnitude while still maintaining an industry accepted level of retention.

This paper presents a methodology to automatically tune these register settings using a combination of Genetic Algorithms and Genetic Programming. We introduce purpose built hardware which performs a combination *embodied evolution* (GAs) and simulated evolution (using search spaces generated by GP), that is, the individuals being tested are tested on the actual hardware that the final, best of run, individual will be required to operate on. We then demonstrate that using this hardware, along with some modifications to the GA which will be of use to anyone testing on hardware, we experience increases in endurance of an order of magnitude.

The paper is laid out as follows. Section 2 gives a background to this work, including a description of earlier work. Section 2 describes Multi-Level Cell (MLC) NAND Flash, the specific type of Flash memory we are concerned with. We describe the evolutionary experiments in Sect. 3, while Sect. 4 is concerned with how the search space was trimmed down, and Sect. 5 details the results. Finally, Sect. 6 gives some conclusions and pointers to future work.

2 Background

Until very recently, SSDs were prohibitively expensive, but the technology is developing incredibly quickly. SSD costs have dropped from $45 per GB in 2005 to $1.97 per GB in 2010, with projections of $1.05 per GB by 2013 (Niebel 2010, p. 3; Singer 2005). At the same time, the market is predicted to grow from around $21.5–$38.9 billion during period 2010–2015 (Niebel 2010).

Virtually all data storage and many of the larger electronics companies are involved in SSD and Flash development, including Samsung, Cisco, EMC^2, Dell, Google, Intel, IBM, etc. and the pressure is such that the complexity of NAND Flash – the type exclusively used in enterprise level storage (LLC 2011) – is shrinking faster than predicted by Moore's law (Abraham 2010, p. 4).

Shrinking geometry is accompanied by MLCs, in which devices can store 2 or more bits per cell. This greatly increases the capacity, but MLC devices are considerably more complex and error prone (Tressler 2010), with the result that the level of ECC (Error Correcting Codes), the ability to detect and correct errors, is dramatically increasing (Abraham 2010, p. 6; Tressler 2010). ECC slows devices due to its complex computational nature, and this is a particular problem for MLC devices, the speed of which is further compromised as the number of bits per cell increases (Abraham 2010, p. 8). Furthermore, the more ECC a device uses, the less space it has for general storage (Hellmold 2010, p. 4), thus reducing the cost benefits of higher numbers of bits per cell.

Multilevel Cell NAND

The most common types of Flash operate by storing charge on a floating gate transistor; the *presence* of charge corresponds to the *absence* of the logical bit being stored on the transistor. Reading the value stored on the gate simply involves

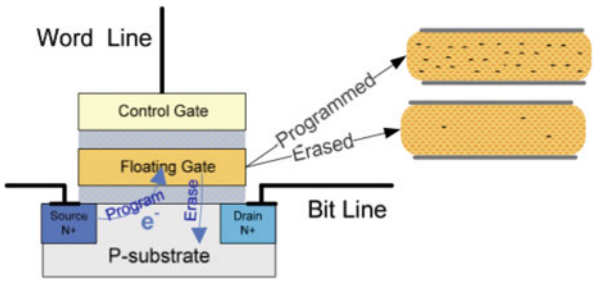


Fig. 5.1 A floating gate cell from Flash memory. Charge flows from the P-substrate to the floating gate when programmed

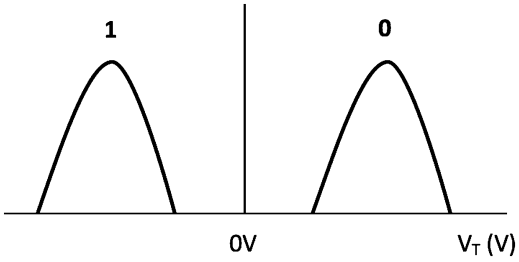


Fig. 5.2 Single Level Cell (SLC) Flash memory. If the voltage is above a particular threshold (the *read pole*), the value is considered 0; if it is *below* the threshold, the value is 1. V_T is the *voltage threshold* and is a rough indicator of the amount of voltage in the gate

detecting whether or not there is a charge; however, the setting/unsetting of the transistor requires charge to be either placed or removed from the floating gate. Both of these require comparatively high voltages to be applied to the transistor to force the electrons through the insulating layers, which degrades the transistor each time it is erased or written to. Each is followed by a test. If the operation failed, it can be retried several times, although this slows operations (Fig. 5.1).

The amount of voltage in a gate is often referred to as the *voltage threshold* or V_T, and the health of a device is sometimes measured in the V_T distributions, which indicate how much of a spread of voltages there are across all (or many) gates in the device. Although an accurate measure of the health of a device, measuring the V_T is a time consuming task during which the device must be offline; it isn't a practical way of observing the device in the field. V_T is typically measured across many cells within a device, hence giving a distribution (Fig. 5.2).

There are two different kinds of Flash, NOR and NAND. In NOR Flash, individual cells are linked in parallel permitting individual (random) access, while in NAND, the cells are linked in series meaning they can only be addressed as groups. They can, however, be packed far more densely, making them cheaper. Enterprise level SSDs exclusively use NAND Flash.

With the introduction of MLC NAND, bits are packed even more tightly, with 2 or 3 bits per cell. Individual (combinations of) bits are decoded by having multiple thresholds, corresponding to, for example 00, 01, 10 and 11 in 2 bit MLC. This higher density makes MLC NAND prone to errors and much faster (over an order of magnitude) to wear out than NOR, and the situation gets worse as more bits are packed into each cell.

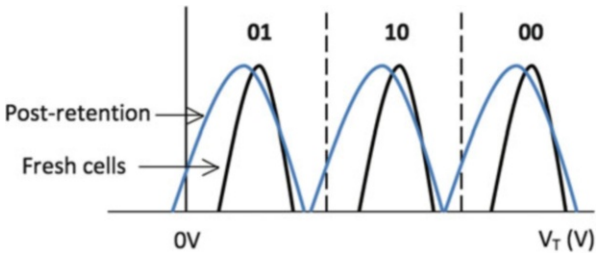


Fig. 5.3 The impact of retention on cells. The longer the cells remain unrefreshed (even when powered on) the wider and more to the left the distributions go, making MLC more prone to errors and giving it less retention time than SLC

Errors can be divided into two broad categories: *Cycling* errors and *Retention* errors. Cycling errors begin to appear through repeated use of the devices as the floating gate begins to deteriorate. In general, the higher the voltages that are applied to the gate, the faster the errors appear. These errors manifest themselves in a lack of precision, i.e. it becomes increasingly more difficult to write an exact voltage to the gate.

This is less of a problem with SLC devices, as it is simply a matter of having either relatively high or low voltages, but, with MLC devices, where precision is more important because there are multiple thresholds, it is possible for the device to write the wrong logic values into a location as a result of incorrectly writing the wrong voltage.

The second category, retention errors, is illustrated in Fig. 5.3. In this case, errors appear over time when the device is not refreshed, as the voltage degrades due to electrons drifting off the floating gate. In this case, the V_T distributions spread out and drift to the left (lower voltages).

Once the distributions begin to drift across the thresholds, the number of errors begins to increase dramatically. The rate at which the cells degrade during retention is directly related to how aggressively (in terms of voltages and number of cycles) they were written earlier in life; the more aggressive, the faster the degradation. However, there is also a direct relationship between the level of voltage used and the duration of retention, as very small voltages, in a tight range, will have a shorter retention time.

It is difficult to quantify the impact that retention has on cells because, as is described later in Sect. 2, it depends on the health of the blocks, which is related to how much wear they have endured, the rate of wear, etc. Similarly, the level of degradation of contents is exponential because as the distributions drift across the read poles, there can suddenly be a massive jump in the number of errors.

To combat the short life, all sorts of extra computation and trickery is added to the SSDs that use NAND Flash. Typically this includes massive (from 50 to 100 %) redundancy as well as sophisticated ECC, which, in the latest drives, is capable of correcting over 40 bits per 512 byte sector. However, ECC is not a panacea. Although using more extends the life of the devices, it requires considerable computational power as well as extra storage, which comes at the cost of the main storage.

This is already a huge problem for SSD and Flash manufacturers. Although the cost per GB is coming down, it is still more than two orders of magnitude more than for magnetic disks (LythOs 2011), prohibitively high for most uses even at the enterprise level. The key factor driving the cost down is MLC, as it dramatically increases density. TLC devices are already available (although no **enterprise level** drive uses them yet); this will demand ever increasingly sophisticated ECC. TLC devices have much worse retention and endurance than MLC devices, because there are more read poles, giving less room for drift.

Logical vs. Physical Arrangements

The individual cells are not addressable in Flash. Instead, they are grouped together into logical units (sectors) for the purposes of ECC, which operates at the sector level. Several sectors make up a *page*, the smallest readable unit (typically 4–8 KB) in Flash, while pages are grouped together as *blocks*, the smallest *writable* unit. All our tests take place at the block level.

Tuning Parameters

The various ways in which the control parameters are set by the various manufacturers are closely guarded secrets; even the SSD manufacturers who built their devices from Flash are not privy this information. In fact, it isn't uncommon for the SSD manufacturer to not even know *what* registers exist, let alone what their values are.

Using a combination of engineer experience, testing and re-testing, followed by qualification (the process in which the manufacturer simulates the lifetime of the devices as final proof of their quality), often sprinkled with a healthy dose of gut feeling, the manufacturers produce a default set of parameters. This is a very time consuming process, as the only way to be fully sure that a parameter set works for the life time of the device is to cycle it to destruction.

However, not all silicon is created equal. The quality of the chips varies not only across batches, but even *within wafers*. To ensure that all their chips are capable of adhering to their specification sheet (that is, how many p/e cycles and how much retention they are capable of), their V_t is calculated. If the distributions don't match those of the expected baselines, the various parameters are adjusted until they do. This gives the manufacturer confidence that the chips will wear and degrade as expected.

This presents a difficulty for an Evolutionary system, however. This significant variation of the parameters means that it is unlikely that any evolved set

of parameters can reasonably be expected to work with every chip. To further complicate matters, the chips that are used to evolve the parameters will, by the very nature of the embodied evolution that is required, be destroyed. Section 3 describes how we overcome this issue.

Endurance vs. Retention

As mentioned in Sect. 2, the only way to properly test a device is to cycle it to destruction. It is actually more complicated than this, as the manner in which a device is cycled affects the measurements. Specifically, the *rate* (how fast the device is written to), the *temperature* of cycling (because higher temperatures help the device recover faster) and even the *data* being written are the key parameters. The data has a massive impact on the life of the device because certain patterns are more difficult to store than others, due to program disturb and cell-to-cell interference.

The former is the unintentional changing of the contents of a cell due to the programming of another cell, while the latter is a phenomenom where a cell's V_T is affected by the V_T of an adjacent cell. Both of these manifest themselves with the appearance of errors in the data.

Any test of non-volatile memory must involve a retention test. It is clearly not practical to wait for the specified retention period (typically 3 months–1 year) to determine if the device has retained its data. Instead, the aging process is accelerated by baking the devices at an elevated temperature. The temperature and duration of the bake are calculated using a well-known law called the Arrhenius equation.

According to the JEDEC specifications, the life time of a chip can be simulated as follows:

- Exercise the chip to maximum cycles (30,000 in our case) over a week at 85 °C. In the field, chips are typically exercised to their maximum cycle count over a much longer time period, allowing time for oxide charges to detrap between cycles, resulting in less damage accumulation. Cycling at elevated temperature accelerates this detrapping process and allows real-world usage to be simulated in a much short timeframe.
- Write data into the locations under test.
- Simulate retention time by baking the chip in an oven at 85° for 13 h. This is equivalent to the enterprise-SSD retention specification of 3 months at 40 °C
- Calculate the number of errors in each location under test

Any location that has fewer than 13,500 errors after this test is deemed to have passed. The test system, including environmental ovens and custom-built testers, is shown in Fig. 5.4.

Fig. 5.4 Test system, incorporating environmental ovens to control the test temperature, and custom-built testers which are controlled from a host computer and may be operated in parallel

3 Evolving Parameter Sets

The confidential nature of this work prohibits us from listing the actual registers for the device that this paper uses. However, we can say that the specification sheet of the device rates it for 3,000 p/e cycles and 1 year retention. This means that, even at 3,000 p/e cycles, the device should be able to maintain its contents for a year. As described in Sect. 2, the health of a location can be measured using the V_T, but, as this is quite time consuming, a faster method is to simply measure the number of errors detected. This has the advantage of being a point measure, which is more natural for EC type methods.

ECC is generally measured in bits per sector, for example, 13 bits per 512 KB sector. With 13-bit ECC available, a block in our device can have approximately 13,500 errors (assuming uniform distribution) and still be usable.

Our target was to modify the device so that it could maintain **3 months** retention at 30,000 p/e cycles. Initial tests with default chips suggested that they were capable of approximately 12,000 cycles and 3 months retention. Note that it is not surprising that the devices could produce more than the 3,000 cycles described in the specification sheet once the retention constraint was relaxed.

Targeting Registers

The device contains 63 registers. These can be roughly divided into three categories: **read**, **write** and **erase** registers. Erasing is significantly slower than either **read** or **write**, with a typical erase operation taking 2.7 ms, versus 900 µs for a typical program operation and 65 µs for a random read operation. For this reason we decided not to modify any of those registers, because if the chip speed dropped below a predetermined level, the chips would be too slow for the SSD market, and it wouldn't matter how good their endurance or retention was. Several of the registers (and this is common across all Flash memory chips) contain *retry* values; that is, how many times to retry an operation that has failed (Baek 2014). Typically, if the device fails to write a value, for example, it will try again almost immediately. If, after several (the actual value is determined by the manufacturer) attempts, it has still failed, that location is marked as bad and not used again. None of these registers were modified in the search.

This left us with 14 8-bit registers, which gives a total of 4,398,046,511,104 different possible parameter sets. The hardware takes around 1.5 h to test a location to destruction, so exhaustive testing would take 274,877,906,944 days, which is approximately 753,090,156 years, somewhat longer than we would like to spend on this endeavour.

4 Reducing Testing Time

We took a four pronged approach to reducing this time. This was:

1. Trimming the parameter space
2. Parallel execution
3. Predictive testing
4. Simulated evolution

Trimming is an often used GA technique to manually cut down the search space by preventing known bad values and/or parameter combinations from entering the population. For example, three of the parameters control where the read poles corresponding to each of the value pairs are. For example, as can be seen from Fig. 5.5 clearly, the pole between 11 and 01 needs to be to the left of the one between 01 and 10. Some of the trimming was the application of common sense constraints such as these, but much of it was concerned with where the *write poles* were placed. This was non-trivial because there is not a direct one-to-one correspondence between the read and write poles.

Parallel testing is an often used tool in EC. The cost of the hardware in this case prohibits the sort of massive parallelism that GAs sometimes enjoy, but a 16 site parallel machine was constructed. This test system is illustrated in Fig. 5.6.

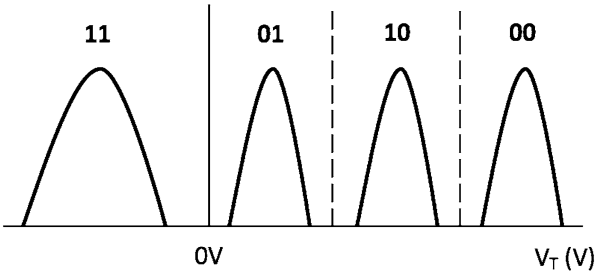


Fig. 5.5 Multi-Level Cell (MLC) Flash memory. In this case, there are four possible values, 11, 01, 10 and 00, packed into the same voltage range, separated by three read poles

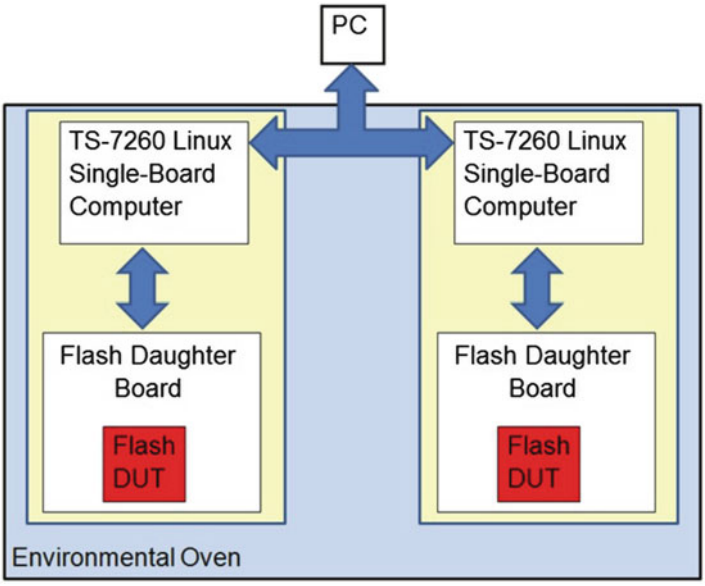


Fig. 5.6 Block diagram of the test system showing two sites. As they are connected using ethernet, the system is very scalable

Each site comprises a Linux-based single-board computer, connected to a daughter board which contains the device under test. Testing is controlled by a host computer which is connected to each site via ethernet.

Recall from Sect. 2 that not all chips are created equal; some chips have thousands of errors at 30,000 p/e cycles, while others have tens of thousands of errors at the same cycles. This means that the same individual can give radically different results when tested on different hardware. To overcome this, all measurements are *relative to baselines*. That is, before any EC runs are conducted, several locations in each chip are tested to destruction, and the average number of errors at the desired level (30,000 in our case) is calculated.

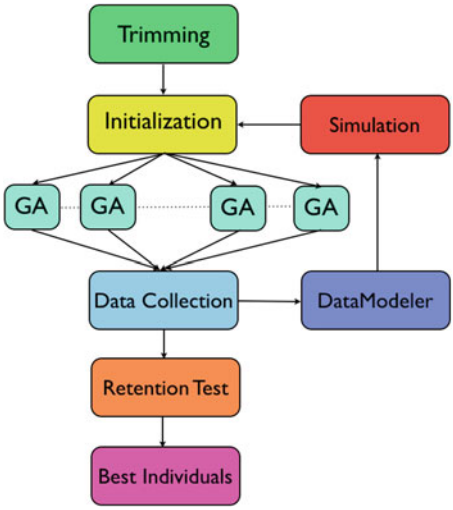


Fig. 5.7 Algorithm for experiments. Each independent run is harvested for a selection of individuals, which are then sent to the retention test

Thus, when an individual is tested, what is returned is its number of errors relative to the baseline, not an absolute count of errors. This means that we calculate its *improvement* (or disimprovement) over the default settings.

Predictive testing was introduced because embodied testing is an expensive business. With 13 bit ECC, approximately 13,000 errors can occur (assuming they are uniformly distributed) in a block (recall that a block is made up of many pages, which is made up of several sectors, the level at which ECC operates). However, an individual that has several thousand errors early in its life is unlikely to be able to have fewer than 13,000 errors at the end of a 3 month retention test, having been subjected to 30,000 p/e cycles.

These sorts of individuals were identified by performing a single write, followed by calculating the number of errors. Any individual with more than 1,000 errors was immediately given the worst possible fitness and the test terminated. This reduced the test from 90 min to less than a minute.

Simulated Evolution used DataModeler (LLC 2013) to simulate the solution space with ensembles of GP individuals. The following section details how this was achieved; it was used to select which individuals to seed the GAs with.

Experiments

Pulling together the tools from the previous section, we have a multi-step process which can be summarized as below in Fig. 5.7. First, we identify which registers need to be targeted, as in Sect. 3. Next we trimmed the registers so that the GA would concentrate only on useful parts of the search space.

We next initialize the GA runs, and run 30 in parallel, occupying 16 different hardware testers. This ensures that individuals from different GA runs are tested

across a variety of chips. The variances within the chips is accounted for by always using relative improvements for fitness measure, and we have successfully mixed chips from different batches for these runs. A simple cache is used to ensure that the same individual isn't constantly evaluated as the populations begin to converge. Any individual being evaluated for the fourth or more time is not evaluated in hardware, and instead an average of its actual hardware derived fitness measures is given.

All data is collected, in the form of values for the various input parameters, e.g. where the read poles should go, etc. and a single output value. At this point, the data is passed to a run of DataModeler, from EvolvedAnalytics, a high powered, very efficient GP symbolic regression tool that produces ensembles of GP individuals. We use GP to map out the search space as explored by the GA individuals.

The ensemble produced by GP is then used as the fitness function by a traditional software GA that optimizes the simulated search space, producing a selection of individuals that are used to seed the next batch of GA runs. The fitness for these individuals is their level of errors, but, as these are all endurance tests, without any retention tests, it isn't as simple as selecting a *best-of-run* individual as one normally does. Instead, a selection of *candidates* are chosen, which are individuals that *may* be good enough.

Candidates

We created a taxonomy of individuals that were tested in hardware. These are

Type 0: Block under test failed to reach 30,000 cycles.

Type I: Block reached 30,000 cycles, but with enough errors to raise concerns about whether or not the individual would successfully pass a retention test.

Type II: Block reached 30,000 cycles with a relatively small number of errors.

Type II individuals were considered to be *candidates*, meaning that there was at least a possibility that they would pass a retention test. The final step in the process was to select candidates for the retention test. This is a crucial final step because of the expense of this test, as each test takes approximately 8 **days** and approximately 300 individuals can be tested (the constraint here is how many testers can physically fit in the ovens). However, to allow for variances within the chips and in anticipation of various hardware failures, which, in this particular project, have manifested themselves in ways varying from melted cables (from touching the elements) to poor contacts with the chips (from expansion and contraction of metal due high temperatures), a lot of redundancy was employed, reducing us to 100 candidates in the initial tests.

5 Results

Due to the various accelerated tests that were developed in the course of this work, over 20,000 individuals were tested in *hardware*. Given that even with our purpose built tester 30,000 p/e cycles takes about 1.5 h to run, this is a huge number, made

Table 5.1 Numbers of individuals discovered at various error counts. Measures are shown at 20,000 also, to indicate that the problem grows more difficult at higher cycles. Notice the dramatic increase in error counts from 20 to 30K. These are cycling errors only

Cycles	Baseline errors	# of individuals with <50 % errs	# of individuals with <50 % errs	# of individuals with <50 % errs
20,000	156	1,252	920	520
30,000	950	1,222	710	410

possible only by the fact that our pre-screening process removed many individuals after just a single p/e cycle.

Embodied Results

The average number of errors in baseline devices, i.e. with standard parameters, was 946. This was discovered by cycling the blocks continuously up to 30,000 p/e cycles, measuring for errors every 1,000 cycles. A special *challenge pattern* was developed for the error test, which ensured that locations had opposite values. This is a non-trivial task because there usually is not a direct mapping from logical units to the physical layer, i.e. the 2 bits stored in the same physical memory cell generally are not in the same logical unit, to reduce the impact of cell failure.

SSD manufacturers often use a pseudorandom pattern as this is more representative of the sort of data that is generally stored in devices, and our experience with the retention bake (see Sect. 5 below) suggests that this probably motivated by the fact that a challenging enough pattern will likely fail any reasonable retention test.

We present results here in terms of comparison with the baseline parameters, and show the number of individuals that were discovered below various thresholds. This is more meaningful than average error counts because each individual must be treated separately. Only unique individuals are reported (Table 5.1).

Of these individuals, 300 were selected as candidates for the retention bake.

GP Results

In total, there were five iterations of (20 parallel) GA runs. At the end of each run, DataModeler was used to create an ensemble of expressions using GP. The only data points used as inputs were those tested in hardware, and these included *failed* individuals, i.e. those that were deemed to have too many errors after just a single p/e cycle. The quality of solution of the GP runs, as measured by $1 - R^2$ of the best solution found increased dramatically across the five iterations.

Starting with a $1 - R^2$ score of 0.595, we experienced a constant improvement over the iterations, until eventually we reached a score of 0.198. Given how little data was available, and, in particular, how much of it related to failures (over 90 % in the first iteration), this was a very impressive result.

Simulated Results

After each iteration, we ran a GA in software using the best ensemble from GP as the fitness function. As there was no baseline to compare with, fitness was simply the number of errors returned for each individual. Rather surprisingly, each iteration, even the first, not only discovered individuals that were predicted to incur zero errors after 30,000 cycles, but also routinely discovered individuals that would incur **negative** (i.e. less than zero) errors!

Clearly, this is due to noise in the data. However, as the point of the exercise was to use GP to direct us to interesting areas of the search space to examine, these sorts of anomalous results are actually more useful than one might expect, as representative individuals with these sorts of results were always amongst those chosen to seed the next batch of GA runs.

We say representative because, in general, when an individual appeared that scored negative errors, the system would discover many others with a similar score, all of which were variants on that individual. Given that we suspected these to be anomalies in the data set rather than indications of fruitful areas of the search space, it made sense to use a relatively small number of them to "flesh out" that area of the input space for the subsequent GP runs.

At no time did any individual with negative results turn out to be a candidate for the retention bake.

Retention Bake

The task faced by this research was to produce a set of operating parameters that could successfully manage an order of magnitude increase in p/e cycle endurance while maintaining retention of 3 months. This is measured on a pass or fail basis. In practice, given the enormous variance in the devices under test, as well as the importance of verification and replicability, the only way to truly validate our results was for a third party to do so. This was conducted by a Flash manufacturer using a Teradayne tester (which typically retails in the region of $1,000,000).

We conducted several retention bakes, each with several blocks using default parameters to ensure a meaningful comparison. Although NDAs prevent us from discussing the exact figures, we can say that at no time did any block using default parameters pass a 3 month retention test, even on chips that performed relatively

well. Of the initial 200 candidates, 120 of them failed the retention test. The failures included those with both the highest **and** lowest error counts from the initial phase.

Although counter-intuitive, recall from Sect. 2 that during the retention period the distributions move from right to left. If they aren't placed far enough to the right with tight enough distributions, they are more likely to cross the read poles, resulting in errors.

The remaining 80 were then tested again, with the top 50 % being retested a third and fourth time. This gave us the opportunity to test them on several different chips and to create as much data for each individual as possible. Eventually, the ten with the best combination of low average errors and low standard deviation of errors were given to the Flash manufacturer for independent testing.

The Final Result

Of the ten given to the manufacturer, all ten passed their independent test. This was a fantastic result, as it meant that the test rig built for this work, which includes ovens, purpose built testers, hand built sockets and tens of thousands of lines of code, was capable of providing the same level of rigour as industry standard, million dollar equipment.

6 Conclusions

We have described a system that uses a combination of simulated and embodied evolution to evolve control parameters for Flash memory devices, resulting in an order of magnitude improvement in their endurance, without compromising their retention. Tests with various levels of precision and associated costs were employed a different times in the process, varying from GP derived fitness function models that could process thousands of individuals in a minute, to JEDEC specification retention tests that could only process several hundred in the space of a week, resulting in a highly focused search.

Our results have been validated by the Flash manufacturers, meaning that the system can now be used for other geometries of chips. The next steps will be to examine sub 20 nm devices, which are considerably more prone to issues such as read disturb, and to test if our automated system can evolve around these problems.

These issues will make the problem considerably more difficult, but, given the cost benefit of using smaller geometries, even more modest increases in endurance will have a dramatic benefit on the performance of SSDs.

References

Abraham M (2010) NAND flash trends for SSD/Enterprise. In: Proceedings of flash memory summit, Santa Clara. http://www.flashmemorysummit.com/English/Conference/Proceedings_Chrono.html [Online]

Baek KH (2014) Memory Apparatus and Method for Controlling Erase Operation of the Same. U.S. http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=51&u=%2Fnetahtml%2FPTO%2Fsearchbool.html&r=2536&f=G&l=50&co1=AND&d=PTXT&s1=20140204.PD.&OS=ISD/20140204&RS=ISD/20140204

Hellmold S (2010) The evolving NAND flash business model for SSD. In: Proceedings of flash memory summit, Santa Clara. http://www.flashmemorysummit.com/-English/Conference/ [Online]

LLC EA (2013) DataModeler 8. http://www.evolved-analytics [Online]

LLC ST (2011) The top 20 things to know about SSD. http://www.seagate.com/-docs/pdf/ssd_faq.pdf. Accessed 18 July 2011

LythOs (2011) SSD vs HDD. http://elitepcbuilding.com/ssd-vs-hdd [Online]

Niebel A (2010) Visionary market research. In: Proceedings of flash memory summit, Santa Clara http://www.flashmemorysummit.com/English/-Conference/Proceedings_Chrono.html [Online]

Singer M (2005) Small gadgets to spark flash memory surge. http://news.cnet.com/2100-1041_3-5965603.html [Online]. Accessed 18 July 2011

Sullivan J, Ryan C (2007) A destructive evolutionary process: a pilot implementation. In: GECCO, London, pp 2167–2173

Sullivan J, Ryan C (2011) A destructive evolutionary algorithm process. Soft Comput 15(1):95–102

Tressler G (2010) Enabling MLC flash SSD in enterprise storage. In: Proceedings of flash memory summit, Santa Clara. http://www.flashmemorysummit.com/-English/Conference/Proceedings_Chrono.html [Online]

Chapter 6
Applying Genetic Programming in Business Forecasting

Arthur K. Kordon

Abstract Since the global recession of 2008–2009, it has been much more widely understood that reliable economic forecasting is needed in business decision-making. Of special interest are the forecasting methods based on explanatory variables (economic drivers), the most popular of which is the Auto-Regressive Integrated Moving-Average with eXplanatory variables (ARIMAX) model. A limitation of this approach, however, is the assumption of linear relationships between the explanatory variables and the target variable. Genetic programming is a potential solution for representing nonlinearity and a hybrid scheme of integrating static and dynamic nonlinear transforms into the ARIMAX models is proposed in the chapter. From an implementation point of the view the proposed solution has several advantages, such as: optimal synergy between two well-known approaches like GP and ARIMAX, avoiding the need for developing a solid theoretical alternative for nonlinear time series modeling, using available forecasting software, and low efforts to train the final user. The proposed approach is illustrated with two examples from real business applications in the area of raw materials forecasting.

Keywords Nonlinear forecasting • Symbolic regression • Nonlinear transforms • Business applications

A.K. Kordon (✉)
The Dow Chemical Company, Freeport, TX, USA
e-mail: akordon@dow.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_6,

1 Introduction

The global recession of 2008–2009 caught industry by surprise and as a result, a trend of growing demand for using economic forecasts in business decision-making is observed (Rey et al. 2012). The increased demand for forecasting triggered the development of new methods in addition to the "classical" time series statistical approaches such as exponential smoothing and the Box-Jenkins ARIMA models. One fruitful direction of development is that of nonlinear time series modeling, based on various computational intelligence methods, such as neural networks, support vector machines and Genetic Programming (GP). Other developments, of special importance to industrial applications, are the efforts for improving the time series forecasts by selecting the best potential drivers using various variable selection methods. A short list of such methods includes but is not limited to the following: similarity analysis, sequential pattern matching, Principal Component Analysis (PCA), decision trees, co-integration analysis, variable cluster analysis, stepwise regression and GP.

Established application areas for nonlinear models are macroeconomics and finance. For example, most real business cycle models are highly nonlinear. The application list includes also bond pricing models, product diffusion processes, and almost all other continuous time finance models (Clements et al. 2004). The key idea behind the nonlinear nature of economic time series is that major economic phenomena, such as output growth in a market economy is represented by the presence of two or more regimes (e.g. recessions and expansions), as is the case of financial variables (with periods of high and low volatility). Other types of nonlinearity might include the possibility that the effects of shocks accumulate until a process reach a catastrophic event (as is the case of the financial crisis in 2008).

One of the methods that are appropriate for nonlinear forecasting is GP. Numerous studies have applied GP to time series forecasting with favorable results (for the current state of the art and many references see the survey paper Brabazon et al. 2008). The main advantage of applying GP to time series forecasting is that it automatically selects and self-adjusts its functional form and the time period on which its forecasts are based. GP has been used for US demand of natural gas forecast, daily exchange rate forecast, real estate prices forecast of residential single family homes in south California, oil revenue in Kuwait forecast, stock trading rules generation, etc. However, these experimental algorithms lack the solid theoretical basis and wide acceptance of the ARIMA models and this is a severe limitation for successful applications in a business setting.

In order to remove this limitation, a hybrid system that integrates GP and ARIMA is proposed in this chapter. The key idea is to apply GP for variable selection and nonlinear transforms generation, which can be used as explanatory variables in the ARIMAX models. The chapter is organized in the following manner. First, the key features and limitations of ARIMAX models are discussed in Sect. 2. The proposed hybrid approach is described in Sect. 3 and examples of real-world applications with hybrid systems based on static and dynamic GP-generated transforms used in ARIMAX models are given in Sect. 4.

2 Key Issues in Business Forecasting

The ultimate objective of business forecasting is to deliver to the key decision-makers a reliable forecast on specific economic variables, such as product demand, raw materials prices, labor cost, etc. Two technical factors contributed to the increased use of business forecasting after the recession in 2008–2009: (1) improved internal discipline and processes on time series data collection of important economic variables within the businesses and (2) the explosion of available business-related time series data. For example, the leading source of this kind of information, IHS Global Insight, contains more than 30 million time series across the globe (www.ihs.com). The key issue is how to use these sources of information for more reliable prediction having in mind the available forecasting methods and their limitations.

Key Forecasting Methods

Classical time series forecasting methods can be subdivided into the following categories (Kennedy 2008):

1. Exponential smoothing methods
2. Regression methods
3. ARIMA methods
4. Threshold methods
5. Generalized autoregressive conditionally heteroskedastic (GARCH) methods

The first three can be considered as linear methods, i.e., methods that employ a linear functional form for time series modeling, and the last two as non-linear methods. For example, in exponential smoothing a forecast is given as a weighted moving average of recent time series observations. In regression a forecast is given as a linear function of one or more explanatory variables. ARIMA methods, presented by Box and Jenkins in 1970s, give a forecast as a linear function of past observations (or the differences of past observations) and error values of the time series itself (Box and Jenkins 1976).

In the ARIMA (*p, d, q*) model, p is the order of auto-regression, d is the order of differencing, and q is the order of the moving average process. Generally speaking, the ARIMA model (aka univariate ARIMA forecasting model) can be represented as a linear combination of the past observations and past errors as follows:

$$\begin{aligned}&(1-\phi_1 B-\phi_2 B^2-\ldots-\phi_p B^p)(1-B)^d y_t\\ &=\delta+(1-\theta_1 B-\theta_2 B^2-\ldots-\theta_q B^q)\varepsilon_t,\quad t=2,3,\ldots,\end{aligned} \tag{6.1}$$

where y_t is the actual value, B is the backward shift operator, d is the constant item, ε_t is the random error at time t, ϕ_p and θ_q are the coefficients of the model and can be estimated by the least squares method. The model development includes three iterative steps of model identification, parameter estimation, and diagnostic checking. The basic idea of model identification is that if a time series is generated from an ARIMA process, it should have some theoretical autocorrelation properties. For this purpose, the autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the time series data are used to identify the order of the ARIMA model. In the identification step, data transformation is often required to make the time series stationary. Stationarity is a necessary condition in building an ARIMA model used for forecasting.

A stationary time series is characterized by statistical characteristics such as the mean and the autocorrelation structure being constant over time. When the observed time series presents trend and heteroscedasticity, differencing and power transformation are applied to the data to remove the trend and to stabilize the variance before an ARIMA model can be fitted by using a nonlinear optimization procedure. The last step in model building is the diagnostic checking of model adequacy. Several diagnostic statistics and plots of the residuals can be used to examine the goodness of fit. If the model is not adequate, a new tentative model should be identified, which will again be followed by the steps of parameter estimation and model verification. Diagnostic information may help suggest alternative model(s). This three-step model building process is typically repeated several times until a satisfactory model is finally selected. The final selected model can then be used for forecasting.

When the model includes eXplanatory variables with their past values, it's called ARIMAX. Often ARIMAX models are referred as dynamic regression (Pankratz 1991). The method has been widely used in financial, economic and social scientific fields (Kennedy 2008; Rey et al. 2012). The key component in the ARIMAX approach is the *transfer function model* (TFM).

Assume two time series denoted Y_t and X_t, which are both stationary. Then, the *transfer function model* (TFM) can be written as follows (Pankratz 1991):

$$Y_t = C + v(B)X_t + N_t \tag{6.2}$$

where: Y_t is the output series (dependent variable), X_t is the input series (independent or explanatory variable), C is constant term, N_t is the stochastic disturbance, i.e. the noise series of the system that is independent of the input series.

$v(B)X_t$ is the transfer function (or impulse response function), which allows X to influence Y via a distributed lag, B is backshift operator. Coefficients v_j are called impulse response weights, which could be positive or negative. This is one of the key features of ARIMAX models which allow representing the dynamic effect of the explanatory variable on the target variable across several time samples (distributed lags). Output series might not react immediately to a change in input series, thus some initial v weights may be equal to zero. The number of v weights equal to zero is called dead time (concentrated lag). The structure of the TFM is defined

by a similar iterative process of identification, estimation, and diagnostics as with the ARIMA model. In some software packages, such as SAS Forecast Studio, this process is highly automated (SAS Institute Inc. 2011).

Issues in Business Forecasting

All linear forecasting methods above assume a functional form which may not be appropriate for many real-world time series. Linear models cannot capture some features that commonly occur in actual data such as asymmetric cycles and occasional outlying observations. Regression methods often deal with non-linear time series by logarithmic or power transformation of the data, however this technique does not account for asymmetric cycles and outliers.

In principle, no univariate forecasting method is capable of accurately predicting the future when history does not repeat itself (Makridakis et al. 1998). As it is well-known, however, this is a very strong assumption. Due to social, technological, and structural economic changes, history never exactly repeats itself. Another limitation factor that has to be taken into account in forecasting is the level of randomness of social complex systems. For example, the efficient markets have a random walk-type behavior, which reduces their predictability. The generic limitations of time series forecasting and the constraints of the different forecasting horizons are discussed briefly below and in detail in Rey et al. (2012).

Generic Limitations of Time Series Forecasting

Univariate forecasting requires a representative historical data set that accurately captures the trend, seasonality, and cyclicality in the data – collecting such data sets is one of the biggest challenges for development of forecasting models. The most difficult requirement is representing the cyclicality, which is business-specific, and may require a long history of several years, containing at least several cycles. Very often internal business data sets are with shorter lengths for satisfying this condition and representing cyclicality.

Changes in fundamental conditions can cause the forecast to vary from actual results – the hypothesis for consistency of the identified patterns in the past into the future faces the reality check with every new actual piece of data. In the case of fundamental changes of the subject of forecasting, such as revolutionary new technology introduction or significant business transformation, time series forecasting cannot identify automatically the new pattern.

Univariate time series forecasting assumes no change in the environment – in a similar fashion, changes in the economic environment, such as introducing new taxes or environmental regulations cannot be captured automatically and model re-development is needed.

Another issue of univariate forecasting is its limited capability in anticipating rare events. Some forecasting software allows defining and adding events, which improves forecast accuracy after the event. Unfortunately, it is of limited value in forecasting of a future event before it occurs.

Other issues in business forecasting are related to the limitations of the forecasting time horizon.

Limitations of Forecasting Time Horizon

Another important factor that has to be taken into account in defining realistic forecasting expectations is the forecasting time horizon length. Obviously, the longer the time horizon, the higher the uncertainty in the forecast and the requirement for a long historical data set. Forecasting time horizon can be defined in three categories – short-term (up to three time samples in the future), medium-term (up to 18 time samples in the future), and long-term (more than 18 time samples in the future).[1] The limitations for the short, medium, and long-term time horizons are discussed briefly below.

- Limitations for short-term forecasting horizon – this is the best case scenario for forecasting since the key economic phenomena that contribute to the forecasting models (trend and seasonality) do not change much or frequently over a short time span (Makridakis et al. 1998). Most of the available forecasting methods can be used for short-term forecasting with different success based on their specific limitations. The key issue in short-term forecasting is the appearance of an unexpected event, such as a weather disaster or an unplanned shutdown.
- Limitations for medium-term forecasting horizon – the key limitation of this forecasting category is the influence of business cycles and main economic moves, such as recessions and booms. Only limited forecasting methods, such as ARIMA, are trustable in this case. Complementary business cycles estimation is strongly recommended. The recommended method is using multivariate ARIMAX-based forecasting methods.
- Limitations for long-term forecasting horizon – in most of the cases the available time series forecasting methods are unreliable over a long time span. In addition to time series forecasting models, it is recommended to identify and extrapolate megatrends, use analogies, and scenario-based forecasting.

[1]The forecasting time horizon classification is according to Makridakis et al. (1998).

3 How Can Genetic Programming Improve Business Forecasting?

Some of the discussed issues can be resolved by a proper use of GP. Two key modes of operation: (1) generating direct nonlinear forecasting models by GP and (2) using nonlinear transforms, generated by GP, in ARIMAX models are discussed below.

Direct Generation of Nonlinear Forecasting Models by GP

As explained previously, the task of time series prediction is accomplished by using previous values of a time series to predict future values. Therefore, the direct generation of nonlinear forecasting models approach requires that the GP terminal set will consist of previous values of the time series up to a certain maximal lag limit. To measure the predictive performance of a solution, the following method is used. First, a vector of past time series values for the last (most recent) test case is applied to the solution (function) and the predicted future value, $y(t+1)$, is calculated. Then this predicted value is appended to the past values to produce a new vector which is again applied to the solution to generate the next predicted future value, $y(t+2)$. This process is continued until the desired number of future values to predict is generated. Finally, the mean of the squared errors between the actual time series values and the predicted values is calculated (Kaboudan 2003). Recently, an interesting study of exploring the capabilities of GP in generating seasonal forecasting models by using time-shift operators based on expected seasonality shows encouraging results (Agapitos et al. 2011).

Another interesting approach for applying GP in time series forecasting is the Dynamic Forecasting Genetic Programming, called DyFor GP (Wagner et al. 2007). The key difference of this method relative to standard GP technique is that it is adding features that are tailored for the forecasting of time series whose underlying data-generating processes are non-static. Such time series often appear for real-world forecasting concerns in which environmental conditions are constantly changing. The distinctive feature of DyFor GP is its adaptive data window adjustment. This feedback-driven window adjustment is designed to automatically hone in on the currently active process in an environment where the generating process varies over time (see details in Wagner et al. 2007).

When applied to time series forecasting, GP has four advantages over classical linear forecasting methods: (1) the functional form of the forecasting model need not be prescribed, and instead, is automatically discovered, (2) the functional form is not restricted to that of a linear model, (3) ability to produce several high-quality solutions, and (4) it has a built-in nonlinear variable selection.

The key disadvantages of applying GP for direct generation of nonlinear forecasting models can be summarized as: (1) very slow model generation relative to other nonlinear approaches, (2) the derived nonlinear forecasting models have no classical statistical interpretation, and (3) the whole approach is new to forecasting community and lacks solid theoretical basis and professional software.

A Hybrid System Between GP and ARIMAX

Exploring the synergistic capability of different methods by their effective integration is one of the best ways to improve their performance and broaden applicability. Examples of successfully applied hybrid intelligent systems integrating neural networks, support vector machines, and GP are given in Kordon (2004). A broader range of integrating the proposed hybrid intelligent systems with first-principles and statistical models is illustrated with several industrial applications in Kordon (2010). At the basis of building hybrid systems is the principle of maximizing the synergy, i.e., enhancing the advantages and compensating the disadvantages of the individual methods. Applied to a hybrid system between GP and ARIMAX approaches, this principle implies using appropriate nonlinear models automatically generated by GP as explanatory variables in ARIMAX models.

The proposed hybrid system is different from the structure, suggested by Zhang (2003) of integrating ARIMA with neural networks and the integration of ARIMA and GP, explored recently by Lee and Tong (2011). In both cases, the defined hybrid system for nonlinear time series forecasting is univariate. First, a linear ARIMA model is built and then, a nonlinear model is generated on the residuals either by neural networks or GP. The final forecast is obtained by adding the forecasted values of the linear and nonlinear components.

The proposed hybrid system for nonlinear forecasting is based on different integration mechanism. The nonlinearity is represented with transforms, generated by GP and used as explanatory variables in ARIMAX models. The final forecast is generated by these ARIMAX models. The nonlinear equations, generated by GP, can be based on both contemporaneous and dynamic relationships. The last option assumes using lagged inputs up to an expected maximum lag. The selected nonlinear transforms from the GP-generation phase are used in ARIMAX model generation but they might not be selected in the final model if they are statistically insignificant. If distributed lags of a statistically significant nonlinear transform exist, it is represented by the corresponding transfer function in the ARIMAX model. As a result, GP complements ARIMAX models with adding contemporaneous and dynamic nonlinear explanatory variables and the final models are with all the benefits of this well-known forecasting approach, such as building multi-step forecasts of all inputs, statistically defined confidence limits, and available software.

The development process of a hybrid GP-ARIMAX forecasting system includes the following steps:

Step 1: Time Series Data Preparation In order to apply GP for variable selection and model generation in time series modeling framework, some data preparation of the available time series is needed. Since time series data are inherently correlated with time, it is necessary to reduce this correlation by making the data stationary and to remove the seasonal and cyclical effects. Also, it is crucial to include the lags for each input when selecting the variables because the relationships among the variables with the lags could be important. Unfortunately, this may increase significantly the number of variables. The selection of the number

of lags is a trade-off between fully representing the dynamics and the increased dimensionality of the search space for variable selection. The key methods for preparing the data for applying GP include: stationarity and seasonality analysis, exploring cross-correlation lags, differencing, de-trending, etc. (see details in Rey et al. 2012).

Step 2: Model generation by Pareto-Front GP Among many versions of GP, it is recommended that model generation is based on Pareto-front GP, which uses multi-objective optimization to direct the simulated evolution toward simple models with sufficient accuracy (Smits and Kotanchek 2004). In Pareto-front GP the simulated evolution is based on two criteria – prediction error (for example, based on 1-R^2) and complexity (for example, based on the number of nodes in the equations). In addition, it is possible to use other criteria, such as nonlinearity or age, which can improve the quality of the generated models.

Step 3: Variable and Nonlinear Transforms Selection One of the unique features of GP is its built-in mechanism to select the variables that are related to the target variable during the simulated evolution and to gradually ignore variables that are not. In this way, a different type of variable selection based on evolutionary computation principles can be used. The inputs ranking is based on the normalized frequency of their use in equations during the whole simulated evolution. An advantage of this type of variable selection is that it includes implicitly nonlinear relationships between the target and the inputs. Details for the principles of GP-based variable selection can be found in Smits et al. (2005). Often GP-based variable selection is the only option in short time series data with many lags because classical statistical methods have difficulties with variable selection from "fat" data.

Nonlinear transform selection is still an art than science and is usually located around a special area on the Pareto Front. An example is given in Fig. 6.1 where each point corresponds to a certain model with the x-coordinate referring model complexity and the y-coordinate – the model error. The points marked with circles form the Pareto front of the given set of models. The Pareto front itself is divided into three areas from point of view of efficient model selection (see Fig. 6.1). The first area, which is the focus of attention for proper transform selection, contains the simple models that occupy the lower left part of the Pareto front. The second area of complex over-fit models lie on the bottom right section of the Pareto front. The third interesting area of best-fit models is around the tipping point of the Pareto front where the biggest gain in model accuracy for the corresponding model complexity is. Very often, however, the fittest models are too complex for integration with the ARIMAX framework. For the proposed hybrid approach, model selection is reduced to a small number of candidate models from the first area of best transforms with low-complexity models. It is also recommended to follow the suggestions for GP transforms definition, given in Castillo et al. (2010).

Step 4: ARIMAX Models Generation The selected transforms are used as explanatory variables in ARIMAX models. The models are generated by the standard methodology, described in Sect. 2 and can introduce dynamics with

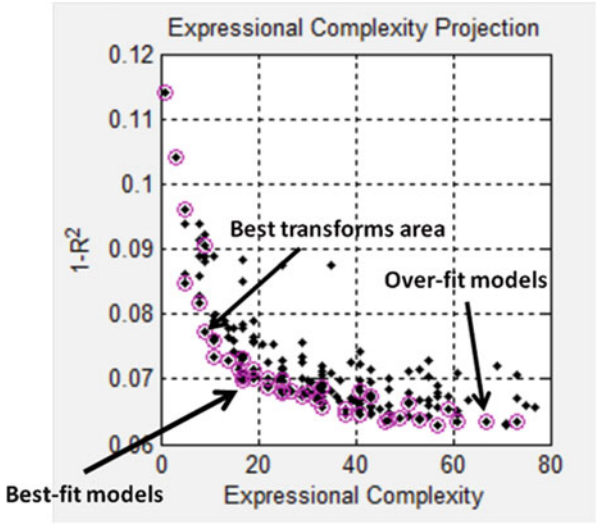


Fig. 6.1 Key Pareto-front areas of GP-generated models

appropriate transfer functions on some transforms. It is recommended to develop alternative ARIMAX models based on the different transforms as well as a linear benchmark model with the original explanatory variables. The final ARIMAX model selection is based on the forecasting accuracy on hold-out data. It has to be taken into account that a better fit of a transform on historical data used for GP model generation doesn't "guarantee" an ARIMAX model with more accurate forecasts on hold-out data. Building an ensemble of ARIMAX models is also an option.

The advantages of the proposed hybrid forecasting system based on integrating GP and ARIMAX approaches are as follows:

- Optimal synergy between two well-known approaches (GP and ARIMAX)
- Avoiding developing a solid theoretical basis for GP-based nonlinear forecasting models
- Using available software for ARIMAX model development
- No additional training of final users.

The limitations of the proposed hybrid structure are:

- Additional data preparation of time series needed
- Approach not recommended for nonstationary time series
- Model developers need knowledge on both GP and ARIMAX

4 Examples

The proposed hybrid method will be illustrated with two examples from real business applications in the important area of raw materials forecasting (Kordon 2012). In all examples, an internal Pareto-Front GP package has been used for

nonlinear transforms generation. For each example, 20 different GP processes with fitness maximal correlation have been run. Each GP process is initialized by a random set of equations, continues for 625 generations, and includes a population of 200 equations. SAS Forecast Studio has been used for ARIMAX model development with different settings for corresponding models.

Forecasting with Contemporaneous Nonlinear Relationships

The first example is based on a raw material price history of monthly data from January 2000 to April 2011. The data from January 2000 to April 2009 are used for model development (learning data), the next 12 months are used for model selection (hold-out or test data), and the last 12 months are used for *ex ante* forecasting. It is generally agreed in the forecasting community that evaluating forecasting performance must be done exclusively on forecasting observations that do not appear in the data set used to estimate the forecasting equation (Kennedy 2008). The experts have defined eight potential economic drivers that may influence this price. The analysis of the cross-correlation functions show that there are no identified concentrated lags between the price and the potential drivers, i.e., the expected relationships are contemporaneous. As a result, the data set for nonlinear transforms generation does not include lags.

The results from the GP model generation phase are as follows:

Variable selection: The variable ranking, based on the normalized frequency of participation of the corresponding variable in GP-generated equations, is shown in Fig. 6.2. On the top of the ranking is x_1, which has been selected in 48 % of GP-generated equations, followed by x_2 with 30 % and x_6 with 14 %.

Nonlinear transforms selection: Two nonlinear transforms – GP_Model50 and GP_Model78 have been selected from the best transforms area of the Pareto front. The first simple transform is given in Eq. 6.3 and its performance on learning and test data is shown in Fig. 6.3.

$$Y\_GP\_Model50 = -573 + 4009 \times \frac{\sqrt{x_2}}{x_6} \tag{6.3}$$

The second transform, given in Eq. 6.4, is more complex with slightly higher performance on learning data (R^2 of 0.96) and test data (R^2 of 0.82).

$$Y\_GP\_Model78 = 307.4 + 65.7 \times \left(\frac{x_2}{x_6} - ln(x_2)\right) \tag{6.4}$$

The results from the ARIMAX model generation phase are as follows:

This phase includes developing and comparing three ARIMAX models: a benchmark linear model based on statistically significant explanatory variables, nonlinear model based on GP_Model50 transform, and nonlinear model based on

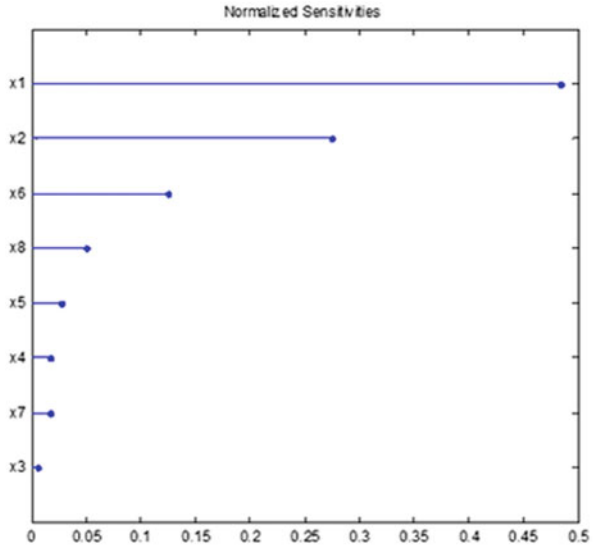


Fig. 6.2 Variable ranking based on normalized frequency of participation in GP-generated functions

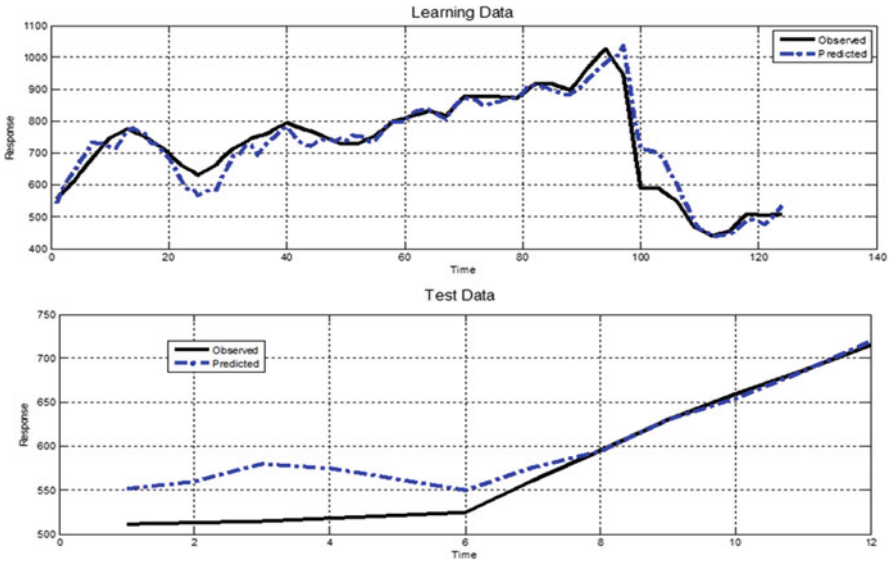


Fig. 6.3 GP_Model50 performance on learning data (R^2 of 0.93) and test data (R^2 of 0.79)

GP_Model78 transform. The generated ARIMAX models are with a forecasting horizon of 12 months, confidence level of 95 %, and the models are evaluated on hold-out data set of 12 months. The performance metric is the Mean Absolute Percentage Error (MAPE).

The parameters of the benchmark linear model are shown in Fig. 6.4. The ARIMAX model includes: a third order moving average term, a second order auto-

Component	Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|
y	MA1_1	-1.70811	0.15982	-10.69	<.0001
y	MA1_2	-1.67887	0.15149	-11.08	<.0001
y	MA1_3	-0.77069	0.14722	-5.23	<.0001
y	AR1_1	0.35390	0.20065	1.76	0.0805
y	AR1_2	0.58892	0.19803	2.97	0.0036
x1	SCALE	0.58378	0.02089	27.95	<.0001
x1	DEN1_1	0.20062	0.03107	6.46	<.0001
x2	SCALE	0.07945	0.05337	1.49	0.1394
x2	DEN1_1	-0.22394	0.09565	-2.34	0.0210
x2	DEN1_2	-0.86983	0.12230	-7.11	<.0001

Fig. 6.4 Parameters of linear ARIMAX model

Component	Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|
y	AR1_1	1.60159	0.07380	21.70	<.0001
y	AR1_2	-0.67217	0.07340	-9.16	<.0001
Model50	SCALE	0.61964	0.04464	13.88	<.0001
Model50	NUM1_1	-0.40347	0.11010	-3.66	0.0004
Model50	DEN1_1	-0.39572	0.13530	-2.92	0.0041
Model50	DEN1_2	0.37047	0.05080	7.29	<.0001

Fig. 6.5 Parameters of nonlinear ARIMAX model based on GP_Model50

Component	Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|
y	MA1_6	0.23271	0.09872	2.36	0.0201
y	AR1_1	1.61769	0.07829	20.66	<.0001
y	AR1_2	-0.65226	0.07585	-8.60	<.0001
Model78	SCALE	0.61606	0.04195	14.68	<.0001
Model78	NUM1_1	0.44113	0.09113	4.84	<.0001
Model78	DEN1_1	0.82030	0.09824	8.35	<.0001

Fig. 6.6 Parameters of nonlinear ARIMAX model based on GP_Model78

regressive term, and two statistically significant explanatory variables $-x_1$ and x_2 with corresponding transfer functions. The hold-out MAPE is 3.06.

The parameters of the nonlinear model, based on GP_Model50 transform are shown in Fig. 6.5. The ARIMAX model includes: a second order auto-regressive term and the transform with a transfer function with a second order denominator, i.e. a dynamic component with a two samples distributed lag has been introduced. The hold-out MAPE of ARIMAX based on GP_Model50 is 2.18.

The parameters of the nonlinear model, based on GP_Model78 transform are shown in Fig. 6.6. The ARIMAX model includes: a first order moving average term, a second order auto-regressive term and the transform with a transfer function with a first order nominator/denominator, i.e. a dynamic component with one sample distributed lag has been introduced. The hold-out MAPE of ARIMAX based on GP_Model78 is 1.33 (the best from all three).

Table 6.1 *Ex ante* forecasting performance of contemporaneous ARIMAX models

Date	Actual	Linear model	Model50	Model78	APE_Linear	APE_50	APE_78
05/01/2010	$511.65	$523.06	$515.99	$521.84	2.23	0.85	1.99
06/01/2010	$513.35	$540.70	$522.99	$529.49	5.33	1.88	3.14
07/01/2010	$515.00	$562.14	$538.26	$539.40	9.15	4.52	4.74
08/01/2010	$518.37	$564.04	$542.00	$536.57	8.81	4.56	3.51
09/01/2010	$521.74	$559.36	$539.65	$530.89	7.21	3.43	1.75
10/01/2010	$525.00	$555.78	$533.01	$522.35	5.86	1.53	0.50
11/01/2010	$560.38	$580.80	$550.05	$534.97	3.64	1.84	4.53
12/01/2010	$594.62	$609.90	$567.28	$545.63	2.57	4.60	8.24
01/01/2011	$630.00	$639.98	$600.35	$563.58	1.58	4.71	10.54
02/01/2011	$659.28	$679.17	$626.36	$581.34	3.02	4.99	11.82
03/01/2011	$685.72	$717.34	$659.42	$600.11	4.61	3.84	12.48
04/01/2011	$715.00	$756.92	$691.55	$623.52	5.86	3.28	12.79
				MAPE	4.99	3.33	6.34

The *ex ante* performance of the three ARIMAX models for 12 months is shown in Table 6.1. The performance is estimated by the absolute percentage error (APE) at each time sample and the MAPE for the 12 months. The ARIMAX model, based on the simple nonlinear transform GP_Model50 demonstrated the best *ex ante* performance.

Forecasting with Dynamic Nonlinear Relationships

The second example is based on a raw material price history of monthly data from July 2007 to June 2012. The data from July 2007 to June 2011 are used for model development (learning data), the next 6 months are used for model selection (hold-out or test data), and the last 6 months are used for *ex ante* forecasting. The experts have defined five potential economic drivers that may influence this price. The analysis of the cross-correlation functions shows that two of them (x_2 and x_3) have identified concentrated lags between the price and the potential drivers, i.e., the expected relationships are dynamic. The corresponding cross-correlation plots between the price and these two drivers are shown in Fig. 6.7. As a result, the data set for nonlinear transforms generation includes data with two lags.

Two nonlinear transforms – a dynamic model GP_Model125 and a contemporaneous model GP_Model29 have been selected from the best transforms area of the Pareto front. The dynamic transform, given in Eq. 6.5 includes the lagged drivers x_2 and x_3 and the contemporaneous driver x_4. Its performance on learning and model generation phase is as follows: (R^2 of 0.74) and test data (R^2 of 0.52). The contemporaneous transform, given in Eq. 6.6 includes the drivers x_3 and x_4. Its performance on learning and model generation phase is as follows: (R^2 of 0.64) and test data (R^2 of 0.31).

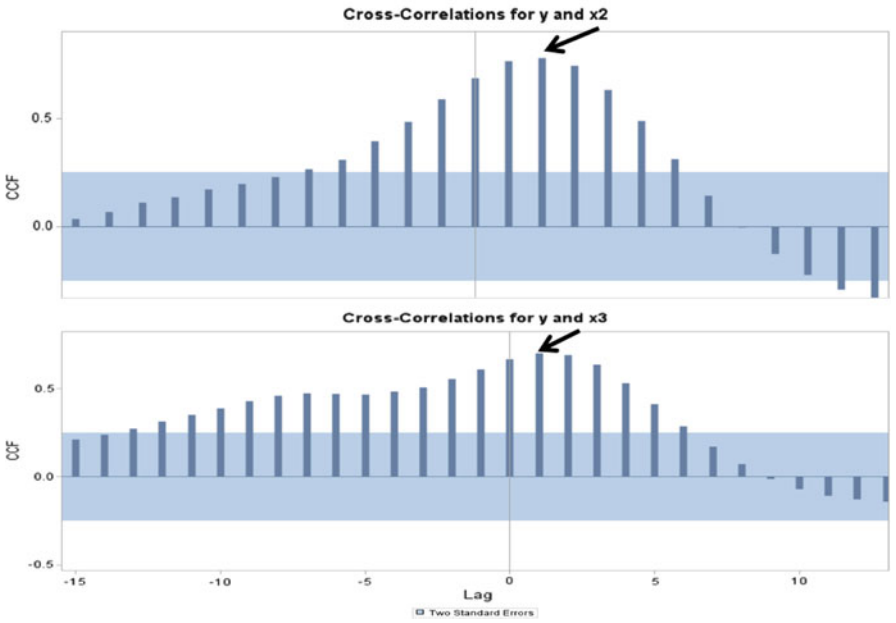


Fig. 6.7 Cross-correlation plots of x_2 and x_3. The estimated concentrated lag for x_2 is two and for x_3 is one

$$Y\_GP\_Model25(t) = -0.35 + 0.086 * [x_4(t) + x_2(t-2) * x_3(t-1)] \quad (6.5)$$

$$Y\_GP\_Model29(t) = 0.4 + 0.00013 * [x_4(t) * x_3(t)] \quad (6.6)$$

The ARIMAX phase includes developing and comparing three ARIMAX models: a benchmark linear model based on statistically significant explanatory variables, the dynamic nonlinear model based on GP_Model125 transform, and the contemporaneous nonlinear model based on GP_Model29 transform. The generated ARIMAX models are with a forecasting horizon of 6 months, confidence level of 95 %, and the models are evaluated on hold-out data set of 6 months. The ARIMAX model based on the dynamic transform has the best MAPE of 9.97 vs. 12.94 for the linear models and 12.56 for the contemporaneous transform model. The parameters of this model are given in Fig. 6.8. The ARIMAX model includes: a first order moving average term, a first order auto-regressive term, and the dynamic transform with a transfer function with a first order numerator, i.e. the dynamic relationship is refined with a one sample of distributed lag.

The *ex ante* performance of the three ARIMAX models and an ensemble for 6 months is shown in Table 6.2. The ARIMAX model, based on the dynamic nonlinear transform GP_Model125 demonstrated the best *ex ante* performance.

Component	Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|
y	MA1_2	-0.45082	0.12559	-3.59	0.0007
y	AR1_1	0.85828	0.07370	11.65	<.0001
GP_Model125	SCALE	0.52747	0.07238	7.29	<.0001
GP_Model125	NUM1_2	-0.48489	0.06756	-7.18	<.0001

Fig. 6.8 Parameters of nonlinear ARIMAX model based on dynamic transform GP_Model125

Table 6.2 *Ex ante* forecasting performance of dynamic ARIMAX models

Date	Actual	Linear model	Model 29	Model 125	Ensemble	APE linear	APE 29	APE 125	APE ensemble
01/01/2012	$1.03	$1.05	$1.03	$1.04	$1.04	2.01	0.06	1.07	1.01
02/01/2012	$1.03	$1.03	$1.06	$1.04	$1.05	0.46	2.59	1.37	1.47
03/01/2012	$1.14	$1.02	$1.10	$1.09	$1.07	10.13	3.14	4.02	5.76
04/01/2012	$1.17	$1.06	$1.16	$1.14	$1.12	9.10	1.23	2.94	4.42
05/01/2012	$1.16	$1.09	$1.19	$1.17	$1.15	5.93	2.98	1.27	0.56
06/01/2012	$1.09	$1.09	$1.21	$1.18	$1.16	0.46	11.11	8.17	6.27
					MAPE	4.68	3.52	3.14	3.25

5 Summary

The growing demand for accurate business forecasts requires more sophisticated methods, especially using nonlinear relationships among important economic drivers. The proposed hybrid system based on nonlinear transforms, generated by GP, used as explanatory variables in the popular ARIMAX models maximizes the synergy of both approaches. The hybrid system has been applied in two typical cases in business forecasting: (1) when the relationships between the forecasted variable and the related economic drivers are contemporaneous and (2) when the relationships are dynamic due to lags. In the first case, a simple nonlinear transform, generated by GP and used as exogenous input in the ARIMAX model, has shown the best *ex-ante* performance. In the second case, GP has generated a dynamic model with accurate lags. This model has been compared to another contemporaneous nonlinear model, generated by GP, and the best linear ARIMAX model. The *ex-ante* performance of the dynamic model is the best for the tested period of time. These encouraging results based on real world applications demonstrate the big potential of the proposed approach.

References

Agapitos A, O'Neill M, Brabazon A (2011) Evolving seasonal forecasting models with genetic programming for pricing weather-derivatives. In: Di Chio C, Agapitos A, Cagnoni S, Cotta C, Fernandez de Vega F, Di Caro GA, Drechsler R, Ekart A, Esparcia-Alcazar AI, Farooq M, Langdon WB, Merelo JJ, Preuss M, Richter H, Silva S, Simoes A, Squillero G, Tarantino

E, Tettamanzi AGB, Togelius J, Urquhart N, Uyar AS, Yannakakis GN (eds) Applications of evolutionary computing, EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, EvoSTOC, Malaga. LNCS, vol 7248. Springer, pp 135–144. doi:10.1007/978-3-642-29178-4-14

Box GEP, Jenkins GM (1976) Time series analysis: forecasting and control. Holden Day, San Francisco

Brabazon A, O'Neill M, Dempsey I (2008) An introduction to evolutionary computation in finance. IEEE Comput Intell Mag 3(4):42–55. doi:10.1109/MCI.2008.929841, http://ieeexplore.ieee.org/xpl/-tocresult.jsp?isYear=2008isnumber=4625777Submit32=Go+To+Issue

Castillo F, Kordon A, Villa C (2010) Genetic programming transforms in linear regression situations. In: Riolo R, McConaghy T, Vladislavleva E (eds) Genetic programming theory and practice VIII. Genetic and evolutionary computation, vol 8. Springer, Ann Arbor, chap 11, pp 175–194. http://www.springer.com/computer/ai/book/978-1-4419-7746-5

Clements M, Franses P, Swanson N (2004) Forecasting economic and financial time-series with non-linear models. Int J Forecast 20:169–183

Kaboudan MA (2003) Forecasting with computer-evolved model specifications: a genetic programming application. Comput Oper Res 30(11):1661–1681. doi:10.1016/S0305-0548(02)00098-9, http://www.sciencedirect.com/science/article/B6VC5-47P1N3H-1/2/d89d466d6ed20bb2d2da43b3701f351b

Kennedy P (2008) A guide to econometrics. Wiley/Blackwell, Hoboken/Malden

Kordon A (2004) Hybrid intelligent systems for industrial data analysis. Int J Intell Syst 19:367–383

Kordon A (2010) Applying computational intelligence how to create value. Springer, Heidelberg/New York. http://www.springer.com/engineering/book/978-3-540-69910-1

Kordon A (2012) Applying data mining in raw materials forecasting. In: SAS analytics 2012 conference, Las Vegas

Lee YS, Tong LI (2011) Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming. Knowl Based Syst 24(1):66–72. doi:10.1016/j.knosys.2010.07.006, http://www.sciencedirect.com/science/article/B6V0P-50JHBSY-1/2/1501a7c1121cbfcf9683f1a0d781806b

Makridakis S, Wheelwright S, Hyndman R (1998) Forecasting: methods and applications. Wiley, New York

Pankratz A (1991) Forecasting with dynamic regression models. Wiley, New York

Rey T, Kordon A, Wells C (2012) Applied data mining for forecasting using SAS. SAS, Cary

SAS Institute Inc. (2011) SAS forecast studio 4.1: user's guide. SAS Institute Inc., SAS, Cary

Smits G, Kotanchek M (2004) Pareto front exploitation in symbolic regression. In: Genetic programming theory and practice II. Springer, Boston, pp 283–300

Smits G, Kordon A, Vladislavleva K, Jordaan E, Kotanchek M (2005) Variable selection in industrial datasets using pareto genetic programming. In: U.-M. O'Reilly, T. Yu, R. Riolo and B. Worzel (eds) Genetic programming theory and practice III. Genetic programming, vol 9. Springer, Ann Arbor, chap 6, pp 79–92

Wagner N, Michalewicz Z, Khouja M, McGregor RR (2007) Time series forecasting for dynamic environments: the DyFor genetic program model. IEEE Trans Evol Comput 11(4):433–452. doi:10.1109/TEVC.2006.882430

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, 50: 159–175

Chapter 7
Explaining Unemployment Rates with Symbolic Regression

Philip Truscott and Michael F. Korns

Abstract Much of the research on the accuracy of symbolic regression (SR) has focused on artificially constructed search problems where there is zero noise in the data. Such problems admit of exact solutions but cannot tell us how accurate the search process is in a noisy real world domain. To explore this question symbolic regression is applied here to an area of research which has been well-travelled by regression modelers: the prediction of unemployment rates. A respected dataset was selected, the CEP-OECD Labor Market Institutions Database, to provide a testing environment for a variety of searches. Metrics of success for this paper went beyond the normal yardsticks of statistical significance to demand "plausibility". Here it is assumed that a plausible model must be able to predict unemployment rates out of the sample period for six future years: this metric is referred to as the "out of sample R^2". We conclude that the two packages tested, Eureqa and ARC, can produce models that go beyond the power of traditional stepwise regression. ARC, in particular, is able to replicate the format of published economic research because ARC contains a high level Regression Query Language (RQL). This research produced a number of models that are consistent with published economic research, have in sample R^2 values over 0.80, no negative unemployment rates, and out of sample R^2 values above 0.45. It is argued that SR offers significant new advantages to social science researchers.

Keywords Abstract expression grammars • Genetic algorithms • Symbolic regression • Non-linear regression

P. Truscott (✉)
Singapore University of Technology and Design, 20 Dover Drive, 138682, Singapore
e-mail: philiptruscott@sutd.edu.sg

M.F. Korns
AIS Foundation, 98 Perea Street, Makati 1229, Philippines
e-mail: mkorns@korns.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_7,

1 Introduction

> . . .when statistics are not based upon computations which are strictly accurate, they mislead instead of guiding aright. The mind is easily imposed upon by the false affectation of exactness, which prevails even in the misstatements of science, and it adopts with confidence errors which are dressed in the forms of mathematical truth.
>
> *Democracy in America* de Tocqueville et al. (1952)

Symbolic regression researchers are engaged in the quest for an infallible search agent but can they really guarantee to find the correct model in a search space with trillions of potential candidates? Much of the recent research in this area has used artificially generated test problems with "zero noise." When the search finds the correct formula, the dependent variable can be predicted exactly (or at least within an infinitesimally small range of error). In the real world the noise is not only significant. At times it can be deafening.

This paper asks some fundamental questions about symbolic regression. Has the technique reached a point where it can make a real contribution to social science research? Can symbolic regression search languages focus the hunt to achieve better results than brute force universal searches? How far can the goal search be constrained to stay within the academic culture of specific disciplines? Can the plausibility of models be tested to ensure that they can make predictions in the real world, or are they misleading abstractions that, as de Tocqueville put it, are "merely dressed in the forms of mathematical truth?"

These issues will be explored by applying symbolic regression to a specific real world problem: the explanation of national unemployment rates. The discussion starts with an examination of one particularly famous unemployment formula and proceeds to show how far it can be extended or improved by evolutionary methods.

2 Predicting Unemployment with Institutional Variables

Possibly the most important post-war book on unemployment appeared in 1991. It was a landmark study on the relationship between jobless rates and labor market institutions. *Unemployment: macroeconomic performance and the labor market* by Layard et al. (1991) included a fascinating regression model that predicted unemployment rates for 20 OECD countries from 1983 to 1988 (see Table 7.1). For the sake of brevity we will call this the LNJ Model (derived from the names of the authors). The notes in the last column of Table 7.1 attempt to give a plain language explanation of the LNJ formula. It contains only one predictor variable from traditional macroeconomics: the change in a country's inflation rate was used to capture the effect of consumer demand. For example, sharply falling prices might indicate deflation, falling demand and rising jobless rates. It also included some effects of a country's system of wage bargaining. High rates of labor union membership tend to increase unemployment. However this can be mitigated by high

Table 7.1 Original Layard, Nickell and Jackman regression formula

Explanatory variable	Coeff.	Sig	Notes
Constant	0.24		
Change in inflation	−0.35	**	The change in inflation tries to capture changes in demand. A falling inflation rate implies falling demand and thus higher unemployment rates
Benefit duration	0.92	**	This measures the number of years a person can receive unemployment benefits. Countries with shorter time limits appeared to have lower unemployment rates
Replacement ratio	0.17	**	This compared the average incomes of those on unemployment benefit with the average income of the poorest 25% of workers. Higher ratios were said to make it less attractive to be in work so leading to higher jobless rates
Active labor market spending	−0.13	**	This included the spending, per unemployed person, on training and job matching services. The higher the active spending the lower the jobless rate
Union coverage	2.45	**	Union coverage measures the proportion of the national labour force covered by collective bargaining agreements. This posits that more powerful trade union movements may exert upward pressure on wages, increase the cost of labour and so increase unemployment
Union coordination	−1.42	*	While union coverage appeared to increase unemployment this effect could be mitigated by careful coordination of the wage bargaining process: where all unions negotiate at the same time inter-union competition is eliminated. Countries were categorized on a three point scale. Those with a wage bargaining covering the whole work force (like Sweden) scored a 3. Those with industry-wide coordination (like Germany) scored a 2. Countries without coordination (e.g. the UK and USA) scored a 1
Employer coordination	−4.28	**	This is the employer version of the Union Coordination described above. The same points system was used (3 = national coordination, 2 = industry coordination, 1 = no coordination)
R^2	0.91		In sample R^2
Sample size	20		

*Sig < 0.1; **Sig < 0.05

levels of wage bargaining coordination. Countries that negotiate across the whole labor force or a whole industry tend to have fewer attempts to leapfrog over other workers and so there is less need to control inflation by cutting consumer demand.

The formula also captures the amount spent on *active* labor market policies such as training and labor exchanges.

The most politically controversial variables capture the effects of cash benefits paid to the unemployed. According to the LNJ formula the longer the duration of unemployment benefits the higher the jobless rate. The *benefit replacement ratio* attempts to measure the generosity of the payments. The LNJ formula compares the average cash value of unemployment benefits with the average income of the poorest 25% of wage earners. A higher replacement ratio implies more generous benefits and a higher unemployment rate.

Table 7.1 is the only regression model in this chapter drawn from previously published research. It is an ordinary least squares model presented in Layard et al. (1991). The regression models in the next section have been derived from Eureqa. The regression models shown in Tables 7.4–7.7 below take the same general form as this model but they are the result of evolutionary searches using ARC. For more information on the internal algorithmic behaviour of ARC's search process please refer to Chap. 1, *Extreme Accuracy in Symbolic Regression*, in this volume.

3 The Quest for an Operational Model

The LNJ formula achieved an impressive 0.91 R^2 but with a small sample size of only 20 countries. Rather than data for individual years, 5 year averages for each country were used. The formula looks tantalizingly close to an *operational* model to predict unemployment rates in future years. However, constructing a generally respected model of this type has proved elusive. If regressions are done over a long time period with individual years, it is very easy to produce a model with a high R^2 but where some of the predicted unemployment rates are below zero. Negative unemployment rates inevitably cast serious doubt upon the validity of the predictive model.

One common strategy economists use to ensure that the model fits the data realistically is to use a Boolean flag variable for each country in the dataset. For example, the records in the dataset describing the country, France, would have a *FrenchFlag* variable coded to 1 while all the other countries would be coded to zero. These Boolean flag variables mean adding one extra variable to the list of predictors for each country in the analysis. Those who use this technique have given it an unprepossessing title: *country dummies*. One of the authors of the LNJ formula, Stephen Nickell, helped to devise a regression model (Nickell et al. 2003) using both country dummies and time dummies (separate Boolean flags for each year in the series). None of the predicted unemployment rates were negative but policy-makers should ask an important question about such research: If so much of the goodness of fit is coming from the dummies are the policy-related variables being reduced

to triviality? Ideally such research should be able to show how much jobless rates will fall in relation to a change in the benefit system, wage bargaining or economic policy, but the influence of the policy-related variables declines as the impact of the dummies rises.

In a traditional OLS regression it would be a simple matter to quantify the impact of the dummies. One could compare the R^2 figure for models with and without them. However Nickell et al. (2001) use generalized least squares (GLS) to allow for heteroscedasticity in the data. While most labor market economists would probably agree with their choice of a statistical procedure that takes this into account, the choice of GLS means their readers are deprived of a readily understandable goodness-of-fit measure like the R^2. Blanchard and Wolfers (1999) use similar country data and use non-linear least squares which does produce an R^2. While acknowledging that not all economists support least squares regression for this type of analysis, we will use it here to search for an operational unemployment prediction formula that satisfies these goals:

1. It should achieve a high goodness of fit (R^2)
2. It should use no dummy variables
3. It should not produce any negative unemployment rates
4. It should appear plausible in relation to out of sample predictions

4 The Test Data: The CEP OECD Data Set

A fascinating dataset on labor market institutions was published by Nickell (2006) covering most OECD countries for the years 1960–2004: the CEP-OECD Labor Market Institutions Database. This lacked an inflation change variable (required by the LNJ formula) and so was combined with information on inflation and real interest rates from the World Bank's[1] Databank.

The full dataset contains 617 rows, but some strategic predictor variables were missing for many years (mainly for the earlier years in the series). To strike a reasonable balance between predictive power and time coverage some records had to be deleted from the historical dataset. After some experimentation with fitness measures, it was decided to remove from the analysis records where there was missing data for inflation, active labor market spending or labor market centralization (entitled CEW in the original dataset). This resulted in a database of 282 rows. This data was further subdivided into two separate databases for testing and training. A training database covering the years up to and including 1993 contained 143 records. The testing database covering the years 1994–2000 contained 137 records. In order to assess how well the formulas predicted unemployment *after* 1993 the regression formulas were applied (with the same coefficients) to the post

[1] World Bank (2012).

1993 data. The mean difference between the predicted values and the actual values was calculated in the same manner as the traditional R^2. We will refer to this statistic below as the *Out-of-Sample* R^2.

All processing was done on a 64-bit Intel Core i5-2520M CPU at 2.5 GHz. The system had 4 GB of installed RAM. All searches in this paper were limited to 30 min.

In order to explore the researcher's options with symbolic regression grammars two very different packages were used. Eureqa (Schmidt and Lipson 2009) is a freely downloadable[2] product with a highly developed user interface. ARC (for Abstract Regression Classification) (Korns 2007, 2010, 2011) is still in the private domain and requires the user to implement commands with its built-in language RQL (for Regression Query Language) and also allows the package to be modified with LISP programming.

5 Testing with Eureqa

Eureqa has probably gone much further than any other software package in making symbolic regression readily accessible. It can be downloaded as a freeware version and has a low price option for academic researchers. It has an extremely user-friendly interface that made importing the CEP-OECD database a two-minute project.

The interface requires the user to set a *target* which is Eureqa's term for the symbolic regression search goal. Figure 7.1 below shows a goal similar to the initial search goal Eureqa suggested after loading the CEP-OECD database. The variable names have been conveniently taken from the column headers of the imported data. With minor editing *UNEM* (the unemployment rate) was specified as the dependent variable. Some other variables had to be removed from the pool of predictor variables such as *strunem* (the structural unemployment rate) because they were too close in concept to unemployment itself. In general, where there were groups of variables with similar characteristics they were all left in the pool

UNEM = f(Year, Inflation, Inf-Change, RIR, ep, epl, epl-Allard, udnet, udnet-vis, uc-Ochel, uc-oecd, uc, co, co-oecd, co-oecd-int, cow, cow-int, ce-oecd, ce-oecd-int, cew, cew-int, brr1, brr23, brr45, bd, brrl, brr-oecd, nrw, minw-med, ed90-50, ed50-10, educ, educ-int, ho, ho-oecd, ho-comb, almp-unem, regref, pmr, AdminR, EconR, T1, T2, T3, TW, TW-Nicol, sing1a, sing1b, sing2a, sing2b, sing3a, sing3b, sing4a, sing4b, mp1a, mp1b, mp2a, mp2b, mp3a, mp3b, mp4a, mp4b, ms1a, ms1b, ms2a, ms2b, ms3a, ms3b, ms4a, ms4b)

Fig. 7.1 Example Eureqa goal formula – universal in form

[2] Scmidt (2013).

Table 7.2 Two representative Eureqa champions

Complexity	In sample R^2	Numeric constants	Formula
18	0.85	2	UNEM = (home-owner-rate-comb + ms3a + 4.636*benefit-replacement-rate-oecd − 53.73 − benefit-replacement-rate-years − 2-and-3)/(bargaining-coordination + education-years-mean + almp-unem)
6	0.61	1	UNEM = 4.407 + benefit-replacement-rate-oecd/active-labour-div-unemployed

so that Eureqa could choose those that produced the best fit. However, it proved impossible to include both of the active labor market variables. The variables *almp* (active labor market policy spending as a proportion of GDP) and *almp-unem*(the active labor spending divided by the unemployment rate) contained too much information. Eureqa was able to predict the unemployment rate exactly by discovering the relationship between these two variables alone. For this reason, *almp* was dropped from the search.

Running the Eureqa's universal goal specified in Fig. 7.1 produced 12 champion formulas arranged in order of complexity. The calculation to define the complexity of a given regression model is fully configurable by the user. For example, the default settings allocate a complexity value of one to a constant or the addition operator, but a value of 4 to the factorial function (but any of these values can be altered). Two of these 12 champion formulas are shown in Table 7.2.

In the context of labour market economics, the output from Eureqa's universal goal (illustrated by Table 7.2) would be difficult to sell. The economists' culture requires regression models of a specific form. Generally, the grammar depth is only one with the independent variables modified by unary operators. Another problem is the explicability of the formula. In the simplest formula, for example, a benefit replacement rate is divided by active labour market policy spending per unemployed person. Culturally economists would find it unacceptable to have two such unrelated things on either side of the operator. Outside the domain of economics (for example in engineering) Eureqa may have the most sophisticated target specification language available. It offers a very flexible and intuitive way of solving differential equations, for example.

In order to try to produce a model more palatable to economists, a search goal was specified to replicate the form of the regression model in Table 7.1. This target is shown in Fig. 7.2 below (in the interests of readability the short form of the variable names has been used). The target expression groups variables into pools any one of which might be inserted into the model. Readers will notice that each group of variables is preceded by a function call with no arguments in the brackets. For example, the final line contains "f14() + f15(Year, ho, ho-oecd, ho-comb, urban)". This target format has been followed to avoid producing a champion formula with

```
UNEM= f0() *f1(bd)
+ f2() *f3(brr1, brr23, brr45, brrl, brr-oecd)
+ f4() *f5(almp-unem)
+ f6() +f7(udnet, udnet-vis)
+ f8() +f9(co, co-oecd, co-oecd-int, cow, cow-int, ce-oecd, ce-oecd-int, cew, cew-int)
+ f10()*f11(co, co-oecd, co-oecd-int, cow, cow-int, ce-oecd, ce-oecd-int, cew, cew-int)
+ f12()*f13(Inflation, Inf-Change, RIR)
+ f14() +f15(Year, ho, ho-oecd, ho-comb, urban)
```

Fig. 7.2 Example Eureqa goal formula – complex form

Table 7.3 Two complex regression champions from Eureqa

Complexity	In sample R^2	Numeric constants	Formula
26	0.82	6	UNEM = 2.354 + 0.22*benefit-replacement-rate-oecd + factorial(-0.01192*home-owner- rate-comb) − 0.05175*almp-unem − 1.299* bargaining-coordination-oecd − 2.213*benefit- duration
35	0.84	8	UNEM = 4.407 + UNEM = 0.2265*brr-oecd + 70.58/(6.68 + almp-unem) + factorial(−0.01179*ho-comb) − 3.698 − 1.132*co-oecd − 1.029*tan(powerof(bd,2))

deeply nested sets of brackets. The format shown in Fig. 7.2 is required to produce a regression model in the form shown in Table 7.1 (i.e. each constant operates on exactly one predictor variable and there are no nested brackets).

Eureqa allows the specification of grouped variables, thus on line 2 the target does not name a single variable but rather five possible variables any one of which might be selected as the best candidate to represent the benefit replacement ratio variable in the LNJ formula. Two of the champion formulas from a universal Eureqa search are shown in Table 7.3.

For each level of complexity Eureqa shows only one regression champion. At the time of writing it does not appear there is a method to constrain Eureqa to produce a variety of models at a single complexity level. For this reason most of the searches and plausibility analysis will center on ARC.

6 Testing with ARC

Next, an attempt was made to replicate the LNJ formula in Table 7.1 using ARC and RQL. Symbolic regression seems particularly well suited to exploring relationships in our labor market institutions database because often there are many slightly different versions of a variable designed to represent the same characteristic. Whereas in Table 7.1, the authors used a single variable to capture the benefit

replacement rate in Nickel's database (Nickell 2006) there were no less than six benefit replacement variables. In this situation, we should specify a symbol for all six possible variables rather than mention one specific one.

In general, the islands have been specified to take the general form of the LNJ formula save that one extra term has been added which is a pool of socio-demographic variables (describing home ownership and urbanization).

It should also be noted that the search process was allowed to choose among three possible macroeconomic variables that might affect consumer demand. The original LNJ model used the change in inflation (the current year's inflation rate minus that of the previous year). This search also allows ARC to choose the simple inflation rate or the real interest rate.

However, the same search produced significantly better results when the scoring process penalized formulas with large outliers (including negative unemployment rates anywhere in the set of predicted values). The customized penalties can be expressed in plain language as follows:

1. Where any unemployment rate is higher than 40% the error was multiplied by 1.015
2. Where any unemployment rate is lower than 0 the error was multiplied by 1.015

A similar penalty was applied to models with at least one insignificant coefficient: where any P-Value exceeded 0.15, the error was also multiplied by 1.015. It is important to stress that such models were not excluded from the evolution. There survivability was merely impaired.

Table 7.4 shows the result of the ARC search where the scoring penalties for minimum and maximum values were applied during the evolution. The R^2 of 0.89 is only slightly lower than that in Table 7.1. However, it was based on 143 records compared to the 20 records of the LNJ regression (where there was only one record per country and one time period based on a 5 year average).

Many labor market economists would object to the regression model shown in Table 7.4 because the signs of the benefit variables do not match the conventional theory. It is common in economic journals to read a smug analysis of a regression model which declares proudly "all the signs are in the expected direction." In Table 7.4 the benefit duration appears to lower unemployment but the benefit replacement variable increases it, whereas in Table 7.1 as both the benefit variables increase the jobless rate also increases. This type of dilemma illustrates a philosophical difference between genetic programmers and many economists. Enthusiasts for evolutionary computation claim that the world is a complex non-linear place and that genetic algorithms are about to make a major contribution to the understanding of it. At their best economists demand some clear theory that underlies a regression model. At their worst they search for the highest R^2 that will confirm a pre-conceived ideological prejudice. Plausibility measures such as the Out of Sample R^2 should afford an objective means to resolve this dilemma for they show how closely a given model helps to map out the real world. Tables 7.4–7.7 are the result of a two stage process. The first stage involved a search using ARC.

Table 7.4 A modified LNJ formula

RSQ = [0.89] Sign	Coefficient	Variable	T-statistic	P-values
−	4.284531	cube(benefit_duration)	−10.25	0.0000
+	7.404359	ln(benefit_replacement_rate_oecd)	20.91	0.0000
−	2.528383	ln(active_labour_per_unemployed)	−11.61	0.0000
+	8.554377E-38	exp(union_density)	3.63	0.0000
−	0.007248	quart(bargaining_coordination_oecd)	−7.56	0.0000
−	0.0135653	quart(bargaining_coordination_w)	−2.00	0.0480
−	0.7791816	ln(Inflation)	−4.09	0.0000
+	2.662730E-07	quart(home_owner_rate)	18.32	0.0000
−	11.1052	Constant	−10.11	0.0000
Plausibility measures			Predicted	Actual
Highest unemployment rate			21.1	23.9
Lowest unemployment rate			−1.4	2.7
Out of sample R-Square			0.02	

Table 7.5 A Hayek libertarian formula

RSQ = [0.88] Sign	Coefficient	Variable	T-statistic	P-values
+	0.894822	quart(benefit_duration)	1.89	0.0610
+	10.172370	ln(benefit_replacement_rate_oecd)	25.16	0.0000
−	4.724046	ln(active_labour_per_unemployed)	−16.86	0.0000
+	1.078187E-07	quart(union_density)	6.24	0.0000
−	4.081558E-07	quart(minimum_wage_median)	−10.73	0.0000
−	0.000352	cube(direct_tax_rate)	−14.05	0.0000
−	0.096746	abs(inflation)	−2.02	0.0450
+	0.087163	home_owner_rate_oecd	12.94	0.0000
−	15.825530	Constant	−11.37	0.0000
Plausibility measures			Predicted	Actual
Highest unemployment rate			22.4	23.9
Lowest unemployment rate			−1.9	2.7
Out of sample R-Square			0.47	

The same models were then recreated inside the econometric package Stata using its standard ordinary least squares procedure. The T-statistics and P-Values in these tables are the standard Stata output of this procedure.

The Table 7.4 should be a solemn warning to many social scientists who publish regression models; it shows how easy it is to strike fool's gold. Table 7.4 shows that seven of the eight terms have a P-Value below 0.01. All of them are significant at 0.05. It has a respectable R^2 of 0.89. With the exception of the benefit duration term, the signs of all the coefficients are in the correct direction (i.e. the direction implied by the prevailing economic theory). However as soon as we require this champion to do work in the real world it collapses. As can be seen from the bottom line of Table 7.4 the out of sample R^2 was only 0.02 implying that it could only explain a tiny proportion of unemployment rates from 1994 to 2000.

Table 7.6 A 'Progressive' formula

RSQ = [0.87] Sign	Coefficient	Variable	T-statistic	P-values
−	0.1299335	benefit_replacement_years 2&3	−6.57	0.0000
+	11.41657	ln(benefit_replacement_oecd)	14.58	0.0000
−	3.062043	ln(active_labour_per_unemployed)	−11.02	0.0000
+	6.024475E-38	exp(union_density)	2.36	0.0200
+	2.837324	bargaining_coordination_w	6.51	0.0000
−	1.550813	bargaining_coordination_oced	−8.19	0.0000
−	0.2318587	abs(real_interest_rate)	−4.50	0.0000
+	2.197473E-07	quart(ho_oecd)	12.91	0.0000
−	21.35325	Constant	−10.53	0.0000
Plausibility measures			Predicted	Actual
Highest unemployment rate			20.5	23.9
Lowest unemployment rate			−0.2	2.7
Out of sample R-Square			0.51	

Table 7.7 Formula with a polynomial benefit replacement effect

RSQ = [0.85] Sign	Coefficient	Variable	T-statistic	P-values
−	0.296193	benefit_replacement_rate_oecd	−4.95	0.0000
+	13.831420	ln(benefit_replacement_rate_oecd)	9.31	0.0000
−	2.942167	ln(active_labour_per_unemployed)	−8.19	0.0000
+	4.193654E-08	quart(union_density_vis)	2.68	0.0080
−	1.806615	abs(bargaining_coordination_oecd)	−8.45	0.0000
+	3.164897	abs(bargaining_coordination_w)	6.11	0.0000
−	0.240120	real_interest_rate	−4.31	0.0000
+	2.202466E-07	quart(ho_oecd)	11.59	0.0000
−	24.207450	Constant	−7.46	0.0000
Plausibility measures			Predicted	Actual
Highest unemployment rate			20.7	23.9
Lowest unemployment rate			1.2	2.7
Out of sample R-Square			0.47	

Genetic programmers are often accused of producing overfit models that mean nothing in the real world. Researchers who use non-evolutionary methods should answer the same charge. The in-sample R^2, T-Statistics and P-values shown in Table 7.4 look entirely respectable within the relevant academic culture. However when one looks at the out-of-sample R^2 the model appears to be virtually useless as a predictor of actual events. How many of the models published in journal articles are equally weak predictors? Of their 100 champion formulas produced by this search none had an out of sample R^2 higher than 0.04. Let us see if some variations on the LNJ formula can produce more convincing results.

7 Virtual Economists

One of ARC's most interesting features is the ability to specify multiple "islands" in the search space that will evolve simultaneously. To demonstrate the power of this feature a set of ideological islands were created with the same general form as the model shown in Table 7.4 but with pools of variables that represent different ideological preferences. We will run simulations with the existing modified LNJ formula that produce Table 7.4. We will call this island *Pragmatic Centrist* because some of its ideological underpinnings seem to lean towards the political right (the effect of the benefit system) and some to the political left (support for bargaining coordination).

The second island we will call *Hayek Libertarian* after the champion of the free market Freidrich von Hayek. Hayek would be able to agree that the effect of the benefit system will push up unemployment. However, Hayek (1994) was a firm opponent of attempts to plan economic behavior so any attempt to promote wage bargaining coordination or centralization would have been anathema to him. On this island, inhabitants can hunt for relationships linked to certain libertarian issues such as taxation and the minimum wage. However, they are not allowed to include any variables describing bargaining centralization or coordination. The island also includes a large number of variables that describe the extent of market regulation and possible disincentives caused by the tax system.

The last island we will call *Progressive.* Our progressive islanders accept most of the arguments underpinning the LNJ model but are unwilling to consider reducing the duration of unemployment benefits. However, they are willing to try to change the benefit replacement ratio. It should be stressed that this can be done in different ways. A conservative might wish to cut the real value of the benefits. A progressive might try to reduce the replacement ratio by raising the incomes of the poorest quarter of wage earners. This could be done by increasing benefits paid to poorer workers such as the UK's Family Credit or the USA's Earned Income Tax Credit (and ending the "unemployment trap" is often claimed as a significant reason for doing so).

There was an attempt to define a socialist island based on the views of the British economist Nicholas Kaldor. The inhabitants of this island were not allowed to consider any changes to the benefit system. Since Kaldor was a strong supporter of the UK's post World War II welfare state (King 2009) it is reasonable to assume that he would not wish to lower unemployment by making benefits less generous. It is also reasonable to assume he would have supported wage bargaining coordination because he supported incomes policies (Kaldor 1982) and was an admirer of West Germany's industry-wide labor unions (Targetti 1992). Unfortunately, the removal of all benefit variables from the search goal proved to be an excessively severe restriction. The '*Kaldorian Socialist*' island produced no plausible winners at all (all the models had negative values for the out-of-sample R^2).

Results of Hayek Libertarian Island

The Hayek Libertarians managed to find several models within their ideological research constraints. Table 7.5 shows one of them. It achieves an in sample R^2 of 0.88 without using any wage bargaining variables. The sign of both benefit system variables was positive which is the direction implied by the LNJ formula (Layard et al. 1991), however many other plausible models developed for the Hayekian island had a positive sign for the benefit replacement rate and a negative sign for benefit duration. The prevailing argument is that the shorter the time when the jobless receive cash benefits the lower will be the unemployment rate. However, where the search process can select multiple benefit variables the varying signs may indicate that the effect of benefit system generosity does not increase unemployment monotonically or exponentially. The varying positive and negative signs for models with high out-of-sample R^2 imply the true relationship may be polynomial. This conflicts with the culture created by academic peer pressure in economics where any regression models putting a negative sign before a benefit duration coefficient would probably be un-publishable.

Another area of potential controversy in Table 7.5 is the impact of the minimum wage. If Hayek were alive to comment on Table 7.5 he could only smile with a lop-sided grin because the minimum wage variable was selected but with a negative sign. This is a case where a good in-sample fitness score should be sacrificed in the interests of robustness: if one drops the minimum wage variable from the model the in-sample R^2 drops from 0.88 to 0.79 but the Out of Sample R^2 rises from 0.47 to 0.60. The direct tax rate also has a negative sign, which is also contrary to the Hayekian argument. The model shows a negative sign for active labour market spending per unemployed person and a negative sign for inflation which both are in line with the published research.

Results of the 'Progressive' Island

As mentioned above the 'Progressive' islanders were allowed to analyze the benefit system but did not want to contemplate cutting the time for paying unemployment benefit. They were allowed two use two benefit system variables (but not benefit duration). Similarly, they were allowed to use two wage bargaining system variables. The results look encouraging in terms of the P-Values. Seven out of eight are under 0.001 and the other is under 0.05. The out of sample R^2 of 0.51 also looks respectable. However one of the predictions in the out of sample period was negative. Another difficulty is that where there are two variables that cover the same general topic the signs of the coefficients do not match. In the LNJ Formula, both benefit duration and benefit generosity (the replacement rate) increase unemployment (both had a positive sign). Similarly, both employee bargaining coordination and employer coordination decrease it (both have a negative sign).

How can we explain the fact that one coefficient for the benefit replacement rate is positive while the other is negative. Is this just the result of an implausibly over-fitted model? If so, why does the formula in Table 7.6 explain so much of the variation in unemployment rates from 1994 to 2000? Philosophically one can argue that whatever model achieves the highest out of sample R^2 is the model that should be used by policy makers in the real world. The same arguments apply to the coefficients for the two wage bargaining variables in Table 7.6. Many economists would reject the model on this account. A genetic programmer might argue this is example of the subtle non-linear relationships that evolutionary algorithms are intended to uncover. Unfortunately, the coefficients in Table 7.6 do not take the form of an acceptable polynomial. For that to be the case *the same variable* would need to appear twice with both positive and negative coefficients.

In order to explore the concept of such a non-linear relationship between replacement rates and unemployment, several islands were defined with the same format as the progressive search goal except that the same benefit variable was repeated in the first and second positions. The different benefit variables tested in this way were brr1 (the replacement rate during the 1st year of unemployment), brrl (the long-term replacement rate), brr-oecd (a summary measure intended to represent the replacement rate across all time periods). A final island was defined with bd (the benefit duration variable) included since the format of this variable in the CEP-OECD database captures both the duration and generosity of the payments. ARC's RQL language is possibly the only SR package that would allow this type of highly customized multi-island search to be defined. Of the 400 regression champions produced by this search, only the brr-oecd variable produced any plausible models. The model with the highest out of sample R^2 is shown in Table 7.7. This model produced no negative unemployment rates in either time period. Here we see a large positive coefficient applied to the log of benefit replacement combined with a small negative coefficient on brr-oecd in an unmodified form.

A crude plain language explanation of this might be as follows: unemployment benefits that are extremely ungenerous have a powerful effect in lowering unemployment but this effect tapers away as benefit payments approach the average incomes of the lower paid. This has important political implications because a government could spend billions of dollars on in-work benefits to change the replacement ratio from say 80 to 70% without much impact on unemployment. The really strong unemployment-reduction effects would be seen in countries with tiny replacement rates (such as Japan where there no benefit at all after the first year). If this non-linearity of benefit levels is true then it lends further weight to Layard and Nickell's call for a job guarantee (2011) rather than benefit reform. Under this scheme the state would have a duty to offer work or training after the first year of unemployment and the jobless would have a corresponding duty to accept it.

Table 7.8 Plausible models by grammar depth and number of variables

	(1) Number of minimally plausible models (out of sample RSQ > 0)	(2) Number of highly plausible models (out of sample RSQ > 0.3)	(3) Most plausible model (highest out of sample RSQ in group)
9 variables 0 depth	1	1	0.42
8 variables 0 depth	2	1	0.30
7 variables 0 depth	19	5	0.42
9 variables 1 depth	14	14	0.66
8 variables 1 depth	12	9	0.57
7 variables 1 depth	5	4	0.39
9 variables 2 depth	2	2	0.39
8 variables 2 depth	0	0	0.00
7 variables 2 depth	18	16	0.56

Fertility of Different Parts of the Search Space

In Table 7.8 we address the question of whether one part of the search space yields higher proportions of plausible models. Universal searches were specified at three "grammar depths". At a grammar depth of zero only linear models can be formed. At a grammar depth of one the variables may be modified by a single operator (e.g. cube(benefit-duration)). At a depth of two the functions may be nested which in the case of unary operators creates terms such as cube(exp(benefit-duration)). It is difficult to detect a consistent pattern but intuitively the models at a grammar depth of one produced the largest number of plausible models (those where the out of sample R^2 was over 0.3). It would be tempting to look at Table 7.8 and assume that the model that achieved the highest ORSQ was the *overall winner* but one of its features would deny it acceptability among economists: it uses no less than three home ownership variables out of its total of seven terms. With universal goals there is no way to constrain ARC from picking the same or similar variables multiple times.

8 Conclusion

Both Eureqa and ARC offer tools to regression modelers that go significantly beyond the power offered by stepwise regression in packages like Stata, SPSS and SAS. ARC, in particular, allows the statistical explorer to fine-tune the target model in a way that adds real value for the social scientist.

Consider the process of using a package like Stata. Stepwise regression can be used to select independent variables and the best fitting model predicts that some countries have negative unemployment rates. There is no automated process to find plausible models where all the predicted rates are positive and which can predict out of sample data with any accuracy.

The most telling advantages of SR and ARC in particular are the ability to specify multiple islands on the same run and to control the grammar depth to be searched.

The Greek goddess of hunting, Artemis, had a dog called Lealaps with a singularly useful quality: it always caught its prey. Those developing symbolic regression systems are engaged in a similar pursuit. This paper demonstrates the ability to find a powerful unemployment prediction model using an automatic universal search. This universal search produced an out of sample R^2 of 0.45 which is creditable, but this figure was exceeded by the models shown in Tables 7.5–7.8 which were inspired by the research of famous economists. Our cybernetic Lealaps is useful, but perhaps he is even more effective when leashed to a human expert.

References

Blanchard O, Wolfers J (1999) The role of shocks and institutions in the rise of European unemployment: the aggregate evidence (working paper no. 7282). National Bureau of Economic Research

de Tocqueville A, Reeve H, Commager HSJ (1952) Democracy in America. Oxford University Press, London

Hayek FA (1994) The road to serfdom. University of Chicago Press, Chicago

Kaldor N (1982) The scourge of monetarism. Oxford University Press, New York

King JE (2009) Nicholas Kaldor. Palgrave Macmillan, New York

Korns MF (2007) Large-scale, time-constrained symbolic regression. In: R. Riolo, T. Soule, & B. Worzel (Eds.), Genetic programming theory and practice IV. Springer, US pp. 299–314. http://www.springerlink.com/content/p77g8gv231v37538/abstract/

Korns MF (2010) Symbolic regression of conditional target expressions. In: R. Riolo, U.-M. O'Reilly, & T. McConaghy (Eds.), Genetic programming theory and practice VII. Springer, US pp. 211–228. http://www.springerlink.com/content/n1m456589un33317/abstract/

Korns MF (2011) Abstract expression grammar symbolic regression. In: R. Riolo, T. McConaghy, & E. Vladislavleva (Eds.), Genetic programming theory and practice VIII. Springer, New York 8:109–128. http://www.springerlink.com/content/m763q88030380332/abstract/

Layard R, Nickell SJ, Jackman R (1991) Unemployment: macroeconomic performance and the labour market. Oxford University Press, New York

Layard PRG, Nickell SJ, Eichhorst W, Zimmermann KF (2011) Combatting unemployment. Oxford; New York: Oxford University Press

Nickell SJ, Nunziata L, Ochel W, Quintini G. (2001) The Beveridge Curve, Unemployment and Wages in the OECD from the 1960s to the 1990s - Preliminary Version (CEP Discussion Paper). Centre for Economic Performance, LSE. http://econpapers.repec.org/paper/cepcepdps/dp0502.htm

Nickell W (2006) The CEP-OECD institutions data set (1960–2004) (CEP discussion paper no. dp0759). Centre for Economic Performance, London School of Economics

Nickell S, Nunziata NL, Ochel W, Quintini G (2003) The Beveridge curve, unemployment and wages in the OECD from the 1960s to the 1990s. Princeton University Press, Princeton

Scmidt M (2013) Eureqa (version 0.98 beta)[software]. Available from http://www.eureqa.com

Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324(5923):81–85

Targetti F (1992) Nicholas Kaldor: the economics and politics of capitalism as a dynamic system. Oxford University Press, New York

World Bank (2012) Databank. The World Bank, Washington, DC. Retrieved on 24/8/2012 from http://data.worldbank.org

Chapter 8
Uniform Linear Transformation with Repair and Alternation in Genetic Programming

Lee Spector and Thomas Helmuth

Abstract Several genetic programming researchers have argued for the utility of genetic operators that act uniformly. By "act uniformly" we mean two specific things: that the probability of an inherited program component being modified during inheritance is independent of the size and shape of the parent programs beyond the component in question; and that pairs of parents are combined in ways that allow arbitrary combinations of components from each parent to appear in the child. Uniform operators described in previous work have had limited utility, however, because of a mismatch between the relevant notions of uniformity and the hierarchical structure and variable sizes of many genetic programming representations. In this chapter we describe a new genetic operator, ULTRA, which incorporates aspects of both mutation and crossover and acts approximately uniformly across programs of variable sizes and structures. ULTRA treats hierarchical programs as linear sequences and includes a repair step to ensure that syntax constraints are satisfied after variation. We show that on the drug bioavailability and Pagie-1 benchmark problems ULTRA produces significant improvements both in problem-solving power and in program size relative to standard operators. Experiments with factorial regression and with the boolean 6-multiplexer problem demonstrate that ULTRA can manipulate programs that make use of hierarchical structure, but also that it is not always beneficial. The demonstrations evolve programs in the Push programming language, which makes repair particularly simple, but versions of the technique should be applicable in other genetic programming systems as well.

L. Spector (✉)
Cognitive Science, Hampshire College, Amherst, MA, USA
e-mail: lspector@hampshire.edu

T. Helmuth
Computer Science, University of Massachusetts, Amherst, MA, USA
e-mail: thelmuth@cs.umass.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_8,

Keywords Uniform mutation • Uniform crossover • ULTRA • Push • PushGP • Drug bioavailability problem • Pagie-1 problem • Factorial regression • Boolean multiplexer problem

1 Introduction

One of the essential components of any genetic programming system, and of any evolutionary algorithm more generally, is the set of genetic operators used to produce genomes of children from genomes of parents. The operators that are used most commonly in genetic programming stem from Koza's work and involve the replacement of single subprograms either with newly generated random subprograms (for mutation) or with subprograms taken from other programs in the population (for crossover) (Koza 1992). Numerous alternatives have since been proposed, on the basis of a wide variety of motivations, including operators designed specifically to control program size (e.g. Langdon 2000; Crawford-Marks and Spector 2002), operators specialized for specific program element types (e.g. Schoenauer et al. 1996), operators that respect or enforce homology between parents (e.g. D'haeseleer 1994; Langdon and Poli 2002), and operators that combine the semantics of parents in tailored ways (e.g. Moraglio et al. 2012). A brief summary of several alternative operators is available in Poli et al. (2008).

In the present chapter we are primarily concerned with operators that act uniformly across programs. By "act uniformly" we mean two different things, both of which have been designated as "uniform" by others as well. In the first place we mean that the probability of an inherited program component being modified during inheritance is independent of the size and shape of the parent programs beyond the component in question. This property does not hold for standard subtree crossover or mutation because components deeper in trees are more likely to be replaced or modified when those operators are used. Furthermore, two equal components at the same depth of different trees will have different probabilities of being replaced depending on the rest of the tree; the larger the rest of the tree, the less likely the component is to be changed. This property does hold to some extent, however, for versions of "uniform subtree mutation" that have been described in the genetic programming literature (Van Belle and Ackley 2002). In the second place we are interested in operators that combine pairs of parents in ways that allow arbitrary combinations of components from each parent to appear in the child. Again, this property does not hold for standard subtree crossover, which replaces a single subtree of one parent with a single subtree of another parent. This property does hold to some extent, however, for previously-presented versions of "uniform crossover" (Page et al. 1998). Both properties hold in some traditional genetic algorithms with linear, fixed-length genomes.

As noted in the work cited above, there are several reasons to think that uniformity of both kinds may be helpful, including possible beneficial effects on program size control and search space coverage. For example, if genetic operators

are uniform then program growth through the accumulation of non-functional code will not confer "protection from crossover" on a program's functional code, so "program bloat" (and consequent decreases in program search efficacy) due to this cause should be eliminated (Luke and Panait 2006).[1] However, the operators described in previous research have been limited in their potential applications, largely because of a mismatch between notions of uniform action and the hierarchical structure and variable sizes of genetic programming representations.

For example, the uniform crossover operator developed by Poli, Langdon and Page swaps nodes only among parts of parent trees that are structurally equivalent between the two parents, and the uniform mutation operator that they use can only change nodes to other nodes of equivalent arity (Page et al. 1998; Poli and Langdon 1998; Poli and Page 2000). This means that non-matched subtrees of parents can appear in children only in an all-or-nothing fashion, and that mutations cannot change tree shapes. In addition, the hierarchical nature of the crossover operator biases the mixture differently at different tree depths. Furthermore, the amount of mixing permitted would be diminished in the context of strong typing or other enrichments of the program representation. The approach in this work is to vary programs uniformly where it is safe and clear how to do so. Poli et al. have demonstrated, both theoretically and empirically (with even-n-parity problems) that this approach can have significant benefits. But even in the context of simple, single-type genetic programming it is not clear how to vary programs uniformly everywhere, and even where it is clear the variations that this method produces are not fully uniform in all of the senses that might be helpful.

More recently Semenkin and Semenkina have extended the methods of Poli et al., specifying random selection of parental components when arities are mismatched. They also allowed the ratio of parent contributions to be adaptively controlled (Semenkin and Semenkina 2012). While this work increases the amount and variety of mixing that can be produced, the constraints on the application of the technique and the deviations from uniformity are largely unchanged from the prior work.

Other researchers have explored an approach to uniform mutation involving iterative applications of subtree replacement, with the number of iterations depending on the size of the program (Van Belle and Ackley 2002). This allows mutation to produce arbitrary changes in tree shape and it helps to decouple the chances of a node being modified from the size and shape of the remainder of the program, but it is still subject to significant depth-based biases in modification probabilities. Experiments with this technique have demonstrated improvements in program size control but less clear results with respect to problem-solving power (Van Belle and Ackley 2002). In other work, different genetic operators have been applied to programs of different depths, leading to somewhat more uniform variation than is produced by standard systems (Kennedy and Giraud-Carrier 1999).

[1]We make no claims here about the prevalence of "protection from crossover." This is just one example of the effects that operator uniformity can have on program sizes and on genetic programming search efficacy; other effects may interact with other hypothesized causes of program bloat in other ways.

The approach to uniform variation that we describe in this chapter differs from that of the past work by prioritizing uniformity (of both kinds): we designed our single new genetic operator, which incorporates aspects of both mutation and crossover, in a way that causes uniformity to take precedence over the effects of program shape and size. We did this, essentially, by ignoring the syntactic structure of programs during the first phase of the action of the operator. This "syntax blindness" can produce children that violate syntactic constraints, so we must follow the syntax-blind variation step with a repair step that ensures or restores syntactic validity. While we do not claim that our new operator is "perfectly" uniform in the sense that we are using that term, we do believe that it is more uniform than other operators described in the literature and that its good performance is a consequence of this fact.

In the following sections we first describe the PushGP genetic programming system, within which all of our demonstrations are conducted; Push's minimal syntactic constraints make the repair step of our method particularly simple. We then describe our new operator, which we call ULTRA (for "Uniform Linear Transformation with Repair and Alternation"). We then demonstrate the utility of ULTRA on several problems. Our demonstrations include applications to the drug bioavailability and Pagie-1 benchmark problems, for which ULTRA provides dramatic improvements both in problem-solving power and in control of program size. We also demonstrate the utility of ULTRA on a factorial regression problem that involves greater use of hierarchical program structure, again documenting significant improvements both in problem-solving power and in control of program size. Finally, we include results of an application to a Boolean multiplexer problem, for which the results are mixed. Following these demonstrations we conclude with some comments about directions for future research.

2 Push and PushGP

Push is a programming language that was designed specifically for use in evolutionary computation systems, as the language in which evolving programs are expressed (Spector 2001; Spector and Robinson 2002; Spector et al. 2005). Push is a stack-based programming language that is similar in some ways to others that have been used for genetic programming (e.g. Perkis 1994). It is a postfix language in which literals are pushed onto data stacks and instructions act on stack data and return their results to stacks.

One novel feature of Push is that a separate stack is used for each data type. Instructions take their arguments (if any) from stacks of the appropriate types and they leave their results (if any) on stacks of the appropriate types. This allows instructions and literals to be freely intermixed regardless of type while still ensuring execution safety. By convention, instructions that find insufficient data on the relevant stacks act as "no-ops"—that is, they do nothing.

Many of Push's most unusual and powerful features stem from the fact that code is itself a Push data type, and from the fact that Push programs can easily (and often do) manipulate their own code as they run. Push programs may be hierarchically structured with parentheses, and this hierarchical structure affects how code-manipulation instructions work. It also affects the ways that traditional genetic operators operate on programs, just as the analogous structure of tree-based programs affects the ways that traditional genetic operators operate on them. In the most standard configuration PushGP uses mutation and crossover operators that are almost identical to those used in tree-based genetic programming, with mutation replacing a sub-expression (a literal, an instruction, or a parenthesized code fragment) with a newly generated sub-expression, and with crossover replacing a sub-expression with a sub-expression randomly chosen from another program in the population.

Push and PushGP implementations have been written in C++, Java, JavaScript, Python, Common Lisp, Clojure, Scheme, Erlang, Scala and R. Many of these are available for free download from the Push project page.[2]

3 The ULTRA Operator

"ULTRA," which stands for "Uniform Linear Transformation with Repair and Alternation," is a new genetic operator that takes two parent programs and produces one child program. ULTRA acts on hierarchically structured programs but treats them as linear sequences. It uses each element of the parent sequences with uniform probability and modifies each element of the resulting child sequence with uniform probability. It was motivated by theoretical considerations regarding relations between program size, function, and mutability, and by analogies to the mechanics of mutation and crossover in biological (linear) genomes. We will describe ULTRA here in terms of the elements of Push programs, but the operator could be used on program representations with suitable modifications.

ULTRA works by first "linearizing" each parent into a flat, depth-first sequence that includes a token for each literal, instruction, and delimiter (e.g. Push parentheses) in the parent program. It then pads the end of the shorter parent program with null tokens so that both parent programs are the same length. These tokens ensure that instructions in programs of different lengths have approximately equal probabilities of being included in the child, no matter where those instructions occur. The null tokens are removed from the child at the end of ULTRA.

ULTRA next traverses the linearized parents, building the child as a linear sequence of tokens taken from the parents. Traversal begins with a "read head" on the first token of the first parent, and the copying of that token to the child. After this

[2] http://hampshire.edu/lspector/push.html

and each subsequent step there is a fixed probability of alternating between parents; that is, of moving the read head to approximately the same location in the other parent program. The probability of alternating at any given step is specified as the "alternation rate." When alternating between parents, the position of the read head is subjected to Gaussian noise and may change to a higher or lower index; the standard deviation of the noise is given by the "alignment deviation" parameter. Note that alignment deviation may cause some elements of parent programs to be skipped or to be repeated in the child program. After deciding whether to alternate or not, the next token from the current parent is added to the child, and the read head is moved forward. The construction of the child terminates when the read head runs off the end of the current parent or when the child reaches the maximum program size.

After the child sequence has been created through this traversal, it is subjected to a uniform mutation during which each token has uniform probability of being deleted or replaced by a randomly chosen literal, instruction, or delimiter. The probability of any specific token being mutated is given by the "mutation rate" parameter.

In the absence of mutations or alternations, ULTRA would simply traverse the first parent and copy all of its tokens to the child; the child would then be a clone of the first parent. However, alternation and mutation may produce novel programs, some of which may be syntactically invalid; in these cases, the child program must be repaired by ULTRA's repair step.

Fortunately, Push programs are particularly easy to repair. Any sequence of valid instructions, literals, and parentheses is a syntactically valid Push program as long as its parentheses are balanced and as long as its outermost parentheses enclose the entire program. This means that ULTRA can repair a program by simply adding and/or deleting parentheses. Our repair algorithm traverses the child until an imbalance is detected and then fixes the imbalance either by deleting the source of the imbalance or by adding a matching parenthesis in a random location on the appropriate side of the imbalance. The complete repair algorithm requires two passes through the program, one in each direction, and minimizes structural bias arising from repair choices (as might occur, for example, if repairs were always accomplished by adding parentheses to the very beginning or very end of the program). After repair, the child sequence is transformed back into a hierarchical Push program. Finally, all null tokens are removed from the child.

As an example of the overall operation of ULTRA, consider a case involving the two parent programs "(a b (c (d)) e (f g))" and "(1 (2 (3 4) 5) 6)". After linearization the first parent has 15 tokens, while the second has only 12, so the second would be padded with 3 null tokens. Alternation might then produce a sequence of tokens like "(a b 2 (3 4 d)) 6) null null null", which has an extra ")". After repair and removal of null tokens we might have a valid child program such as "(a (b 2 (3 4 d)) 6)".

4 Experiments

To test the performance of ULTRA compared to standard genetic operators, we conducted runs of PushGP on four problems: drug bioavailability, Pagie-1 symbolic regression, factorial symbolic regression, and 6-multiplexer.

The drug bioavailability problem is a predictive modeling problem in which the programs must predict the human oral bioavailability of a set of drug compounds given their molecular structure (Silva and Vanneschi 2009, 2010). This problem has been used for genetic programming benchmarking in various studies (Silva and Vanneschi 2009; Harper 2012), and is recommended as a benchmark problem in a recent article on improving the use of benchmarks in the field (McDermott et al. 2012).[3] Each fitness case for this problem represents a molecule, with 241 floating point inputs, each of which represents a different molecular descriptor of the molecule, and a single floating point output representing the human oral bioavailability of that molecule. The dataset is available online.[4]

The Pagie-1 symbolic regression problem, proposed in Pagie and Hogeweg (1997), is a function on two variables of the form

$$f(x, y) = \frac{1}{(1 + x^{-4})} + \frac{1}{(1 + y^{-4})}.$$

Training set inputs are taken from the range $[-5, 5]$ in steps of 0.4, resulting in 676 fitness cases. This problem has also been used for benchmarking (Harper 2012), and has been recommended as a replacement for "toy" problems such as symbolic regression of the quartic polynomial (McDermott et al. 2012; White et al. 2013).

The factorial symbolic regression problem is an integer symbolic regression problem with one input and one output, in which the output should be the factorial of the input. We used 10 test cases, ranging from $1! = 1$ to $10! = 3{,}628{,}800$. Because error magnitudes vary significantly across cases we used "lexicase selection" instead of tournament selection for these runs. Lexicase selection is a parent selection algorithm that was developed to help solve problems that are "modal" in the sense that they require solution programs to perform qualitatively differently actions for inputs that belong to different classes, but it is also useful for problems in which error magnitudes are likely to vary significantly across cases. In lexicase selection a parent is selected by starting with a pool of potential parents—normally the entire population—and then filtering the pool on the basis of performance on individual fitness cases, considered one at a time (Spector 2012).

The 6-multiplexer problem (MUX6) is the standard boolean multiplexer problem used in Koza (1992) and in many subsequent studies by many authors.

In our experiments, we used the PushGP parameters listed in Table 8.1. We made an effort to use parameters similar to those used in previous work on these problems

[3] Recently, however, concern have been raised about the use of this problem; see http://jmmcd.net/2013/12/19/gp-needs-better-baselines.html

[4] http://personal.disco.unimib.it/Vanneschi/bioavailability.txt

Table 8.1 Parameters for experiments

Problem	Bioavailability	Pagie-1	Factorial	MUX6
Runs per condition	200	100	100	100
Population size	500	1,000	1,000	500
Max generations	100	1,000	500	200
Max program size	500	500	500	200
Max initial program size	500	500	100	200
Max size for mutation code	50	50	20	20
Parent selection tournament size	7	7	Lexicase	7

Table 8.2 ULTRA parameters used in our experiments

Problem	Bioavailability, Pagie-1, MUX6	Factorial
ULTRA mutation rate	0.01	0.05
ULTRA alternation rate	0.01	0.05
ULTRA alignment deviation	10	10

where possible. We used unbiased node selection for all subtree replacement operators. Table 8.2 presents the parameters we used for ULTRA.

For the bioavailability and Pagie-1 problems, we used the float stack instructions $add, sub, mult$, and div as the only non-input instructions. The bioavailability problem uses 241 input instructions, one for each molecular descriptor. As in Silva and Vanneschi (2009), we made input instructions and arithmetic instructions equally likely to be chosen by the random code generator. The Pagie-1 problem only requires the input instructions x and y. We also used the constant 1.0, but did not provide an ephemeral random constant.

For the factorial symbolic regression problem we used a more extensive function set that allowed for the manipulation of integers, boolean values, and the execution stack (to permit conditional branches and recursion), but we did not include Push's high-level iteration instructions that allow for trivial solutions. Specifically, we used the constants 0 and 1; an input instruction *in*; the boolean instructions *and, dup, eq, fromInteger, not, or, pop, rot*, and *swap*; the integer instructions *add, div, dup, eq, fromBoolean, greaterThan* (which pushes a boolean), *lessThan, mod, mult, pop, rot, sub*, and *swap*; and the exec instructions *dup, eq, if, noop, pop, rot, swap, when*, and the combinators k, s, and y (Spector et al. 2005).

Our instruction set for the 6-multiplexer problem included the boolean instructions and, or, and not, the exec stack instruction if, and the input instructions $a0$, $a1$, $d0$, $d1$, $d2$, and $d3$.

For some problems we conducted runs in multiple non-ULTRA conditions to show that the relative performance of ULTRA was not due solely to poor choices of parameters for the traditional genetic operators; in these cases we describe the runs with notation such as "81/9/10," meaning that the run used 81% subtree-replacement crossover, 9% subtree-replacement mutation, and 10% straight reproduction.

For all runs described here, fitness is defined as a measure of error, with lower numbers being better. For the bioavailability problem, we use root mean square error

(RMSE) as the fitness measure. On this problem, we separate the fitness cases into training and test sets by randomly selecting 70% of the fitness cases for training and 30% of them for testing. For this problem, we determine the statistical significance of whether the RMSE results of two runs come from the same distribution using the Kruskal-Wallis one-way analysis of variance at $p = 0.01$.

For the Pagie-1, factorial, and 6-multiplexer problems we used mean error across fitness cases and no separate test set. We present the number of successes and mean best fitness (MBF: the mean of the best individual fitnesses attained in each run) for these problems. The fitnesses given here are mean errors across test cases, not the sums of those errors. As recommended in Luke and Panait (2002) and McDermott et al. (2012), we use unpaired t-tests to compare differences in MBF for different conditions. For the factorial and 6-multiplexer runs, we also present the computational effort, which gives an estimate of the number of fitness evaluations required to have a good chance of finding a solution (Koza 1992; Niehaus and Banzhaf 2003).

5 Results

Figure 8.1 gives two box plots from our runs of the bioavailability problem, where each genetic operator setting was used in 200 runs. The left plot shows the root mean square error (RMSE) of the best program as measured on the training set. The right plot shows the RMSE of the same individuals on the test set. Both subtree replacement 81/9/10 and subtree replacement 45/45/10 differ statistically significantly from ULTRA on both the training and test sets. ULTRA appears to be able to find more accurate models of the training data than subtree replacement without overfitting the training data.

The mean program sizes with respect to generation are plotted in Fig. 8.2. The runs using subtree replacement show steady growth in program sizes, whereas those using ULTRA quickly fall at the beginning of the run and then remain relatively steady. The lower program sizes of ULTRA runs may contribute to the relative lack of overfitting (that is, better generalization) of the programs produced in these runs.

Table 8.3 presents the results of our experiments on the Pagie-1 problem. PushGP using ULTRA found perfect solutions in 11 out of 100 runs, whereas runs with subtree replacement found none with either parameter setting. The differences in MBF between subtree replacement 80/10/10 and ULTRA, and between subtree replacement 45/45/10 and ULTRA, are statistically significant at the $p = 0.01$ level according to an unpaired t-test.

The mean program sizes in our Pagie-1 experiments are shown in Fig. 8.3. Runs using subtree replacement experienced quick code growth, reaching mean sizes near the maximum program size of 500 within the first 50 generations. After this point, it is difficult for the genetic operators to make changes to large programs without

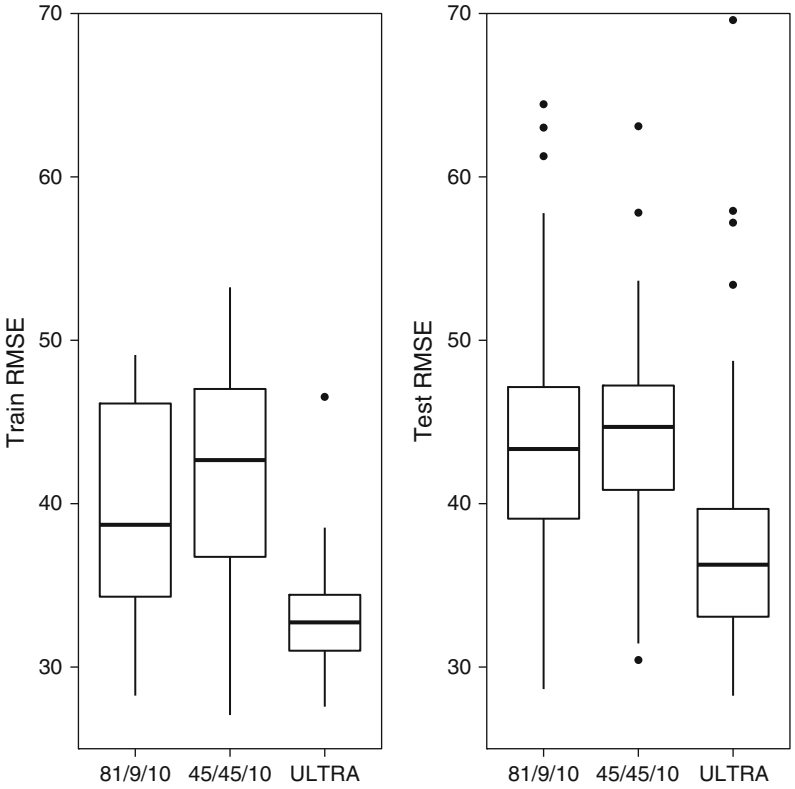


Fig. 8.1 Results from the bioavailability problem. We conducted 200 runs for each choice of operators. The RMSE of the best individuals on the training fitness cases (*left*) and on the test fitness cases (*right*). In each plot, subtree replacement 81/9/10 is plotted first, followed by subtree replacement 45/45/10 and then ULTRA. In each box plot, the box stretches from the first quartile to the third quartile with a line for the median in the middle. The whiskers extend to the furthest value within 1.5 times the inter-quartile range. Points beyond the whiskers are outliers, plotted as points. Note that in the *right plot*, 8 outliers in the 81/9/10 set, 5 outliers in the 45/45/10 set, and 4 outlier in the ULTRA set fell outside the of the visible plot

exceeding the program size limit. On the other hand, the mean program sizes of ULTRA runs quickly drop to around size 50, and then climb to approach 100.

Table 8.4 presents the results from our experiments using the factorial problem. ULTRA produced a better success rate and lower computational effort. The difference between the MBF subtree replacement 45/45/10 and ULTRA is statistically significant based on an unpaired t-test at $p = 0.01$

Mean program sizes for the factorial problem runs are presented in Fig. 8.4. The runs using ULTRA maintained a relatively constant mean program size, while runs using subtree replacement 45/45/10 show very fast code growth over the first 100 generations, followed by stable sizes near the maximum program size of 500.

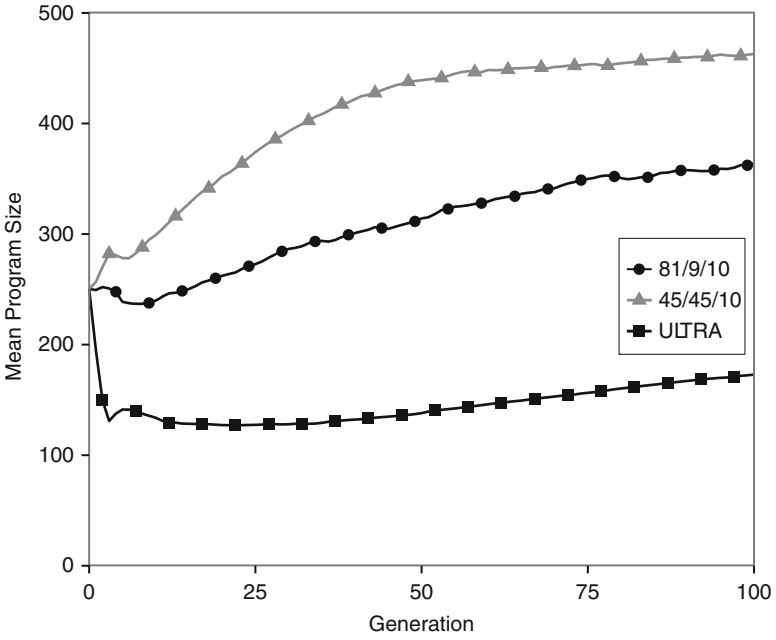


Fig. 8.2 Mean program sizes for the bioavailability problem

Table 8.3 Results on the Pagie-1 problem. We conducted 100 runs for each choice of operators. MBF is the mean best fitness of the run. Note that the reported fitnesses are the mean errors over test cases, not the summed errors

Operators	Successes	MBF
Subtree replacement 80/10/10	0	0.304
Subtree replacement 45/45/10	0	0.333
ULTRA	3	0.172

Table 8.4 Results on the factorial problem for 100 runs in each condition. CE is computational effort and MBF is the mean best fitness of the run. Note that the reported fitnesses are the mean errors over test cases, not the summed errors

Operators	Successes	CE	MBF
Subtree replacement 45/45/10	2	77,520,000	121,867
ULTRA	61	2,470,000	28,980

Table 8.5 presents results from our experiments on the 6-multiplexer problem. In contrast to the results on other problems presented here, subtree replacement performs better than ULTRA on all measurements of problem-solving performance. The difference between the MBF of subtree replacement 80/10/10 and ULTRA is statistically significant based on an unpaired t-test at $p = 0.01$.

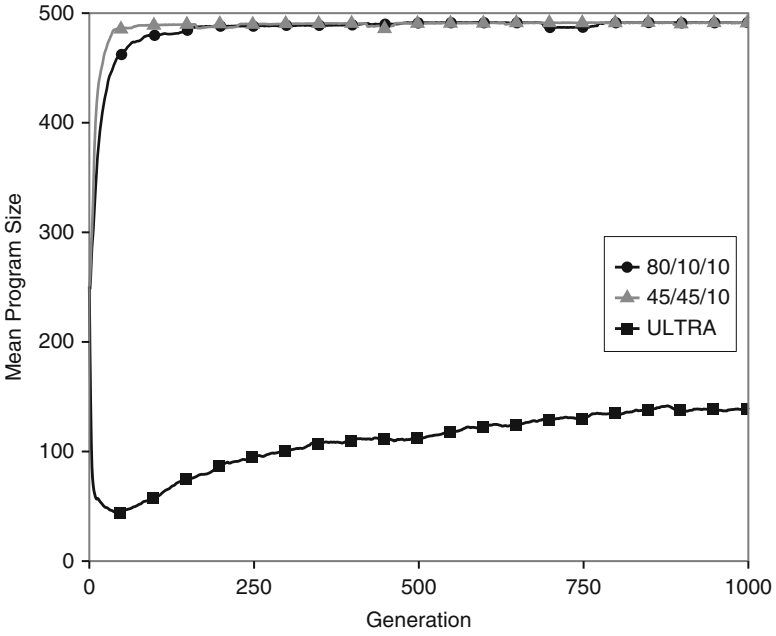


Fig. 8.3 Mean program sizes for the Pagie-1 problem

Program sizes for the 6-multiplexer problem are shown in Fig. 8.5. As we have seen before, sizes in subtree replacement runs grow rapidly and stay high, whereas sizes in ULTRA runs decrease rapidly and stay relatively low.

6 Discussion and Future Work

The results presented here demonstrate that ULTRA, a new genetic operator that prioritizes uniformity and incorporates features of both traditional mutation and traditional crossover, can be an effective tool in helping genetic programming to solve difficult programs and to manage program sizes over the evolutionary process.

The results on the drug bioavailability and Pagie-1 problems demonstrate that ULTRA can produce dramatic improvements both with respect to problem-solving power and with respect to managing program sizes. However, it should be noted that these problems do not rely on the hierarchical structure of Push programs when ULTRA is being used since they do not involve code manipulation instructions. A solution to one of these programs would, because of the way that the Push interpreter interprets programs, work just as well with its parentheses moved to different locations or eliminated entirely. Parentheses matter for these problems when traditional subtree-replacement operators are being used because parentheses

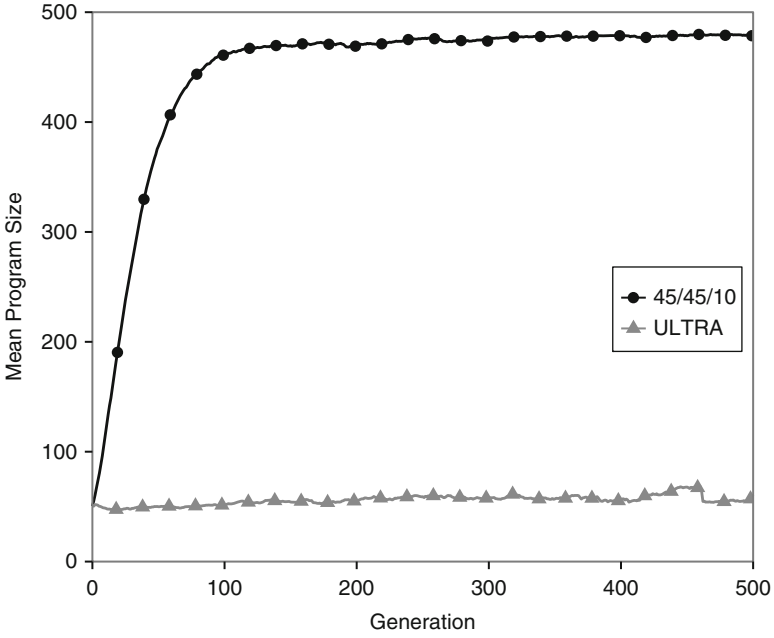


Fig. 8.4 Mean program sizes for the factorial problem

Table 8.5 Results on the 6-multiplexer problem, with 100 runs in each condition. CE is computational effort and MBF is the mean best fitness of the run

Operators	Successes	CE	MBF
Subtree replacement 80/10/10	85	135,000	0.009
ULTRA	58	369,000	0.038

delineate the units that can be replaced, but when only ULTRA is being used their effects would be limited to providing sites for insertion of new instructions via mutation, and for influencing the effects of deviations during alternation in minor ways.

For this reason we sought to demonstrate the use of ULTRA on a problem for which parentheses play a semantic role in the execution of programs, through the use of code manipulation instructions. The factorial symbolic regression problem that we demonstrated includes several instructions that are affected by the placement of parentheses. For example, the $exec\_if$ instruction will execute one of the two expressions that follows it and skip the other, depending on the value on top of the boolean stack. Since parentheses delineate the boundaries of these expressions, their placement is crucial. Several other instructions used in this problem, including the Y combinator instruction, rely on the placement of parentheses in a similar way. The fact that ULTRA performed so well on this problem indicates that it is capable

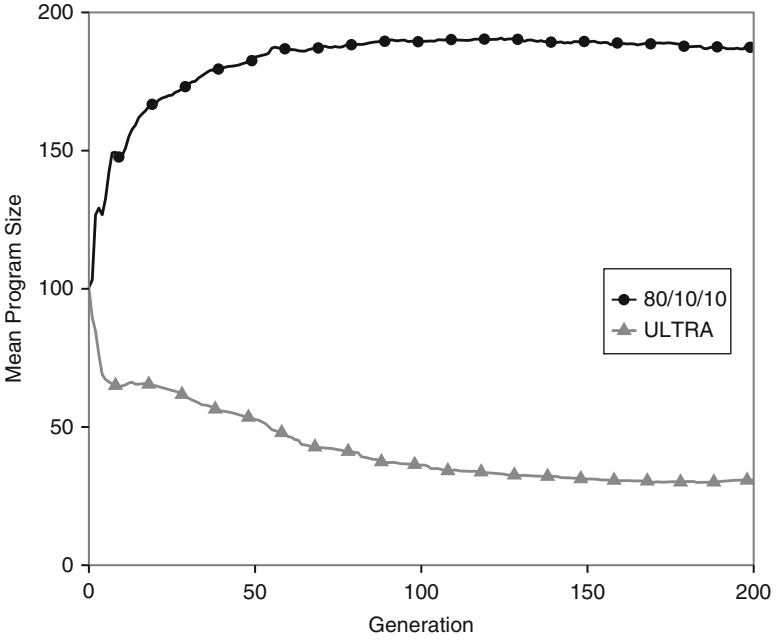


Fig. 8.5 Mean program sizes for the 6-multiplexer problem

of evolving hierarchically structured program representations even though it treats these programs as linear structures and uses a repair process to ensure that syntactic constraints (parentheses matching) is maintained.

In other work, not presented here, we have shown that ULTRA is also useful for the evolution of multiple-output digital multipliers (Helmuth and Spector 2013).[5]

The results on the 6-multiplexer problem indicate that ULTRA is not a panacea, at least not with the parameters that were used here. It may be the case that if we explored the space of parameters for ULTRA as well as subtree replacement operators that we would find settings that allow ULTRA to perform better than subtree replacement. On the other hand, these results may indicate that there aren't as many solutions to the 6-multiplexer problem at the smaller mean program sizes that ULTRA tends to produce.

The 6-multiplexer results lead to one obvious avenue for future research: How should ULTRA's parameters be set? We set them more or less arbitrarily for the runs presented here; conceivably we could develop guidelines for their values, based on characteristics of a problem, or mechanisms by which the parameters could be set adaptively over the course of a run.

[5]Note that this work gives an outdated description of ULTRA that does not pad the shorter program before alternation.

Another important avenue for future research concerns more rigorous analysis of when ULTRA has better performance than traditional subtree-replacement operators, when ULTRA produces smaller programs than traditional subtree-replacement operators, and when and how these two things are related to each other. Furthermore, the extent to which ULTRA is truly uniform should be studied more systematically. An initial examination of ULTRA on flat programs shows that instructions within the parent programs have approximately equal probability of being included in the child program, and that the mean program size of a child produced by ULTRA with differently sized parents is approximately the mean of the parent program sizes. But it is possible, for example, that program repair and/or alignment deviations near the beginnings or ends of parent programs will produce non-uniform effects. It would be worthwhile to investigate these issues further.

Finally, while the application of ULTRA to the evolution of Push programs is particularly simple—because Push program repair requires only the balancing of parentheses—we would like to look at the application of ULTRA in several different contexts. On the one hand it would be interesting to look at the application of ULTRA to programs in Push-like languages that do not even require parentheses, but rather use high-level instructions and/or markers within a linear program to delineate program structure. This might make it unnecessary to repair programs at all, and might provide both enhanced power and enhanced elegance.

On the other hand it would be interesting to look at the application of ULTRA to traditional tree-based genetic programming and to other genetic programming representations. For any such representation a repair mechanism will have to be developed that can re-establish syntactic constraints that may be violated by mutation and alternation. For example, in tree-based genetic programming, using traditional representations, it would be necessary not only to balance parentheses but also to ensure that only functions appear in function position (immediately after a "("), that only sub-expressions and terminals appear in non-function positions, and that the arities of functions are respected. Alternative representations, for example representations that omit parentheses and require structure to be inferred from the positions of functions and a table of function arities, might allow for simpler repair mechanisms. In any event, while repair in some representations will be more difficult than it is in Push, repair should nonetheless be possible and once a repair mechanism has been implemented one could use ULTRA with any program representation. Indeed, for some other representations, e.g. grammatical evolution with linear genomes (O'Neill and Ryan 2001), the implementation of ULTRA should be particularly straightforward.

Acknowledgements Thanks to Micah Savitzky for exploratory work that later led to the ideas in this paper, to Emma Tosch and Kyle Harrington for discussions and code, to Josiah Erikson for systems support, and to Hampshire College for support for the Hampshire College Institute for Computational Intelligence. This material is based upon work supported by the National Science Foundation under Grant Nos. 1017817 and 1129139. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Crawford-Marks R, Spector L (2002) Size control via size fair genetic operators in the PushGP genetic programming system. In: GECCO 2002: proceedings of the genetic and evolutionary computation conference, New York. Morgan Kaufmann, pp 733–739

D'haeseleer P (1994) Context preserving crossover in genetic programming. In: Proceedings of the 1994 IEEE world congress on computational intelligence, Orlando, vol 1. IEEE, pp 256–261

Harper R (2012) Spatial co-evolution: quicker, fitter and less bloated. In: GECCO '12: proceedings of the fourteenth international conference on genetic and evolutionary computation conference, Philadelphia. ACM, pp 759–766

Helmuth T, Spector L (2013) Evolving a digital multiplier with the PushGP genetic programming system. In: Workshop on stack-based genetic programming, Amsterdam. ACM, pp 1627–1634

Kennedy CJ, Giraud-Carrier C (1999) A depth controlling strategy for strongly typed evolutionary programming. In: Proceedings of the genetic and evolutionary computation conference, Orlando, vol 1. Morgan Kaufmann, pp 879–885

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Langdon WB (2000) Size fair and homologous tree genetic programming crossovers. Genet Program Evolvable Mach 1(1/2):95–119

Langdon WB, Poli R (2002) Foundations of genetic programming. Springer. http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/

Luke S, Panait L (2002) Is the perfect the enemy of the good? In: GECCO 2002: proceedings of the genetic and evolutionary computation conference, New York. Morgan Kaufmann, pp 820–828

Luke S, Panait L (2006) A comparison of bloat control methods for genetic programming. Evol Comput 14(3):309–344

McDermott J, White DR, Luke S, Manzoni L, Castelli M, Vanneschi L, Jaskowski W, Krawiec K, Harper R, De Jong K, O'Reilly UM (2012) Genetic programming needs better benchmarks. In: GECCO '12: proceedings of the fourteenth international conference on genetic and evolutionary computation conference, Philadelphia. ACM, pp 791–798

Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: Parallel problem solving from nature, PPSN XII (Part 1), Taormina. Lecture notes in computer science, vol 7491. Springer, pp 21–31

Niehaus J, Banzhaf W (2003) More on computational effort statistics for genetic programming. In: Genetic programming, proceedings of EuroGP'2003, Essex. Lecture notes in computer science, vol 2610. Springer, pp 164–172

O'Neill M, Ryan C (2001) Grammatical evolution. IEEE Trans Evol Comput 5(4):349–358. doi:10.1109/4235.942529

Page J, Poli R, Langdon WB (1998) Smooth uniform crossover with smooth point mutation in genetic programming: a preliminary study. Technical report CSRP-98-20, School of Computer Science, University of Birmingham. ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1998/CSRP-98-20.ps.gz

Pagie L, Hogeweg P (1997) Evolutionary consequences of coevolving targets. Evol Comput 5(4):401–418

Perkis T (1994) Stack-based genetic programming. In: Proceedings of the 1994 IEEE world congress on computational intelligence, Orlando, vol 1. IEEE, pp 148–153

Poli R, Langdon WB (1998) On the search properties of different crossover operators in genetic programming. In: Genetic programming 1998: proceedings of the third annual conference, University of Wisconsin, Madison. Morgan Kaufmann, pp 293–301

Poli R, Page J (2000) Solving high-order Boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. Genet Program Evolvable Mach 1(1/2):37–56

Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza)

Schoenauer M, Sebag M, Jouve F, Lamy B, Maitournam H (1996) Evolutionary identification of macro-mechanical models. In: Angeline PJ, Kinnear KE Jr (eds) Advances in genetic programming 2. MIT, Cambridge, chap 23, pp 467–488

Semenkin E, Semenkina M (2012) Self-configuring genetic programming algorithm with modified uniform crossover. In: Proceedings of the 2012 IEEE congress on evolutionary computation, Brisbane, pp 2501–2506

Silva S, Vanneschi L (2009) Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In: GECCO '09: proceedings of the 11th annual conference on genetic and evolutionary computation, Montreal. ACM, pp 1115–1122

Silva S, Vanneschi L (2010) State-of-the-art genetic programming for predicting human oral bioavailability of drugs. In: Advances in bioinformatics. Springer, 74:165–173. doi:10.1007/978-3-642-13214-8. http://dblp.uni-trier.de/db/conf/isami/iwpacbb2010.html#SilvaV10

Spector L (2001) Autoconstructive evolution: Push, PushGP, and pushpop. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2001), San Francisco. Morgan Kaufmann, pp 137–146

Spector L (2012) Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In: 1st workshop on understanding problems (GECCO-UP), Philadelphia. ACM, pp 401–408

Spector L, Robinson A (2002) Genetic programming and autoconstructive evolution with the Push programming language. Genet Program Evolvable Mach 3(1):7–40

Spector L, Klein J, Keijzer M (2005) The push3 execution stack and the evolution of control. In: GECCO 2005: proceedings of the 2005 conference on genetic and evolutionary computation, Washington, vol 2. ACM, pp 1689–1696

Van Belle T, Ackley DH (2002) Uniform subtree mutation. In: Foster JA, Lutton E, Miller J, Ryan C, Tettamanzi AGB (eds) Genetic programming, proceedings of the 5th European conference, EuroGP 2002, Kinsale. Lecture notes in computer science, vol 2278. Springer, pp 152–161

White DR, McDermott J, Castelli M, Manzoni L, Goldman BW, Kronberger G, Jaskowski W, O'Reilly UM, Luke S (2013) Better GP benchmarks: community survey results and proposals. Genet Program Evolvable Mach 14(1):3–29

Chapter 9
A Deterministic and Symbolic Regression Hybrid Applied to Resting-State fMRI Data

Ilknur Icke, Nicholas A. Allgaier, Christopher M. Danforth, Robert A. Whelan, Hugh P. Garavan, Joshua C. Bongard, and IMAGEN Consortium

Abstract Symbolic regression (SR) is one the most popular applications of genetic programming (GP) and an attractive alternative to the standard deterministic regression approaches due to its flexibility in generating free-form mathematical models from observed data without any domain knowledge. However, GP suffers from various issues hindering the applicability of the technique to real-life problems. In this paper, we show that a hybrid deterministic regression (DR)/genetic programming based symbolic regression (GP-SR) algorithm outperforms GP-SR alone on a brain imaging dataset.

Keywords Symbolic regression • Hybrid algorithm • Regularization • Resting-state fMRI

I. Icke (✉) • J.C. Bongard
Department of Computer Science, University of Vermont & Vermont Complex Systems Center, Burlington, VT, USA
e-mail: Ilknur.Icke@uvm.edu; Josh.Bongard@uvm.edu

N.A. Allgaier • C.M. Danforth
Department of Mathematics and Statistics, University of Vermont & Vermont Complex Systems Center, Burlington, VT, USA
e-mail: Nicholas.Allgaier@uvm.edu; Chris.Danforth@uvm.edu

R.A. Whelan • H.P. Garavan
Department of Psychiatry, University of Vermont, Burlington, VT, USA
e-mail: Robert.Whelan@uvm.edu; Hugh.Garavan@uvm.edu

IMAGEN Consortium
http://www.imagen-europe.com/en/consortium.php

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_9,

1 Introduction

Despite various success stories (Koza 2010; Dubcakova 2011) and claims that they will ultimately *replace scientists* (ACM 2011), GP-SR applications (or any evolutionary computation based approach in general) have not yet been widely accepted as standard tools for data science. The resistance is not without a cause: GP suffers from various issues that hinder its applicability to many real-world machine learning problems (O'Neill et al. 2010). The theoretical foundations of GP are not as well developed as many of the standard machine learning (ML) algorithms. Moreover, scalability is a very challenging problem due to the trial-and-error nature of GP. Some efforts to make GP more scalable via GPUs and cloud computing have been reported (such as in (GPG 2013) and (Sherry et al. 2011)). It is our belief that, if GP-SR is to be a trustable data science tool, it needs to take advantage of the recent developments in ML as well as parallel and distributed computing techniques.

The idea of studying evolutionary computation techniques from a ML perspective has previously been explored. The behavior of GP has been studied in terms of learning theory in (Amil et al. 2009) and (Castelli et al. 2011) among others. The learnable evolution model (LEM) proposed in (Michalski 2000) is a technique to guide evolutionary processes with ML by creating hypotheses characterizing the differences between high performing and low performing individuals in the population.

Recently, it has been suggested that GP might not be the best option for SR and that stochasticity is not necessarily a virtue (McConaghy 2011). The author proposed a deterministic basis function expansion method used in conjunction with a state-of-the-art deterministic ML regression algorithm as an alternative to GP-SR. This algorithm, Fast Function Extraction (FFX), was reported to outperform GP-SR on a number of real-world regression problems with dimensionalities ranging from 13 to 1,468. This paper shares the same basic ideology: SR should not stray from the well-established techniques of ML. However, we argue that abandoning GP altogether may be premature. Instead, we propose to hybridize the two approaches.

Our long term hypothesis is that a hybrid algorithm combining symbolic regression with state-of-the-art deterministic regression out-competes both symbolic regression alone and deterministic regression alone. Our short-term hypothesis, which is covered in this paper, suggests that hybridizing symbolic regression with state-of-the-art deterministic regression out-competes symbolic regression (GP-SR) alone. In this paper, we explore one way to incorporate a deterministic ML method into GP-SR in order to improve GP-SR and demonstrate the utility of this hybrid algorithm on a brain imaging dataset.

The functional magnetic resonance imaging (fMRI) is a non-invasive way of monitoring the activation of various brain regions while the subject lays in the MRI scanner. In the case of resting-state fMRI, the subject is not given any external stimuli. It is possible to model the activation in one brain region in terms of activations of other brain regions using a regression algorithm. By identifying the most predictive regions for each target region, one can build a functional brain

connectivity network (Poldrack and Nichols 2011). For successful identification of the brain connectivity network, the performance of the regression algorithm is very important, especially when a large number of brain regions are to be modeled simultaneously. In this paper, we focus on the first step, namely the regression algorithm that identifies the concurrent relationships between the brain regions. The subsequent steps of building functional connectivity networks from the fMRI resting-state data will not be discussed here.

The organization of this paper is as follows: Sects. 2–4 discuss the problem domain, technical background and related work respectively. Our proposed algorithm to hybridize the GP-SR and deterministic ML approaches is detailed in Sect. 5. Experimental results are presented and discussed in Sect. 6. Finally, Sect. 7 discusses our conclusions and future work.

2 Resting-State fMRI

Direct measurement of electrical neuron activity is not easily accomplished. Electroencephalograms (EEGs) can be recorded by placing electrodes on the surface of subject's head, and are adequate for measuring an averaged whole-brain electrical signal. However, targeting specific regions of the brain, especially at the resolution of a neuron, is not practical with current technology. This difficulty in direct measurement is the motivation for neuroimaging.

One of the most recent methods of neuroimaging, developed about 20 years ago, is functional MRI. Time series of 3-D images of the brain captured by magnetic field indicate changes in blood flow, called the *hemodynamic response*, that correlate with neuronal activity. The reason for this correlation is that active neurons require more glucose and oxygen than inactive ones, and thus neuronal activity is said to be blood-oxygen-level dependent, giving rise to the name BOLD for the fMRI signal. It is important to note that there is some danger in using the BOLD signal as a proxy for neuronal activity. Many other factors contribute to increased blood flow in the brain, including contraction/dilation of vasculature, the structure of that vasculature (which can vary considerably with demographic), and even subject-specific general health and behavioral habits. Some techniques for handling their conflation in the BOLD signal were introduced, however, the discussion of these techniques is beyond the scope of this overview.

Very high image resolutions are attainable with fMRI. For this work, the data collected fall on a grid with voxels (3D pixels) approximately 3 mm on a side, corresponding to about 50,000 voxels in a single whole-brain image. Typically, a region of interest (ROI) in the brain is represented by an averaged BOLD signal over some number of contiguous voxels, in order to smooth the noise present in a single voxel. Each of the 52 ROI's chosen for this study consists of a roughly spherical group of about 100 voxels, (with a radius of 3 voxels, or almost 1 cm). Their locations were chosen based on work from (Laird et al. 2011), in which statistical analysis across thousands of previous imaging studies (both stimulus/task-based and

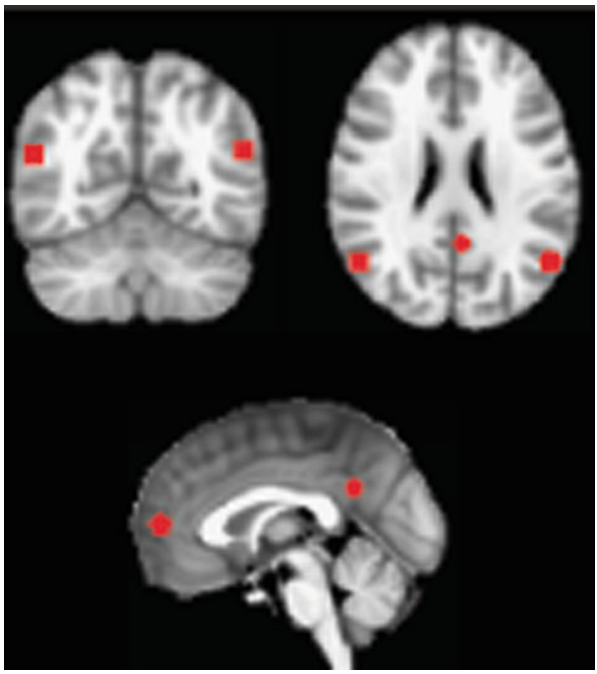


Fig. 9.1 Four selected ROI spheres for a resting-state network with centers corresponding to the highest *z*-statistics indicating the likelihood of co-activation across all studies (derived from independent component analysis) reported in (Laird et al. 2011)

resting-state) was used to identify networks of brain regions that tend to activate together. Figure 9.1 shows one such network, and spheres chosen to represent the ROI's in that network. Figure 9.2 shows a selection of equally spaced axial cross sections (top panel) that includes most of the 52 ROI's.

The data for the present study come from fMRI scans of a group of adolescents who were asked to keep their eyes closed while in the scanner, but were presented with no other task or stimulus. The term used for this type of data is "resting-state", though the brain is in no way at rest. In fact, the regions that activate during various task and stimulus-based studies also activate intermittently during the resting state, which suggests that some meaningful interactions are occurring and are ripe for analysis. Resting-state scans pose a challenge to standard fMRI analysis techniques, which typically involve regression of the BOLD signal in terms of a stimulus or task signal, (e.g. a square wave representing when the stimulus was being presented, or when the task was being performed). The application of deterministic regression, GP-SR, or a hybrid of the two, to model the activity in ROI's as functions *of one another* represents a potentially fruitful analytical strategy. Ideally, selection of ROI's would be avoided as well, and the input to the algorithm would be a much larger set of randomly selected regions within the brain. The very high dimensionality in this case provides an impetus for the hybrid technique, given the difficulty in applying GP-SR alone to such problems.

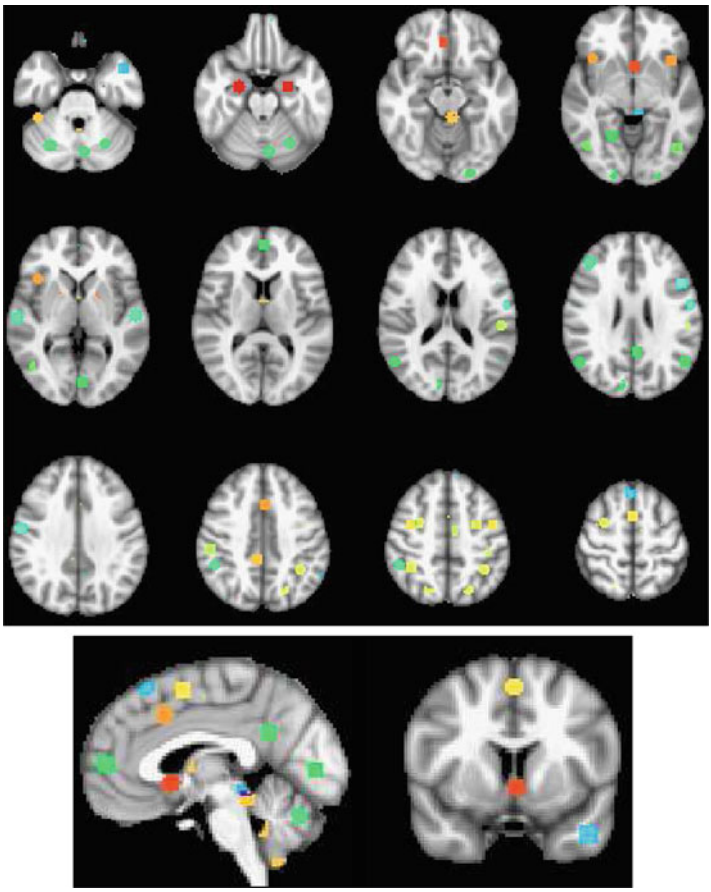


Fig. 9.2 (*Top panel*) Axial cross sections showing the majority of the 52 ROI's, colored by network, where the *top left* image is from a slice near the *bottom* of the brain, and the *bottom right* is near the *top* of the brain. (*Bottom panel*) Shown are a central sagittal cross section (*left*) and a coronal section from the front half of the brain (*right*)

3 Feature Extraction and the Regularization Approach

Data dimensionality poses a great challenge for both the numeric and symbolic regression algorithm. As the number of predictors increases, it becomes more difficult to identify the informative predictors and to build accurate models. This problem has been well-studied in ML. The task of seeking the best representation for a given dataset in order to optimize the performance of a ML algorithm is known as *feature extraction* (Guyon and Elisseeff 2003). Feature extraction can be seen as a sequential process, where new features are first constructed from the input variables and then the most informative ones are selected among the constructed features

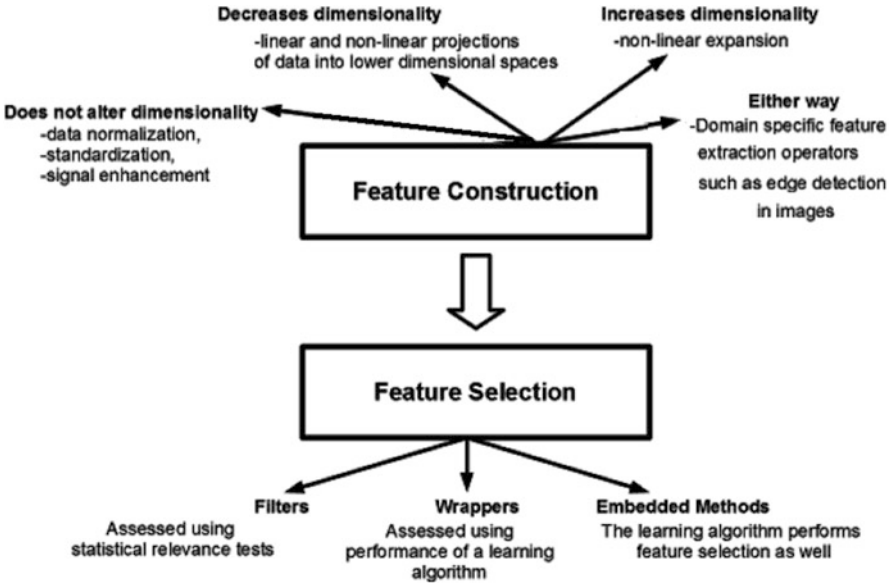


Fig. 9.3 Feature extraction as a sequential process of creating features from the input variables and then selecting the most informative features

(Fig. 9.3). *Feature construction* may or may not increase data dimensionality. If the input variables are suspected to have interactions, it is generally the practice to create additional features via non-linear expansion (such as $x_1 * x_2$).

As for *feature selection*, the simplest approach is to rank the features with respect to how well they correlate with the predicted variable. This method risks eliminating such features that might not be informative by themselves but may be very informative together. Subset selection methods aim to address this issue by considering a subset of features together. These techniques are divided into three main groups: filters, wrappers and embedded methods. *Filters* are pre-processing techniques that, independent of the learning algorithm, select a subset of variables with respect to some criteria such as mutual information. The *wrapper* techniques consider the learning algorithm as a black-box and select the set of features that optimize the performance of the learner. The feature subsets are generated either by forward selection, which gradually adds features or backward elimination, which starts with the whole set of features and eliminates least informative ones. The wrapper approach is computationally expensive as the learning algorithm needs to be executed many times. The *embedded* methods incorporate feature selection into the learning algorithm itself. The decision tree algorithms are the earliest examples of embedded methods. More recent embedded methods utilize the *regularization* technique, a version of which will be used in this work.

Within the context of the linear regression problem, regularization imposes additional constraints on the coefficients in order to reduce overfitting. In linear

regression, given a multivariate dataset $\mathbf{X}_{[M\times N]} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$, a matrix of observations, the response variable $\mathbf{Y}$ is defined as:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{N} \beta_j * \mathbf{x_j}$$

The coefficients are computed via the least squares estimation by minimizing the residual sum of squares:

$$RSS = min_\beta \left(\sum_{i=1}^{M} y_i - \beta_0 - \sum_{j=1}^{N} \beta_j * x_{ij} \right)^2$$

Since the parameters are computed on the training data, overfitting may occur. Overfitting manifests itself as extreme coefficient values. Therefore, an additional constraint on the coefficients is imposed in order to tame the coefficients ($\sum_{j=1}^{N} ||\beta_j||_1 \leq t$). This algorithm, known as lasso (least absolute shrinkage and selection operator), shrinks the coefficients and also performs feature elimination since the l_1-norm promotes sparsity. Therefore, the coefficients of uninformative features will be close to 0. An l_2-norm constraint is also possible and it is called ridge regression. Ridge regression has the effect of grouping the correlated variables so that they are included in the model together (Tibshirani 1996; Murphy 2012). The elastic net approach (Zou and Hastie 2005; Jerome Friedman and Tibshirani 2010) is a hybrid of lasso and ridge regression and is formulated as:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{N} \beta_j * \mathbf{x_j} + \lambda_2 ||\beta||_2^2 + \lambda_1 ||\beta||_1$$

Generally, λ_1, λ_2 are balanced by defining one single parameter ($0 \leq \alpha \leq 1$) that is called the mixing parameter. At the extreme values of α, the elastic net behaves like purely lasso or purely ridge regression. A very large value of λ forces all βs to be 0. As λ is relaxed, the coefficients start to take nonzero values. This sweep of λ values can be visualized as a regularization path (Fig. 9.4). The algorithm is named an elastic net since the "regularization path is like a stretchable net that retains all the big fish" (Zou and Hastie 2005).

This basic linear regression algorithm applies to generalized linear models (GLM) of the form:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{N} \beta_j * b_j(X)$$

where $b_j(X)$ are nonlinear basis functions applied to the input variables in order to construct new features.

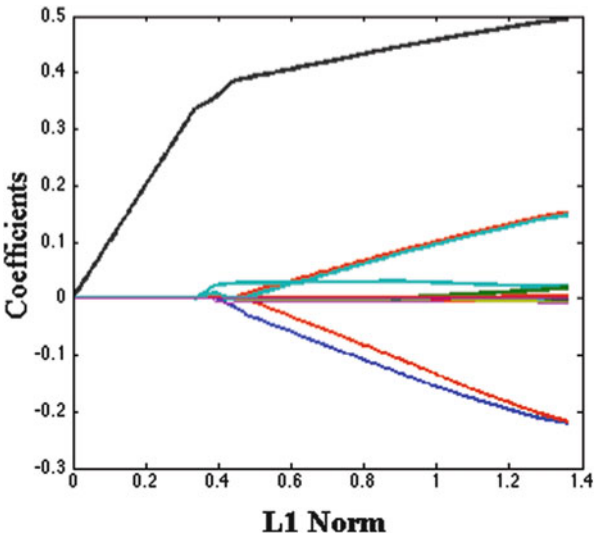


Fig. 9.4 Regularization path for an elastic net on a 10-dimensional dataset. For each λ, the l_1-norm of the coefficients vector versus individual coefficient values are shown. Each line traces the change of coefficient values for one variable. At the beginning, no features are selected. Gradually, more features are incorporated into the models (coefficients >0)

4 Related Work

GP-SR inherently performs feature selection when it finds sufficiently accurate models; any feature that does not appear in the evolved expression can be considered uninformative. However, when data dimensionality is high, the search space grows exponentially, making it difficult for GP-SR to find good solutions. The issue of feature selection in GP-SR algorithms has been studied by various researchers. Koza's automatically defined functions (ADF) (Koza 1992) can be seen as a feature extraction method within the context of SR. A pareto-GP based variable selection scheme was proposed in (Smits et al. 2005). In (McRee 2010), permuation tests were introduced in GP-SR to discriminate between informative and uninformative variables. In (Stijven et al. 2011), feature selection capabilities of GP-SR and random forests were compared. The authors report that when it finds an accurate model, GP-SR captures the important features that are missed by the random forests algorithm.

The regularization approach has been applied to GP-SR in (Nikolaev and Iba 2001) for polynomial functional form discovery. The authors incorporate a function smoothness term into the fitness function as a way to decrease overfitting.

The Fast Function Extraction (FFX) algorithm reported in (McConaghy 2011) employs a nonlinear basis function expansion method that creates new features via unary and binary interactions of the input variables. The algorithm does not employ GP-SR to construct the features or the models. The new features are created in a deterministic manner and passed to the elastic net algorithm for model building. The algorithm generates multiple models for the λs on the regularization path. The non-dominated set of these models with respect to accuracy versus complexity are identified as the final models.

The difference of our proposed technique is that we perform feature extraction using an efficient deterministic ML algorithm such as in FFX, and pass the features to GP-SR for model building. By taking advantage of the state-of-the-art deterministic regression methods, our algorithm aims to ease the burden of feature extraction on GP-SR and help it excel in model building, especially when higher order interactions between the variables exist. We have shown on a synthetic data suite with up to fourth order variable interactions that our technique significantly improves the performance of GP-SR as the data dimensionality increases (Icke and Bongard 2013).

5 Improving GP-SR Using Deterministic Machine Learning

The technique we propose in this paper has been largely inspired by the FFX algorithm reported in (McConaghy 2011). The author mainly suggested eliminating GP for the symbolic regression problem in favor of a deterministic method for augmenting the dataset with polynomial features and then using a state-of-the-art machine learning algorithm (elastic net) for model building. In this paper, we propose to hybridize GP with the deterministic ML techniques so as to take advantage of the strengths of both approaches to solve symbolic regression problems more accurately and efficiently in comparison to either technique alone. The outline of the general idea behind FFX is presented in Algorithm 1 (for a detailed description of the FFX algorithm, see (McConaghy 2011)).

Algorithm 1: Our variant of the basic FFX algorithm

Input: V={v_1,v_2, ..., v_N}
Output: The set of non-dominated evolved models based on validation data error-model complexity (number of bases) trade-off
[bases, expandedTrainingDataset] = *basisFunctionExtraction*(trainingDataset)
models={}
foreach $\alpha \in (0, 0.05, 0.1, \ldots, 1)$ **do**
    models = models $\bigcup$ *glmnetfit*(variables,trainingDataset)
    nonDominatedModels = *extractParetoFrontier*(models,expandedValidationDataset)
    models=nonDominatedModels
    models = models $\bigcup$ *glmnetfit*(bases,expandedTrainingDataset)
    nonDominatedModels = *extractParetoFrontier*(models,expandedValidationDataset)
    models=nonDominatedModels
end

The algorithm consists of three stages: feature construction, model building and model selection. The feature construction stage creates new features by applying binary nonlinear interactions (basis functions) and augmenting the original dataset

(Algorithm 2). It is clear that, with only the unary and binary features, the FFX algorithm will not accurately model data generated by a system with third or higher order interactions. Instead, it will find a quadratic approximation based on the unary and binary features which might increase the complexity of the expression in terms of the number of bases. It is possible to expand FFX beyond binary interactions; however, this would increase the number of constructed features exponentially. In this paper, we considered only unary and binary features as in (McConaghy 2011).

The model building stage utilizes the coordinate descent elastic net algorithm (glmnet, line 4 of Algorithm 1) that was proposed in (Jerome Friedman and Tibshirani 2010). As it is shown in Fig. 9.4, for each value of λ, one can build an expression using the corresponding coefficients. Therefore, the model building stage returns multiple expressions containing different numbers of basis functions. Note that the models are built on the training data. At the model selection stage, the non-dominated set of models with respect to error on validation data versus expression complexity (the number of basis functions or bases for short) are identified.

Algorithm 2: *basisFunctionExtraction*: polynomial basis function generation as new features form the observed data

Input: V={v_1,v_2, ..., v_N}
Output: Expanded Dataset: V_e={v_{e1},v_{e2}, ..., v_{eM} }
//Generate unary bases
foreach v_1,v_2, ..., v_N **do**
  unaryBases = unaryBases $\bigcup$ v_i
  foreach exp_j **do**
    unaryBases = unaryBases $\bigcup$ $v_i^{exp_j}$
  end
  foreach $unaryOperator_k$ **do**
    unaryBases = unaryBases $\bigcup$ $\text{unaryOperator}_k(v_i)$
  end
end
//Generate binary bases
foreach $u_i \in unaryBases$ **do**
  foreach $u_j \in unaryBases$ **do**
    binaryBases= binaryBases $\bigcup$ $u_i * u_j$
  end
end

Our hybrid approach takes advantage of the basic FFX idea in order to identify the most important building blocks that might help GP-SR build accurate models. The method is outlined in Algorithm 3.

After one run of the FFX algorithm, the most useful basis functions are extracted from the set of models by computing a histogram of how frequently each basis function uniquely appears in the expressions (Fig. 9.5). The number of basis functions to be selected can be determined in a number of ways. In this paper, we chose the top N basis functions, where N is the number of original input variables. This choice was made in order to provide both the GP-SR and the hybrid

Algorithm 3: The hybrid FFX/GP-SR algorithm

Input: V={v_1,v_2, ..., v_N}
Output: One best model with respect to the validation data error and complexity
nonDominatedModels = *ffx*(trainingDataset)
bases = *identifyUsefulBasisFunctions* (nonDominatedModels, validationDataset)
newDataset=*createNewDataset*(bases)
bestModel = GP-SR(newDataset)

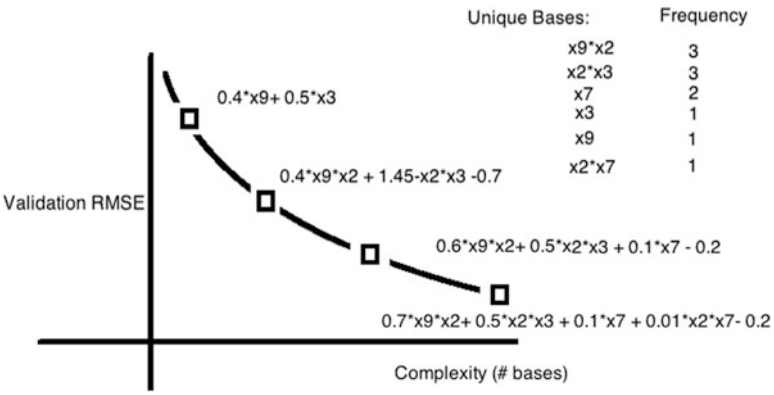


Fig. 9.5 Determination of the most useful features using validation data based on the models built by the elastic net on training data

FFX/GP-SR algorithms the same amount of data in terms of rows and columns. Before the hybrid algorithm runs, a new dataset with at most N features are computed using the selected basis functions and GP-SR is run on this new dataset instead of the original dataset.

6 Experimental Results

In this section, we first outline the procedure we followed in preparing the datasets and the parameters for each algorithm. Then, we compare the performance of our proposed algorithm to the plain GP-SR implementation on the resting-state fMRI data from a number of subjects. For each subject, the dataset contains 187 observations (rows) and 52 brain regions (columns). The data points for each brain region were divided into three non-overlapping and interleaving partitions for training (113 observations), validation (37 observations) and testing (37 observations) purposes (Fig. 9.6). The hybrid FFX/GP-SR algorithm performs the initial modeling on the training partition, and the best models are selected based on validation set performance. Then, the set of most useful basis functions are selected as described in the previous section. The test partitions are never used in model generation or

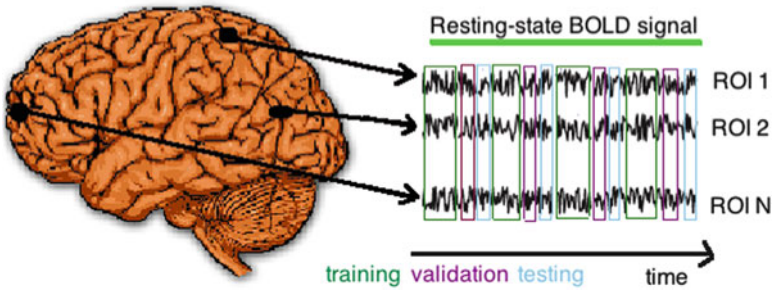


Fig. 9.6 Data partitioning: for each region of interest (ROI), the data points are partitioned into non-overlapping sets of training, validation and test points. Every fifth point is in the test partition and every fourth point is in the validation partition

Table 9.1 Default FFX-variant parameters

Parameter	Value
Basis function expansion	Exponents: 1
	Interactions: unary, binary
	Operators: { }
Elastic net	$\alpha : \{0, 0.05, 0.1, \ldots, 1\}$
	λ: 100 λ values calculated by glmfit based on α
	Maximum basis functions allowed: 250
Model selection	Non-dominated models with respect to validation data error versus number of bases

selection. Finally, performances of the GP-SR and FFX/GP-SR hybrid are reported based on the errors on the test partitions.

We implemented our GP-based Symbolic Regression application using the GPTIPS Matlab package downloaded from (Searson 2013). Our version of the FFX algorithm and FFX/GP-SR algorithms were also implemented in Matlab using the glmnet package downloaded from (Jiang 2013) and the GPTIPS package. All experiments were run on a cluster computing environment. Unless otherwise specified, all FFX and GP-SR runs were performed using the default parameters specified in Tables 9.1 and 9.2.

For each subject, we ran the FFX algorithm for each region (column) as the target variable and all other columns as the predictors. For the GP-SR algorithm, we ran the GP-SR 30 times for each region as the target variable. Each GP-SR run was given a run-time budget of 5 min. For the FFX/GP-SR hybrid, we ran the FFX algorithm once for each region, extracted basis functions to recreate the dataset and ran the GP-SR 30 times for each region for a run-time budget of 5 min.

The comparisons between GP-SR and our hybrid FFX/GP-SR methods on subject 1 are presented in Fig. 9.7. For each region, we compare the test set errors of the best models found by the 30 runs of each algorithm. The winning algorithm for each region is shown on top of each plot in parentheses. The winner is determined

Table 9.2 Default GP-SR parameters

Parameter	Value
Representation	GPTIPS (Searson 2013) multigene syntax tree Number of genes: 1 Maximum tree depth: initially 3, 12 afterwards
Population size	2,000
Runtime budget	5 min
Selection	Lexicographic tournament selection
Tournament size	7
Crossover operator	Subtree crossover
Crossover probability	0.85
Mutation operator	Subtree mutation
Mutation probability	0.1
Reproduction probability	0.05
Building blocks	Operators: $\{+, -, *, protected/\}$ Terminal symbols: $\{x_1, \ldots, x_N\}$
Fitness	RMSE: $\sqrt{\frac{1}{N}\sum(y-\hat{y})^2}$
Elitism	Keep 1 best individual

based on a Wilcoxon rank sum left-tailed test, $\alpha = 0.05$. In the cases where no winner is specified, the algorithms were not found to be statistically different in terms of test set performance.

We repeated the experiments for a total of seven subjects. The summary of the results are provided in Table 9.3. Based on 364 distinct regression tasks, the results clearly indicate that the hybrid algorithm has a significantly better chance than random coin flipping in performing at least as well as the GP-SR algorithm. Moreover, the hybrid algorithm is around two times or more likely to outperform the plain GP-SR than to be outperformed by it.

We also compared all three approaches in order to see if it would just suffice to utilize FFX and not resort to the GP-SR techniques at all. Figures 9.8–9.10 show the non-dominated sets of models generated by all three algorithms combined into one single Pareto front. For each brain region, we apply the final models from each algorithm to the testing partition and record error versus expression complexity (number of nodes) and then compute the non-dominated set. For the GP-SR and hybrid methods, there are 30 final models each, while for the FFX-variant, the number of final models varies. Each plot shows the non-dominated models marked with respect to the algorithms that generated them. As it is evident, the FFX-variant does not dominate the GP-SR based approaches. Specifically, for some brain regions, the solutions found by the FFX-variant are significantly more complex for a relatively small decrease in error. Therefore, we are convinced that we should not completely give up on GP based approaches for symbolic regression.

Finally, we inspected those brain regions for which GP-SR outperformed the hybrid approach. For instance, for subject 1, region 9 (Fig. 9.7), we identified that one region that was found to be predictive of region 9 by the GP-SR algorithm was totally omitted from the models built by the FFX-variant. Therefore, it was

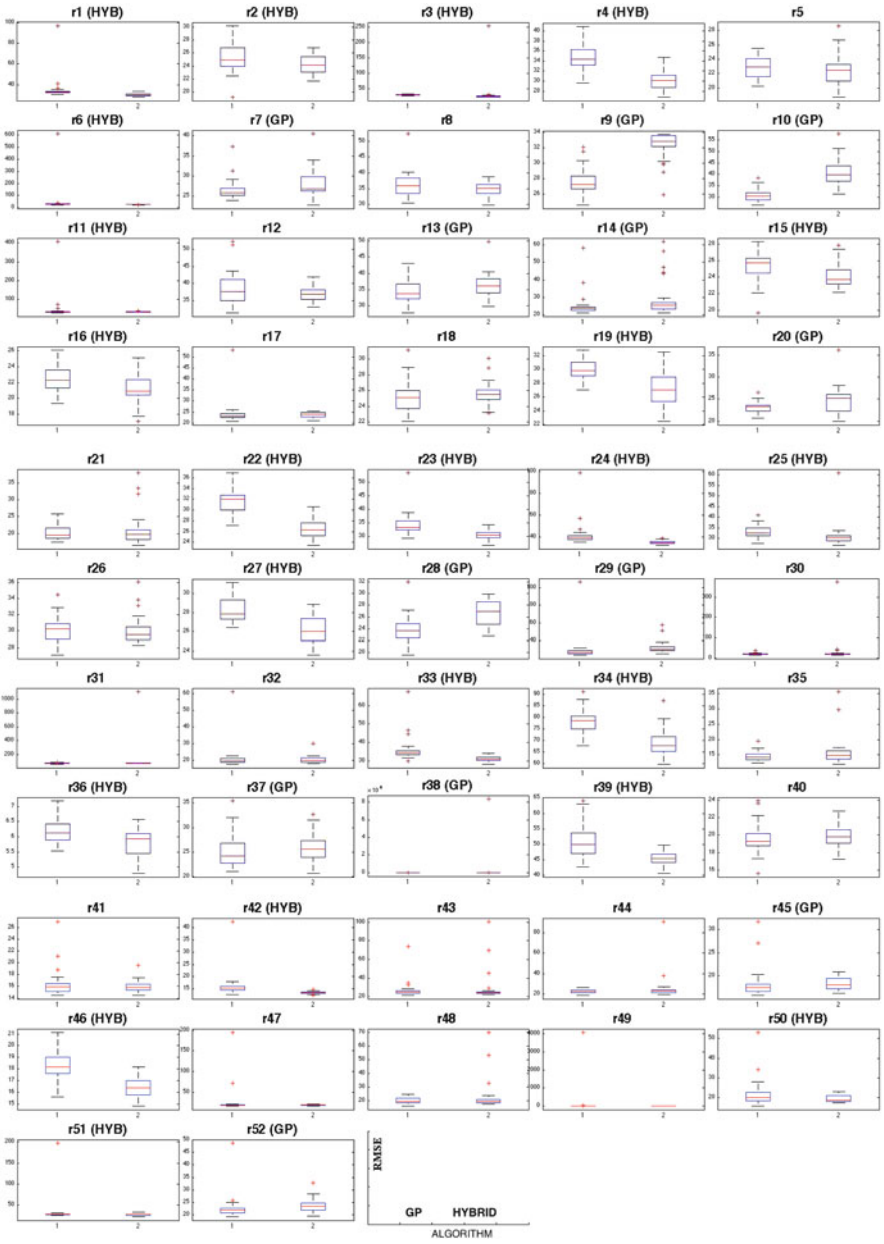


Fig. 9.7 Comparison of the hybrid and GP-SR on subject 1 based on testing partition RMSE. Hybrid wins 22 out of the 52 regions (42.3 %). GP wins 12/52 (23 %), and no performance difference in 18/52 (34.7 %) regions. At the expense of an additional 90 s for the one-time run of the FFX-variant, the hybrid algorithm has a 77 % chance of performing at least as good as the GP-SR

Table 9.3 Summary of results on a number of subjects (over all 52 regions per subject). The probability of the hybrid algorithm not being worse than the GP-SR is significantly higher than random chance (approximately 82%) over $52 * 7 = 364$ regression runs

Subject	%Hybrid wins	%GP-SR wins	%No difference	%Hybrid at least as good as GP-SR (100-GP-SR wins)
1	42.3 %	23%	34.7%	77%
2	61.5%	15.4%	23.1%	84.6%
3	46.1%	23.1%	30. 8%	76.9%
4	32.7%	17.3%	50%	82.7%
5	36.6%	19.2%	44.2%	80.8%
6	44.2%	9.6%	46.2%	90.4%
7	44.2%	19.2%	36.6%	80.8%
Overall	44%	18.1%	37.9%	81.9%

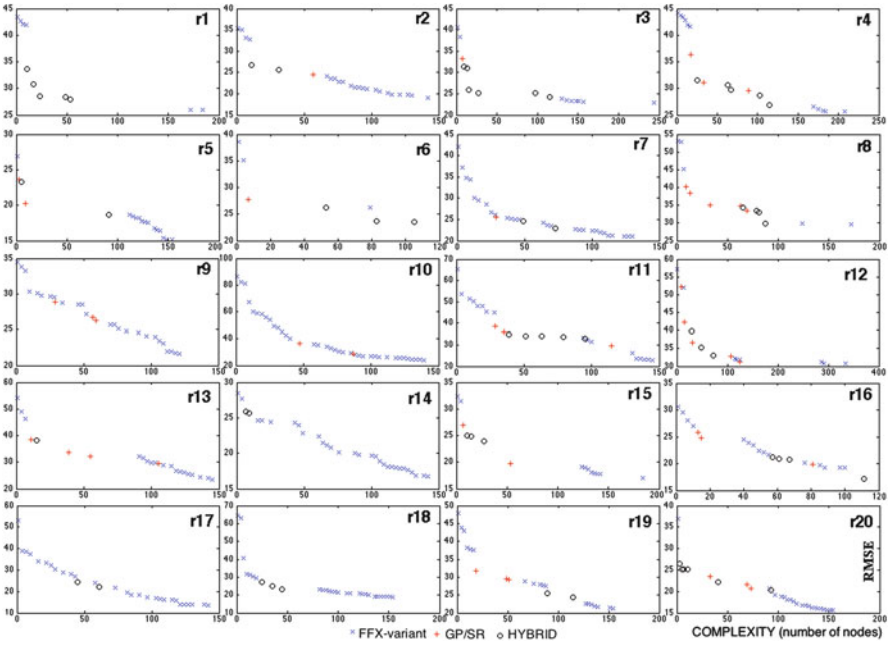


Fig. 9.8 Comparison of all 3 approaches on subject 1 for regions 1–20. The plots show combined non-dominated models with respect to complexity versus error on testing data

not possible for the hybrid algorithm to incorporate that predictive variable into the models since the input to the hybrid algorithm is determined by the outcome of the FFX-variant.

In summary, our experimental results show that hybridizing GP-SR with a state-of-the-art deterministic ML technique is significantly more likely to improve the performance of GP-SR. We also show evidence that the hybrid approach works better than the deterministic ML technique on a considerable number of cases strengthening our claim that abandoning GP altogether for SR is premature.

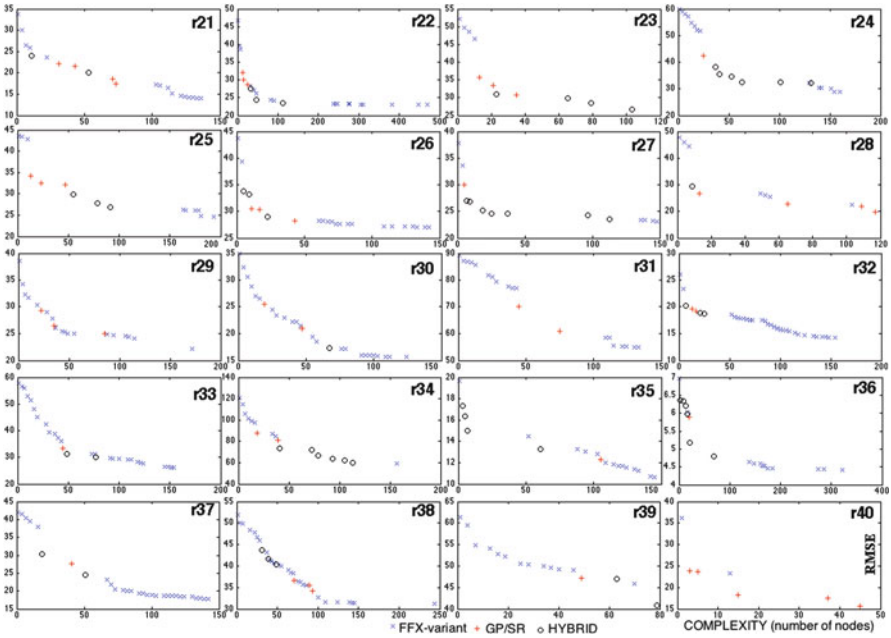


Fig. 9.9 Comparison of all 3 approaches on subject 1 for regions 21–40. The plots show combined non-dominated models with respect to complexity versus error on testing data

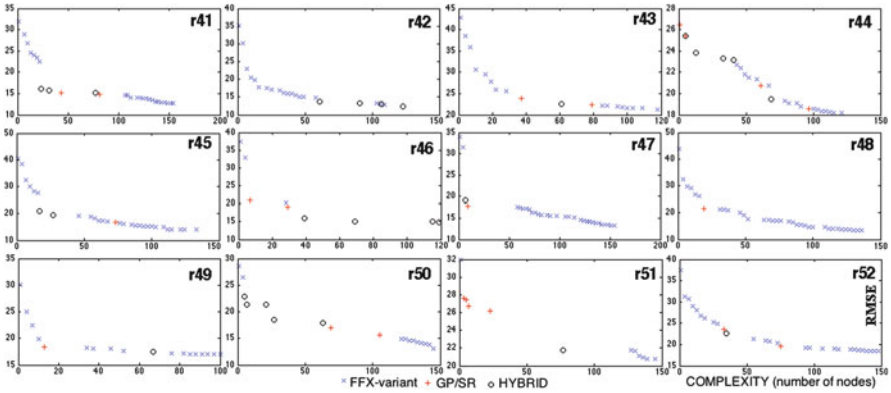


Fig. 9.10 Comparison of all 3 approaches on subject 1 for regions 41–52. The plots show combined non-dominated models with respect to complexity versus error on testing data

7 Conclusions

Despite all the various efforts to improve it, GP-SR is yet to be accepted as a standard data analysis tool. In this paper, we argued that the resistance from the ML community was not totally unfounded. First of all, theoretical underpinnings of the GP-SR such as convergence proofs are not well established. GP-SR is computationally more expensive compared to most deterministic ML algorithms due to its stochastic nature. Even though many intuitive strategies might be built into the algorithm, there is no guarantee that optimal data models will emerge at the end of the run. More importantly, despite all the success stories, GP-SR techniques do not necessarily outperform the state-of-the-art deterministic ML methods, especially on high dimensional problems. On the other hand, the strength of GP-SR is in its model-free nature which enables the algorithm to discover more accurate, more interpretable and novel models in comparison to a deterministic approach. In summary, it is our belief that stochasticity can be a virtue for SR if it is directed intelligently. To this end, we showed here that it was possible to create synergy between the deterministic and GP-SR approaches by hybridizing them. The technique presented in this paper is just one way out of many possible options to combine the GP-SR and deterministic techniques for regression problems. Here, we incorporated building blocks extracted by the deterministic algorithm into the GP-SR algorithm. Application of the algorithm to a real-world dataset from the neuroscience field gave encouraging results. Our current work focuses on improving this hybrid algorithm and validating it on other datasets. We are also working on devising a more rigorous scheme to compare all three approaches (the FFX-variant, GP-SR and the hybrid) and investigate the situations where each method would succeed or fail.

Acknowledgements This research was partially supported by a DARPA grant #FA8650-11-1-7155. We thank the IMAGEN consortium for providing us with the resting-state fMRI dataset. The authors also acknowledge the Vermont Advanced Computing Core which is supported by NASA (NNX 06AC88G), at the University of Vermont for providing High Performance Computing resources that have contributed to the research results reported within this paper.

References

ACM (2011) Robot biologist solves complex problem from scratch. http://cacm.acm.org/careers/136345-robot-biologist-solves-complex-problem-from-scratch/fulltext

Amil NM, Bredeche N, Gagné C, Gelly S, Schoenauer M, Teytaud O (2009) A statistical learning perspective of genetic programming. In: Vanneschi L, Gustafson S, Moraglio A, De Falco I, Ebner M (eds) Proceedings of the 12th European conference on genetic programming, EuroGP 2009, Tuebingen. Lecture notes in computer science, vol 5481. Springer, pp 327–338. doi:10.1007/978-3-642-01181-8-28

Castelli M, Manzoni L, Silva S, Vanneschi L (2011) A quantitative study of learning and generalization in genetic programming. In: Silva S, Foster J, Nicolau M, Machado P, Giacobini M (eds) Genetic programming, Torino. Lecture notes in computer science, vol 6621. Springer, Berlin/Heidelberg, pp 25–36

Dubcakova R (2011) Eureqa: software review. Genet Program Evolvable Mach 12(2):173–178. doi:10.1007/s10710-010-9124-z

GPG (2013) Genetic programming on general purpose graphics processing units. http://www.gpgpgpu.com/

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. J Mach Learn Res 3:1157–1182

Icke I, Bongard J (2013) Improving genetic programming based symbolic regression using machine learning. In: IEEE congress on evolutionary computation, CEC 2013, Cancun

Jerome Friedman TH, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. J Stat Softw 33(1):1–22

Jiang H (2013) Glmnet for Matlab. http://www-stat.stanford.edu/~tibs/glmnet-matlab/

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Koza JR (2010) Human-competitive results produced by genetic programming. Genet Program Evolvable Mach 11(3/4):251–284. doi:10.1007/s10710-010-9112-3, tenth anniversary issue: progress in genetic programming and evolvable machines

Laird AR, Fox PM, Eickhoff SB, Turner JA, Ray KL, McKay DR, Glahn DC, Beckmann CF, Smith SM, Fox PT (2011) Behavioral interpretations of intrinsic connectivity networks. J Cogn Neurosci 23(12):4022–4037

McConaghy T (2011) FFX: fast, scalable, deterministic symbolic regression technology. In: Riolo R, Vladislavleva E, Moore JH (eds) Genetic programming theory and practice IX. Genetic and evolutionary computation. Springer, Ann Arbor, chap 13, pp 235–260. doi:10.1007/978-1-4614-1770-5-13

McRee RK (2010) Symbolic regression using nearest neighbor indexing. In: Gustafson S, Kotanchek M (eds) GECCO 2010 symbolic regression workshop, Portland. ACM, pp 1983–1990. doi:10.1145/1830761.1830841

Michalski RS (2000) Learnable evolution model: evolutionary processes guided by machine learning. Mach Learn 38:9–40

Murphy KP (2012) Machine learning a probabilistic perspective. MIT Press, Cambridge, MA

Nikolaev NY, Iba H (2001) Regularization approach to inductive genetic programming. IEEE Trans Evol Comput 54(4):359–375. http://ieeexplore.ieee.org/iel5/4235/20398/00942530.pdf?isNumber=20398

O'Neill M, Vanneschi L, Gustafson S, Banzhaf W (2010) Open issues in genetic programming. Genet Program Evolvable Mach 11(3–4):339–363. doi:10.1007/s10710-010-9113-2, http://dx.doi.org/10.1007/s10710-010-9113-2

Poldrack RA, Nichols TE (2011) Handbook of functional MRI data analysis. Cambridge University Press, New York

Searson D (2013) GPTIPS for Matlab. https://sites.google.com/site/gptips4matlab/

Sherry D, Veeramachaneni K, McDermott J, O'Reilly UM (2011) Flex-GP: genetic programming on the cloud. In: Di Chio C, Agapitos A, Cagnoni S, Cotta C, Fernandez de Vega F, Di Caro GA, Drechsler R, Ekart A, Esparcia-Alcazar AI, Farooq M, Langdon WB, Merelo JJ, Preuss M, Richter H, Silva S, Simoes A, Squillero G, Tarantino E, Tettamanzi AGB, Togelius J, Urquhart N, Uyar AS, Yannakakis GN (eds) Applications of evolutionary computing, evoapplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, EvoSTOC, Malaga. Lecture notes in computer science, vol 7248. Springer, pp 477–486. doi:10.1007/978-3-642-29178-4-48

Smits G, Kordon A, Vladislavleva K, Jordaan E, Kotanchek M (2005) Variable selection in industrial datasets using Pareto genetic programming. In: Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice III. Genetic programming, vol 9. Springer, Ann Arbor, chap 6, pp 79–92

Stijven S, Minnebo W, Vladislavleva K (2011) Separating the wheat from the chaff: on feature selection and feature importance in regression random forests and symbolic regression. In: Gustafson S, Vladislavleva E (eds) 3rd symbolic regression and modeling workshop for GECCO 2011, Dublin. ACM, pp 623–630. doi:10.1145/2001858.2002059

Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc (Ser B) 58:267–288

Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. J R Stat Soc Ser B 67(2):301–320

Chapter 10
Gaining Deeper Insights in Symbolic Regression

Michael Affenzeller, Stephan M. Winkler, Gabriel Kronberger, Michael Kommenda, Bogdan Burlacu, and Stefan Wagner

Abstract A distinguishing feature of symbolic regression using genetic programming is its ability to identify complex nonlinear white-box models. This is especially relevant in practice where models are extensively scrutinized in order to gain knowledge about underlying processes. This potential is often diluted by the ambiguity and complexity of the models produced by genetic programming. In this contribution we discuss several analysis methods with the common goal to enable better insights in the symbolic regression process and to produce models that are more understandable and show better generalization. In order to gain more information about the process we monitor and analyze the progresses of population diversity, building block information, and even more general genealogy information. Regarding the analysis of results, several aspects such as model simplification, relevance of variables, node impacts, and variable network analysis are presented and discussed.

Keywords Symbolic regression • Algorithm analysis • Population diversity Building block analysis • Genealogy • Variable networks

1 Introduction

Among modern data-based modeling techniques, symbolic regression distinguishes itself by its ability to identify nonlinear white-box models without the need to specify the structure of the models. The basic principles of symbolic regression

M. Affenzeller (✉) • S.M. Winkler • G. Kronberger • M. Kommenda • B. Burlacu • S. Wagner
Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria
e-mail: michael.affenzeller@fh-hagenberg.at; stephan.winkler@fh-hagenberg.at; gabriel.kronberger@fh-hagenberg.at; michael.kommenda@fh-hagenberg.at; bogdan.burlacu@fh-hagenberg.at; stefan.wagner@fh-hagenberg.at

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_10,

as well as characteristic applications have been described previously (Koza 1992; Langdon and Poli 2002). However, symbolic regression using GP has some limitations which make it rather a gray-box technique (Kotanchek et al. 2013; Kronberger 2011). In particular, code bloat and occurrence of introns (non-functional code fragments) result in overly complex models (Banzhaf and Langdon 2002; Jackson 2010). Furthermore, the stochastic effects of genetic drift cause different search paths in the model space leading to diverse models with differing structural properties, utilized variables, model complexity, and generalization properties.

In this contribution we want to highlight and discuss a set of methods to, on the one hand, get deeper insights in the modeling process, and, on the other hand, improve the understandability and trust in symbolic regression models (Kotanchek et al. 2007). We do not aim to analyze the effectiveness of different methods or algorithmic adaptations; instead exemplary results are shown with the aim to highlight different observations of dynamical aspects like selection pressure, population diversity or building block evolution during the run.

We distinguish between pre-analysis and post-analysis methods. Pre-analysis methods are performed dynamically during the run of an algorithm and post-analysis methods focus on aspects of model simplification, variable impacts, and interaction analysis. The goal of these analysis techniques is to gain insights into algorithm dynamics and to subsequently improve algorithms. The focus of pre-analysis is algorithm design and parameterization. In contrast, post-analysis based on sets of models are mainly aimed to support the interpretation of models from the viewpoint of domain experts. For this purpose techniques for model simplification and measures for variable impacts are discussed. Further investigations concern network analysis where symbolic regression is extended to build a comprehensive representation of all dependencies between variables. This approach is also helpful to harmonize the often ambiguous modeling results for a single variable.

The remaining part of this chapter is organized as follows: Sect. 2 starts with a short summary of offspring selection (OS) as OS has been chosen as an exemplary algorithm variant used to demonstrate the analysis techniques discussed in the following parts when compared with standard genetic programming. The remainder of Sect. 2 summarizes algorithm and model analysis measures like population diversity, building block, and genealogy graph analysis. Section 3 summarizes measures for analyzing of the impact of variables and subtrees as well as variable networks formed by symbolic regression. Finally, Sect. 4 concludes with a summary.

2 Analysis of Algorithm Dynamics

In this section we discuss analysis methods concerning GP dynamics for gaining deeper insights in the evolution of populations. The comparison between standard GP and offspring selection GP (OSGP) was chosen to demonstrate the different algorithm analysis techniques. Therefore, a brief description of the OSGP algorithm is given in the following section.

Offspring Selection in Genetic Programming

The general idea of offspring selection (OS) is to apply selection after reproduction and to apply selection rather on the gene level than on the chromosome level. OS is integrated into the general workflow of an evolutionary algorithm by introducing an additional selection step in order to decide whether or not a candidate offspring is accepted as a member of the next population.

The use of this concept in genetic algorithms has been introduced as the single population formulation of the coarse-grained parallel SASEGASA algorithm (Affenzeller and Wagner 2004). Several variants of OS and their impact on the performance of evolutionary algorithms and genetic programming for various problems have already been discussed (Affenzeller et al. 2009). When applied to symbolic regression and classification, usually a simplified version of OS (here called strict OS) is applied which is closely related to soft brood selection and upward-mobility selection (Altenberg 1994).

In strict OS which is described in Algorithm 4 the OS criterion for accepting new offspring is formulated in the following way: A candidate offspring, which is generated by crossover and optional mutation, is accepted as a member of the next generation if and only if its fitness value surpasses the fitness of the better of the two parent solution candidates. A further aspect of strict OS is that all members of the next generation have to fulfill the claimed OS criterion. This means that all members of the next generation are reproduction results which were able to surpass the fitness of the better of their own parents.

Algorithm 4: Strict offspring selection

Input: $PopSize$, $MaxGenerations$, $MaxSelectionPressure$, fitness function F
  $CurrentSelectionPressure \leftarrow 0$, $g \leftarrow 0$
  $Population_0 \leftarrow$ Create $PopSize$ Individuals
  while $g < MaxGenerations \wedge CurrentSelectionPressure < MaxSelectionPressure$ **do**
    $Population_{g+1} \leftarrow \{\}$
    $CurrentSelectionPressure \leftarrow 0$
    while $|Population_{g+1}| < PopSize \wedge CurrentSelectionPressure < MaxSelectionPressure$ **do**
      $Parent_1, Parent_2 \leftarrow$ Select parent individuals from $Population_g$
      $Child \leftarrow$ Generate child individual from selected parents
      if $F(Child) > \max(F(Parent1), F(Parent2))$ **then**
        $Population_{g+1} \leftarrow Population_{g+1} \cup Child$
      end if
      $CurrentSelectionPressure \leftarrow CurrentSelectionPressure + \frac{1}{PopSize}$
    end while
    $g \leftarrow g + 1$
  end while

Within OS selection pressure can be defined quite easily by the ratio of candidate solutions that had to be generated in order to fill up the next generation with successful offspring and the population size. This more obvious formulation is

equivalent to the formulation of *CurrentSelectionPressure* as stated in Algorithm 4. As a consequence, the current selection pressure is adjusted self-adaptively with respect to the difficulty of producing successful offspring of the members of the current generation. The current value of selection pressure therefore gives additional hints about how easy or difficult it is to take usage of the current gene pool. Furthermore, this strategy allows the definition of a quite easy and dynamic termination criterion.

Population Diversity

Population diversity is essential for the progress of GP. In analogy to nature, diversity can be seen as the average difference between individuals, and can be measured on a structural (genotypic) or behavioral (phenotypic, or fitness-based) level. Considering the typical variable-length tree encoding employed in GP, it is clear that fitness equality does not imply genotypic equality, although two identical genotypes will lead to the same fitness.

A very comprehensive overview of program tree similarity and diversity measures has been given for instance in Burke et al. (2004). An example of phenotypic diversity is given as the entropy of a population of trees taking into account their fitness values (Rosca 1995). In contrast to phenotypic diversity, genotypic diversity is used to characterize the amount of unique, non-repeated genetic material contained by the individuals in the population (Keijzer 1996). A common approach to calculate the genetic diversity is to utilize a tree similarity or distance measure (Ekart and Nemeth 2000; Vanneschi et al. 2006). Furthermore, fine-grained population diversity estimations for GP as well as the comparison of diversity progresses of several GP variants have been analyzed (Affenzeller et al. 2009; Winkler 2009).

Maximum Common Subtree Similarity

We define the following tree similarity measure based on the maximum common subtree in Eq. 10.1. The maximum common subtree is computed using a tree node comparison function that considers the structure of trees and optionally node properties (i.e., variable weights, variables, constants). By taking two times the size of the maximum common subtree (since it is present in both trees) and dividing by the sum of the tree lengths, we ensure that $Sim(t_1, t_2) \in [0, 1]$.

$$Sim(t_1, t_2) = \frac{2 \cdot |MaximumCommonSubtree(t_1, t_2)|}{|t_1| + |t_2|} \tag{10.1}$$

The population diversity, based on the maximum common subtree similarity, is defined in Eq. 10.2, where N denotes the population size. Since the tree similarity

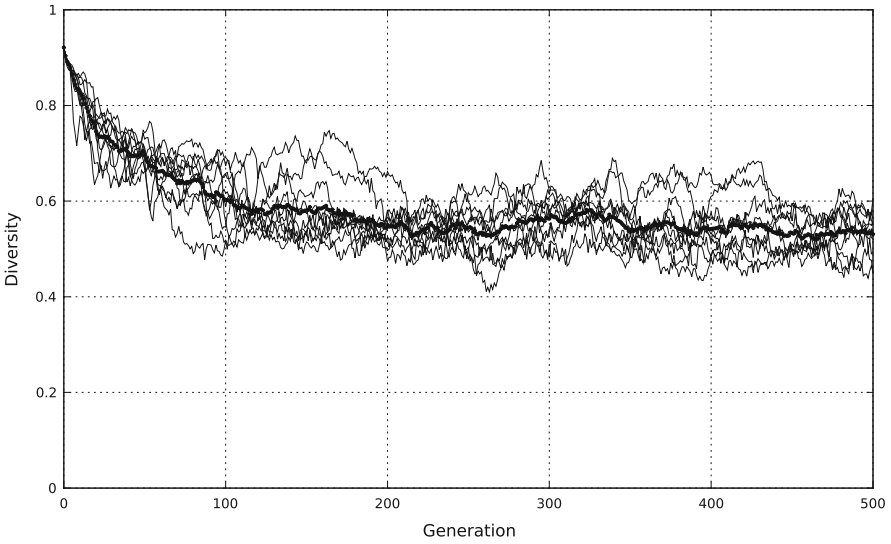


Fig. 10.1 Population diversity in standard GP

is symmetric, a total number of $N(N-1)/2$ similarity computations (excluding the similarity of an individual with itself) must be performed. After scaling by the number of computations the population diversity is in the interval $[0, 1]$, where a diversity of 0 indicates that all trees are exactly identical.

$$Div(T) = 1 - \frac{\sum_{i=1}^{N} \sum_{j=i+1}^{N} Sim(t_i, t_j)}{N(N-1)/2} \tag{10.2}$$

Exemplary Population Diversity Progresses

We illustrate the evolution of diversity for the standard GP and GP with strict OS (OSGP) on the Poly-10 symbolic regression benchmark problem (Poli 2003) given in Eq. 10.3:

$$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}. \tag{10.3}$$

For both algorithms the population size was set to 500 individuals, with different termination criterions; a generation limit of 500 for SGP, and a maximum selection pressure value of 100 for OSGP.

Figures 10.1 and 10.2 show the diversity curves for 10 independent SGP and OSGP runs, respectively; the highlighted curve in each figure represents the average diversity. We see that typically in standard GP the population diversity eventually fluctuates around 0.5–0.6, whereas in OSGP the population diversity significantly decreases towards the end of the evolutionary process and ends up between 0.25–0.3. The more pronounced diversity decrease in OSGP can be

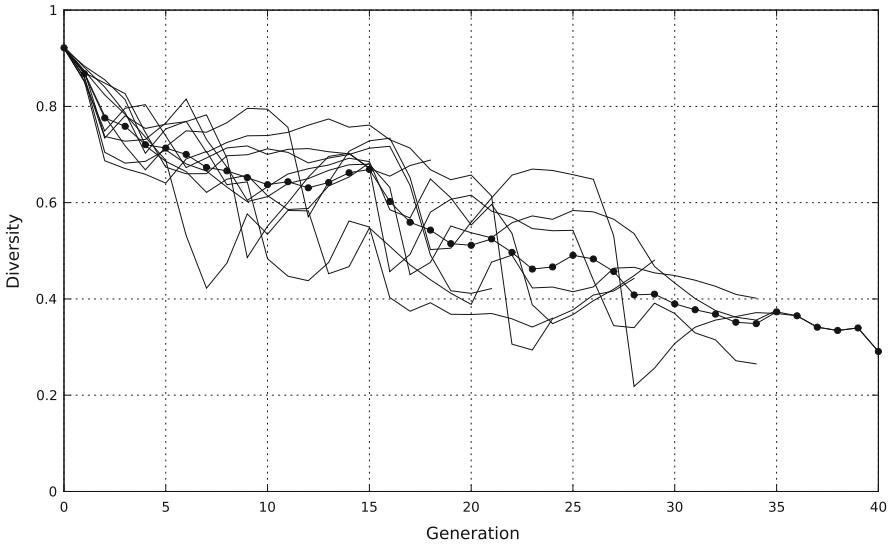


Fig. 10.2 Population diversity in offspring selection GP

explained by the adaptive adjustment of selection pressure; the relevant alleles are exploited from the beginning and propagated through the population much faster.

Figure 10.3 shows the evolution of selection pressure values for 10 GP runs with strict OS. As we see in this figure, the maximum selection pressure (100) is in some cases reached after 15–20 generations, whereas some runs are executed for more than 30 generations.

Occurrence Frequencies of Relevant Building Blocks

Building blocks are generally defined as parts of the solution that the GP algorithm has to find, for example certain terms for the formula of a symbolic regression problem (Langdon and Poli 2002). With the repeated action of crossover and selection, they are iteratively assembled into larger hierarchical structures until, ideally, a complete representation of the solution is found.

A common problem, associated with the premature convergence of the algorithm, is the disappearance of relevant solution elements from the population before they get the chance to be assembled into building blocks with a high fitness value. This may happen when relevant structures are overlooked by selection due to bad parameter values (constants, variable weights) causing them to have a low fitness.

We investigate this behavior on the *Poly-10* problem, considering the terms of the formula as the relevant building blocks: $(* \; x_1 \; x_2)$, $(* \; x_3 \; x_4)$, $(* \; x_5 \; x_6)$, $(* \; x_1 \; x_7 \; x_9)$ and $(* \; x_3 \; x_6 \; x_{10})$. Larger building blocks such as the last two terms in the *Poly-10*

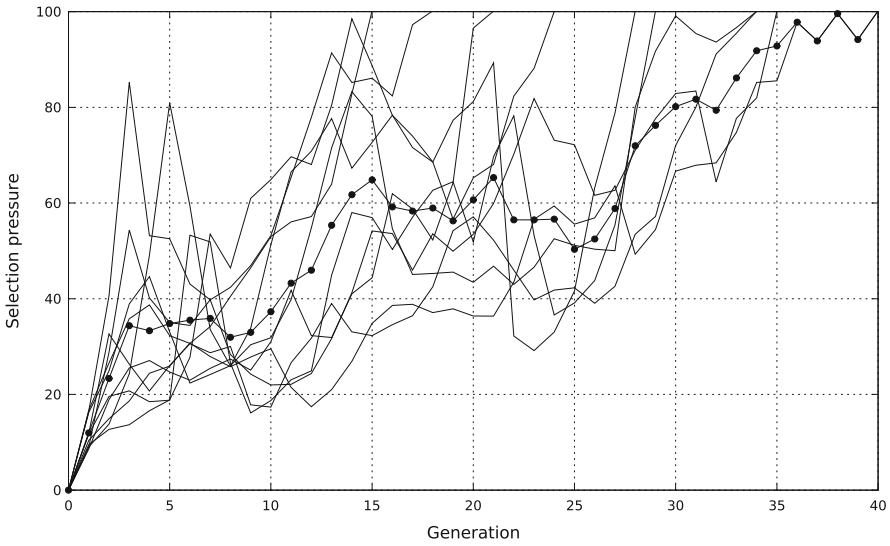


Fig. 10.3 OSGP selection pressure history

formula may require the combination of two or more function symbols. For instance, the subtrees $(* \; x_1 \; (* \; x_7 \; x_9))$ and $(* \; (* \; x_1 \; x_7) \; x_9)$ are equivalent to the canonical form $(* \; x_1 \; x_7 \; x_9)$ and should only be counted once when computing the building block occurrence frequency, which is defined as the ratio of trees in the population that contain the given building block.

In Figs. 10.4 and 10.5 we show typical frequency progresses for the *Poly-10* problem attacked by standard GP and OSGP (averages of 10 runs). Overall, we see that both GP and OSGP find all necessary terms of the correct solution to the problem and in fact both algorithms were able to solve ($R^2 > 0.999$ on the test) this problem in our experiments at least once.

OSGP tends to find relevant building blocks earlier in the run due to the additional effort spent on producing better offspring. Although each building block occurs with a certain frequency in the initial population, the bigger ones, $(* \; x_1 \; x_7 \; x_9)$ and $(* \; x_3 \; x_6 \; x_{10})$ are not favored by selection in the beginning. This illustrates the point that good structures tend to be harder to preserve in the exploratory phase of the algorithm, and may even become prematurely extinct, leading to convergence towards a suboptimal solution.

GP Genealogies

So far, we have seen that population diversity can be quantified at both the semantic and the syntactic levels. As diversity is related to selection, genetic programming algorithm designers and practitioners tried to use lineage information and the history

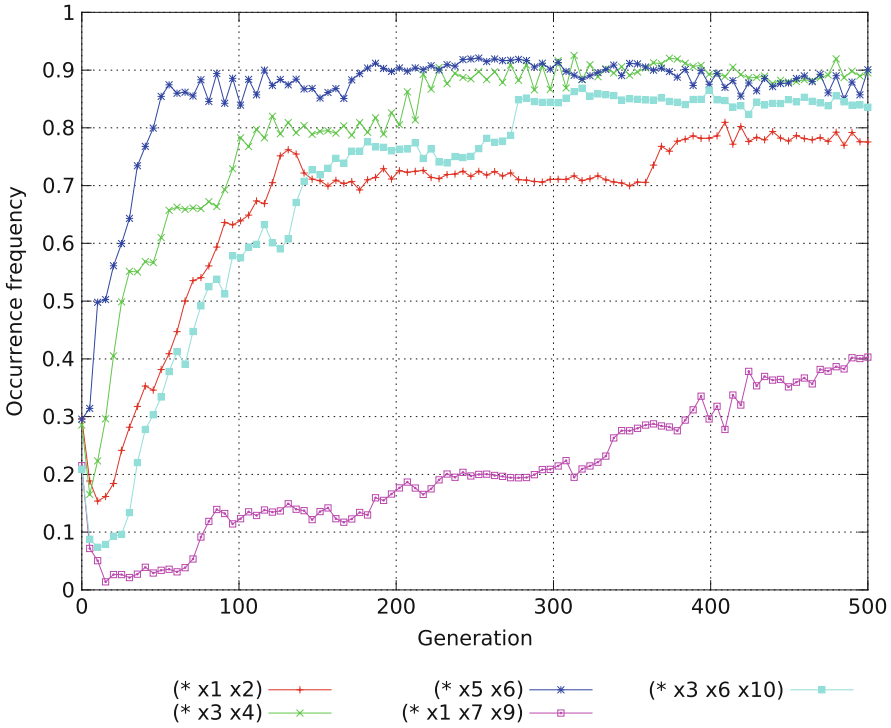


Fig. 10.4 Standard GP: evolution of the relevant building blocks for the Poly-10 problem

of the population to guide the selection process toward a more efficient exploration of the problem space (McPhee and Hopper 1999; Burke et al. 2004; Essam and Mckay 2004).

More recent work focused on the exploration of the complete genealogical history of the GP population, where he history can be viewed as the set union of the individual genealogies that are part of the evolutionary process.

By tracking selection and recombination events, a map of the parent-child relations and the respective exchanges of genetic materials is iteratively constructed. The information is stored as a directed acyclic graph in which vertices represent individuals and arcs represent parent-child relations. Each layer of vertices in the graph represents the population at a specific iteration of the algorithm. The vertices are ordered by descending fitness, so that the left-most vertex in a layer represents the best individual of that generation (Fig. 10.6). A more detailed description of the genealogy tracking is given in Burlacu et al. (2013).

We use the term *genetic fragment* to denote a subtree swapped by crossover or altered by mutation during the recombination phase. Graph arcs, oriented from the parent towards the child, contain the genetic fragment information and are used to traverse the graph and search for specific building blocks.

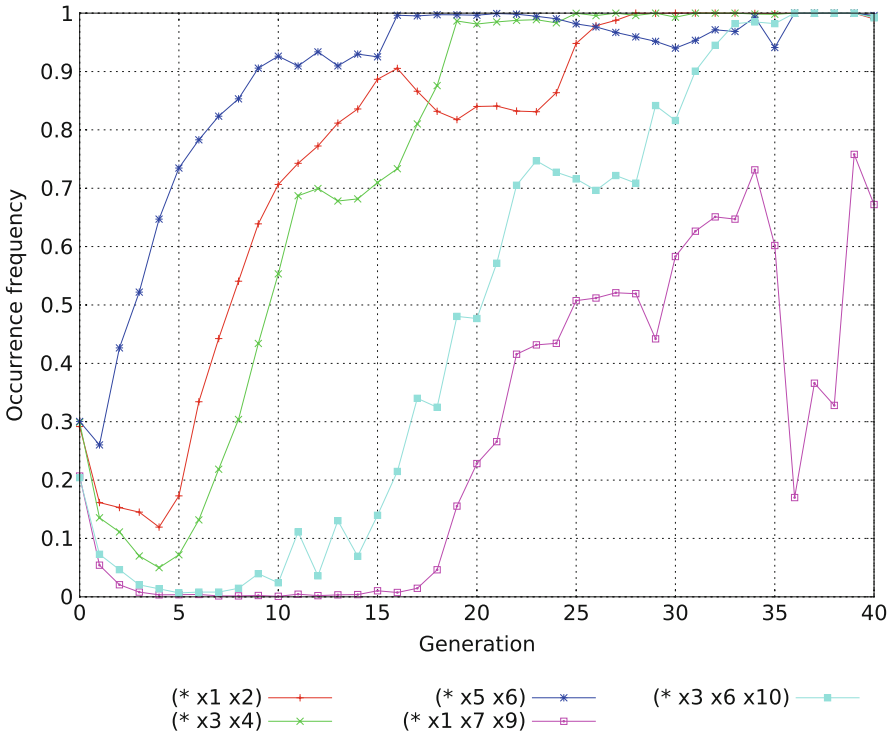


Fig. 10.5 OSGP: evolution of the relevant building blocks for the Poly-10 problem

We illustrate the construction of the genealogy graph on a different synthetic symbolic regression benchmark problem, the *Vladislavleva-5* (Vladislavleva 2008). For this problem it is possible to find a good solution with a small population size of 40 individuals evolved over 20 generations.

Figure 10.6 shows the root lineages and genealogies of the best solutions of SGP and OSGP (best run out of 20), with the OSGP run chosen to match the number of generations of SGP. The vertices belonging to the genealogy or root lineage of the best individual are highlighted in the graph and colored according to their fitness value, such that darker nodes represent less fit individuals.

We see the evolution of fitness in the best solution's ancestry, and the different behaviors of the two algorithms, with OSGP having a better average fitness per generation (less darker vertices) due to the strict offspring selection and adaptive selection pressure, and SGP a more "spread out" root lineage, signifying a more pronounced exploratory behavior in the first 10 generations of the run.

Although in the above example the dimensions of the graph were kept small for the purpose of visualization, in practice, the same approach can be used also for realistic parameter settings with larger populations and higher generation numbers, resulting in order of magnitude bigger graph sizes. Additionally, our approach provides an interface for the user, allowing the interactive exploration of lineages, genealogies and genetic fragments.

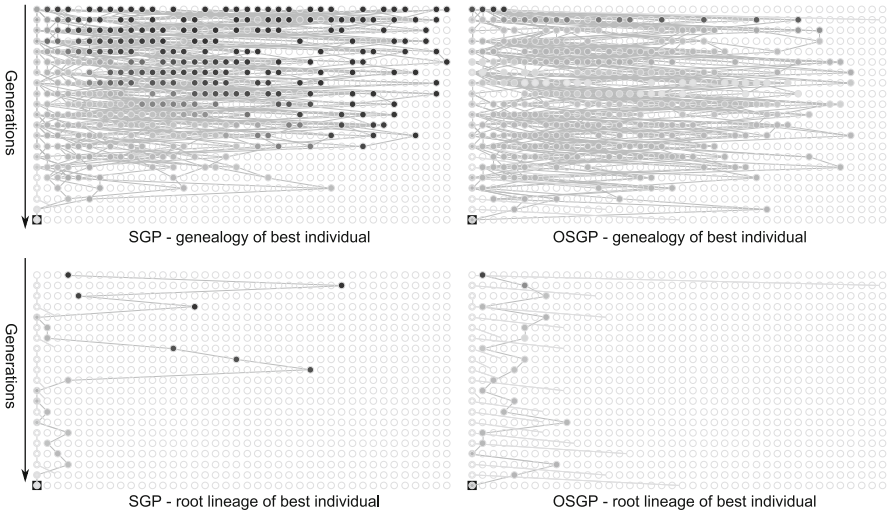


Fig. 10.6 Best solution genealogy and root lineage for SGP (*left*) and OSGP (*right*)

3 Enhanced Interpretability of Symbolic Regression Models

Symbolic regression creates models in the form of mathematical functions whose structure need not be specified a priori. As a result these models can be interpreted, transformed and validated by domain experts, which increases their trustworthiness compared to black box models. However, especially when working with real world data, the interpretability of symbolic regression models is hampered by the inconclusive usage of variables, overly complex formulas, or bloat phenomena. In the following several methods to strengthen this interpretability are presented.

Variable Relevance

The minimal subset of variables necessary to describe relations in an examined dataset is of great importance as it reduces the dimensionality of the problem, which eases the task of building predictors. Furthermore, redundant information in the data is detected, which allows to resolve ambiguities between similar models, that describe the exact same relation with different variables.

Symbolic regression includes an automatic feature selection step during the evolution of model structures to describe the dependent variable. This embedded feature selection step of symbolic regression has been proven to be highly competitive compared to other methods (Muni et al. 2006; Stijven et al. 2011), but more knowledge about the importance of features can be generated by analyzing which variables are present in the final models.

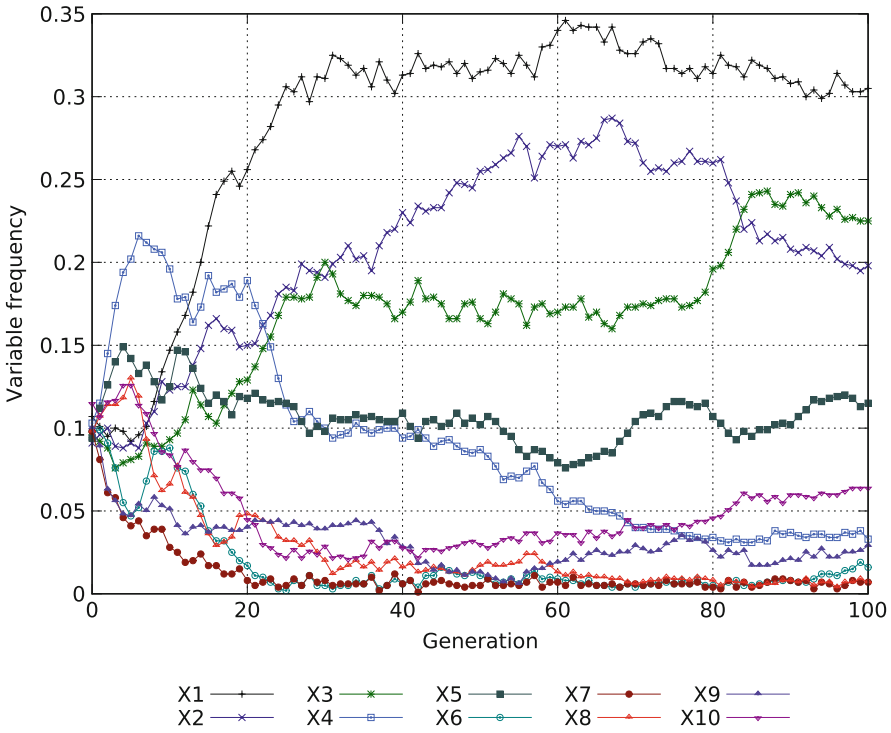


Fig. 10.7 Variable frequencies of a standard GP run

The most intuitive way of interpreting the relevance of variables is by assessing their presence in finally selected models. However, due to stochastic effects judging the relevance on one sole model is too naive and multiple models shall always be taken into account. A variable relevance metric for symbolic regression, where the relevance is the sum of models referencing it was introduced in Winkler (2009). A similar methodology is used in Smits et al. (2005), where models of a Pareto front (complexity vs. quality) are analyzed and the relative number of models referencing a variable is stated as the variable importance (optionally weighted by their fitness).

We extended these variable relevance metrics to include information about the algorithm run by taking all models into account and calculating the relative variable frequency on the whole population (Kronberger 2011). The analysis and visualization of the results can lead to insights on the algorithm dynamics itself and parameter configuration necessary to achieve the best possible performance on a given problem.

Figure 10.7 shows the different variable frequencies over 100 generations for X_1 to X_{10} for one exemplary standard GP run (population size 500) on *a simple function of 10 variables* (Friedman 1991). The variables X_1–X_5 are necessary to model the dependent variable, whereas X_6–X_{10} are superfluous. A disadvantage of tracking

Table 10.1 Variable impacts of the standard GP run displayed in Fig. 10.7

Related variables		Unrelated variables	
X_1	0.284	X_{10}	0.049
X_2	0.210	X_9	0.031
X_3	0.170	X_8	0.027
X_5	0.108	X_6	0.019
X_4	0.091	X_7	0.012

the relative variable frequencies during the algorithm runtime is that sequences of numbers are harder to interpret compared to a single metric. Therefore, we have aggregated the information in the graph in so-called variable impacts, by averaging the relative frequencies. In Table 10.1 the variable impacts for the same exemplary run as used in Fig. 10.7 are displayed and it clearly shows that impacts for related variables are almost twice as large as impacts of unrelated variables.

Model Simplification

Although GP produces white-box models these can be overly complex and bloated, which makes them rather hard to interpret. Methods to make the models more intelligible include static and dynamic tree size limits, pruning approaches, parsimony pressure inclusion, and multi-objective GP. All these methods modify the evolutionary search behavior of GP, however another possibility is to provide the user with enhanced post-analysis capabilities.

A straight-forward approach is to automatically simplify evolved models according to algebraic rules. The GP algorithm has no knowledge about the semantics of the generated trees and hence trees encoding constant expressions (e.g., $constant_1 + constant_2$, $variable_1 - variable_1$, $variable_2 * 0$) are evolved. Such expressions increase the tree length without providing any valuable information and should be pruned to allow an easier interpretation.

Another revealing analysis method is the visualization of node impacts, which quantify the impact of a node/subtree to the overall evaluation result. The node impact of a subtree is defined as the difference in the coefficient of determination R^2 between the original tree and a modified version, where the particular subtree is replaced by a constant value, its average evaluation result. Hence, node impacts are in the interval $[-1, 1]$ where a positive value states that without this subtree the quality of the model decreases and vice versa for negative values. Constant tree nodes automatically have a impact of 0.0, due to their average evaluation result being the same.

By combining automatic simplification and manual pruning according to node impacts the size of generated models can be significantly reduced. In Fig. 10.8 an originally evolved model (left side) and its simplified version (right side) are displayed. The coloring of the nodes visualizes the node impacts, where the absolute value is indicated by the saturation and green respectively red, whether the impact value is positive or negative. A dashed stroke indicates nodes, which are replaced

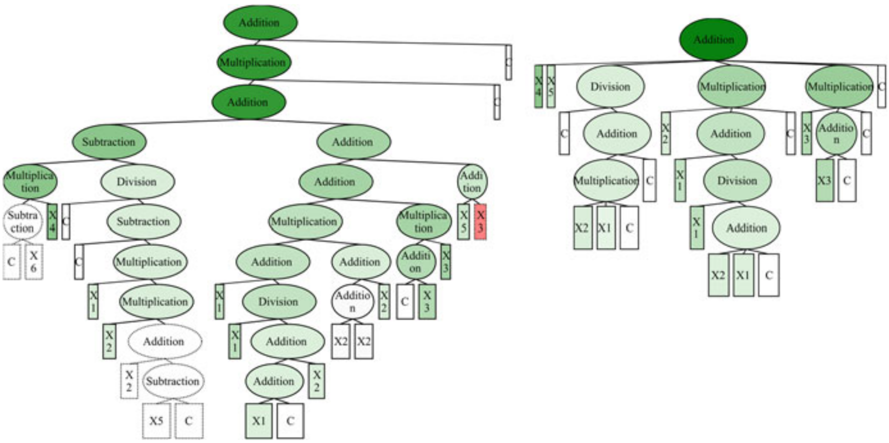


Fig. 10.8 Simplification of a symbolic regression model. The *left image* shows the original model (constants C are omitted), whereas the *right image* shows the same model after replacing *dashed nodes* and automatic simplification

by a constant (their average evaluation results) due to their low node impact and afterwards the tree is automatically simplified to get rid of constant expression. In this way the length of the symbolic expression tree could be reduced from 49 to 29 nodes without affecting its quality.

Networks and Analysis

Variable network analysis can be especially useful if the analyzed system or data set is only poorly understood. In such situations there is frequently no single target variable to be modeled; instead multiple variables can be considered as targets. Additionally, strong dependencies between input variables are often prevalent. In such situations the data analysis process is more exploratory than goal-driven; the aim is to identify groups of strongly related variables and independent variables to improve the understanding of the system as a whole. We have previously used this approach for exploratory analysis of data sets from different domains including economics (Kronberger et al. 2011) and medicine (Winkler et al. 2011).

Causal networks (Pearl 2009) and similar representations are a popular way for representing dependencies of random variables. Such networks are usually built manually based on expert knowledge, because deriving causal networks automatically directly from observations is rather difficult, especially if the set of variables is large or if non-linear dependencies have to be identified.

Using the techniques described in the previous sections it is rather easy to create a network of variable dependencies based on symbolic regression models. The advantages are the capability to identify non-linear models and automatic feature

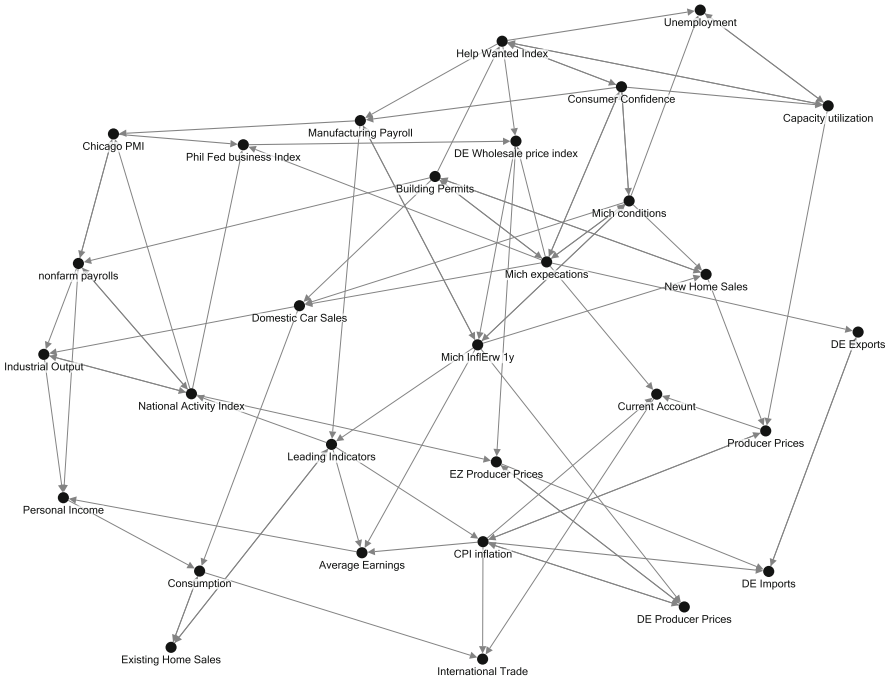


Fig. 10.9 An exemplary variable network for a data set of economic and financial variables and indicators (Kronberger et al. 2011)

selection. To create the network we collect results from several symbolic regression runs using each variable as the target variable. The dependency network can then be derived from the model accuracies and variable relevances.

Figure 10.9 shows a network of dependencies of economic and financial variables and indicators based on the results of multiple symbolic regression runs. In this example, the top three most relevant input variables are shown for each dependent variable and it is possible to segregate variables into several clusters of strongly connected variables.

Networks derived from symbolic regression results are useful for exploratory data analysis but more importantly such networks are also helpful to reduce the set of input variables for symbolic regression models and to resolve structurally different but semantically similar models.

4 Conclusion

In this chapter we have focused on the analysis of symbolic regression runs and on the analysis of achieved modeling results. The objective throughout has been to provide measures to support the process of symbolic regression algorithm

analysis and algorithm design as well as to support model interpretation with respect to a specific problem domain. The proposed analysis methods have been discussed exemplarily on the basis of benchmark problem instances with the aim to highlight the potential of analysis methodology. All presented techniques are publicly available and have been implemented using the open source optimization system HeuristicLab[1] designed and developed by our group.

Acknowledgements The work described in this chapter was done within the Josef Ressel Center for Heuristic Optimization *Heureka!* sponsored by the Austrian Research Promotion Agency (FFG).

References

Affenzeller M, Wagner S (2004) SASEGASA: a new generic parallel evolutionary algorithm for achieving highest quality results. J Heuristics Spec Issue New Adv Parallel Meta-Heuristics Complex Probl 10:239–263

Affenzeller M, Winkler S, Wagner S, Beham A (2009) Genetic algorithms and genetic programming: modern concepts and practical applications. Numerical Insights. CRC, Singapore

Altenberg L (1994) The evolution of evolvability in genetic programming. In: Kinnear KE Jr (ed) Advances in genetic programming. MIT, Cambridge, chap 3, pp 47–74

Banzhaf W, Langdon WB (2002) Some considerations on the reason for bloat. Genet Program Evolvable Mach 3(1):81–91

Burke EK, Gustafson S, Kendall G (2004) Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Trans Evol Comput 8(1):47–62

Burlacu B, Affenzeller M, Kommenda M, Winkler SM, Kronberger G (2013) Visualization of genetic lineages and inheritance information in genetic programming. In: Proceedings of the GECCO'13: VizGEC workshop, Amsterdam (accepted to be published)

Ekart A, Nemeth SZ (2000) A metric for genetic programs and fitness sharing. In: Proceedings of EuroGP'2000 genetic programming, Edinburgh. LNCS, vol 1802. Springer, pp 259–270

Essam D, Mckay RI (2004) Heritage diversity in genetic programming. In: 5th international conference on simulated evolution and learning, Busan

Friedman JH (1991) Multivariate adaptive regression splines. Ann Stat 19(1):1–141

Jackson D (2010) The identification and exploitation of dormancy in genetic programming. Genet Program Evolvable Mach 11(1):89–121

Keijzer M (1996) Efficiently representing populations in genetic programming. In: Angeline PJ, Kinnear KE Jr (eds) Advances in genetic programming 2. MIT, Cambridge, chap 13, pp 259–278

Kotanchek M, Smits G, Vladislavleva E (2007) Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models. In: Genetic programming theory and practice V, genetic and evolutionary computation. Springer, Ann Arbor, chap 12, pp 201–220

Kotanchek ME, Vladislavleva E, Smits GF (2013) Symbolic regression is not enough: it takes a village to raise a model. In: Genetic programming theory and practice X, genetic and evolutionary computation, vol 10. Springer, Ann Arbor, chap 13, pp 187–203

[1] http://dev.heuristiclab.com

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Kronberger G (2011) Symbolic regression for knowledge discovery. Schriften der Johannes Kepler Universität Linz, Universitätsverlag Rudolf Trauner

Kronberger G, Fink S, Kommenda M, Affenzeller M (2011) Macro-economic time series modeling and interaction networks. In: EvoApplications (2). Lecture notes in computer science, vol 6625. Springer, Berlin/New York, pp 101–110

Langdon WB, Poli R (2002) Foundations of genetic programming. Springer, Berlin/New York

McPhee NF, Hopper NJ (1999) Analysis of genetic diversity through population history. In: Proceedings of the genetic and evolutionary computation conference, Orlando, vol 2. Kaufmann, pp 1112–1120

Muni DP, Pal NR, Das J (2006) Genetic programming for simultaneous feature selection and classifier design. IEEE Trans Syst Man Cybern Part B 36(1):106–117

Pearl J (2009) Causality: models, reasoning and inference, 2nd edn. Cambridge University Press, New York

Poli R (2003) A simple but theoretically-motivated method to control bloat in genetic programming. In: proceedings of EuroGP'2003 genetic programming, Essex. LNCS, vol 2610. Springer, pp 204–217

Rosca JP (1995) Entropy-driven adaptive representation. In: Rosca JP (ed) Proceedings of the workshop on genetic programming: from theory to real-world applications, Tahoe City, pp 23–32

Smits G, Kordon A, Vladislavleva K, Jordaan E, Kotanchek M (2005) Variable selection in industrial datasets using pareto genetic programming. In: Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice III, genetic programming, vol 9. Springer, Ann Arbor, chap 6, pp 79–92

Stijven S, Minnebo W, Vladislavleva K (2011) Separating the wheat from the chaff: on feature selection and feature importance in regression random forests and symbolic regression. In: 3rd symbolic regression and modeling workshop for GECCO 2011, Dublin. ACM, pp 623–630

Vanneschi L, Gustafson S, Mauri G (2006) Using subtree crossover distance to investigate genetic programming dynamics. In: Proceedings of the 9th European conference on genetic programming, lecture notes in computer science, Budapest, vol 3905. Springer, pp 238–249

Vladislavleva E (2008) Model-based problem solving through symbolic regression via pareto genetic programming. PhD thesis, Tilburg University

Winkler SM (2009) Evolutionary system identification: modern concepts and practical applications. Johannes Kepler University, Linz, Reihe C, vol 59. Trauner, Linz

Winkler SM, Affenzeller M, Kronberger G, Kommenda M, Wagner S, Jacak W, Stekel H (2011) Analysis of selected evolutionary algorithms in feature selection and parameter optimization for data based tumor marker modeling. In: EUROCAST (1). Lecture notes in computer science, vol 6927. Springer, Berlin/New york, pp 335–342

Chapter 11
Geometric Semantic Genetic Programming for Real Life Applications

Leonardo Vanneschi, Sara Silva, Mauro Castelli, and Luca Manzoni

Abstract In a recent contribution we have introduced a new implementation of geometric semantic operators for Genetic Programming. Thanks to this implementation, we are now able to deeply investigate their usefulness and study their properties on complex real-life applications. Our experiments confirm that these operators are more effective than traditional ones in optimizing training data, due to the fact that they induce a unimodal fitness landscape. Furthermore, they automatically limit overfitting, something we had already noticed in our recent contribution, and that is further discussed here. Finally, we investigate the influence of some parameters on the effectiveness of these operators, and we show that tuning their values and setting them "a priori" may be wasted effort. Instead, if we randomly modify the values of those parameters several times during the evolution, we obtain a performance that is comparable with the one obtained with the best setting, both on training and test data for all the studied problems.

L. Vanneschi (✉)
ISEGI, Universidade Nova de Lisboa, 1070-312 Lisboa, Portugal
INESC-ID, IST, Universidade Técnica de Lisboa, 1000-029 Lisboa, Portugal
D.I.S.Co., University of Milano-Bicocca, 20126 Milan, Italy
e-mail: lvanneschi@isegi.unl.pt

S. Silva
INESC-ID, IST, Universidade Técnica de Lisboa, 1000-029 Lisboa, Portugal
CISUC, Universidade de Coimbra, 3030-290 Coimbra, Portugal
e-mail: sara@kdbio.inesc-id.pt

M. Castelli
ISEGI, Universidade Nova de Lisboa, 1070-312 Lisboa, Portugal
INESC-ID, IST, Universidade Técnica de Lisboa, 1000-029 Lisboa, Portugal
e-mail: mcastelli@isegi.unl.pt

L. Manzoni
D.I.S.Co., University of Milano-Bicocca, 20126 Milan, Italy
e-mail: luca.manzoni@i3s.unice.fr

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*,
Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_11,

Keywords Geometric semantic operators • Fitness landscapes • Overfitting • Parameter tuning

1 Introduction

A recent trend among researchers in Genetic Programming (GP) (Koza 1992) is the attempt of integrating semantic awareness in the genetic operators. Several definitions of semantics are possible for GP individuals. However, one of the most commonly used ones, also employed here, is the vector of output values obtained on a set of input data. Based on this definition, Moraglio and coworkers (2012) have recently introduced geometric semantic operators for GP. They have the extremely interesting property of inducing a unimodal fitness landscape (i.e. error surface) for any problem consisting in matching input data into known target outputs (like regression and classification), but they have the strong limitation of producing offspring that are larger than their parents, causing an exponential growth in the size of the programs. To overcome this limitation, in Vanneschi et al. (2013) we have defined a new implementation of these operators, which allows us to use them efficiently. This implementation has paved the way to a deeper investigation on the usefulness and properties of these operators on several applications, including real life problems.

In this chapter, after describing geometric semantic crossover and mutation for the case of regression problems, we revise and extend the description of our implementation and finally apply it to four different complex real life problems. Our main objectives are:

- To understand whether inducing a unimodal fitness landscape actually brings a benefit in terms of optimization ability;
- To study the generalization ability of geometric semantic operators, already suggested in Vanneschi et al. (2013);
- To study how some important parameters, typical of these operators, influence the ability of GP to find good solutions, offering suggestions on how to set them.

To accomplish these tasks, for each one of the considered applications, we study the behavior of the geometric semantic GP framework both on training data and on unseen test samples. We use different variants of the framework, with both geometric semantic crossover and mutation, with mutation in isolation, and with different parameter values. We also suggest a new variant in which the parameter values are not set "a priori" by the user, but instead are randomly updated several times during the evolution.

The paper is organized as follows: Sect. 2 presents the state of the art concerning the use of semantics in GP. Section 3 describes the geometric semantic operators introduced by Moraglio and coworkers in 2012. Section 4 discusses our implementation of those operators, already introduced in Vanneschi et al. (2013). In Sect. 5 we

address our experimental study, describing the considered test problems, presenting the experimental settings and reporting and discussing the obtained results. Finally, Sect. 6 concludes the paper and provides hints for future research.

2 Semantics in GP: Literature Review

Several recent contributions have proposed the definition of semantic based methods for GP. In McPhee et al. (2008) the authors used truth tables to analyze behavioral changes in crossover for boolean problems, discovering that very often crossover has no effect on semantics. For this reason, in Beadle and Johnson (2008) semantics were used to define a new type of crossover called Semantically Driven Crossover (SDC). If the offspring are semantically equivalent to their parents, the children are discarded and the crossover is repeated. This process is iterated until semantically different children are found. The authors argue that this results in increased semantic diversity in the evolving population and a consequent improvement in the GP performance. In the same line of research, the authors of Uy et al. (2010) investigated the role of syntactic locality and semantic locality of crossover in GP. Their results show that improving syntactic locality reduces code growth and leads to a slight improvement of the ability to generalize. On the other hand, improving semantic locality significantly enhances GP performance, reduces code growth and substantially improves the ability of GP to generalize. In Nguyen et al. (2009a) the authors proposed Semantics Aware Crossover (SAC), a crossover operator promoting semantic diversity, based on checking semantic equivalence of subtrees. It was subsequently extended to Semantic Similarity based Crossover (SSC) (Uy et al. 2011), which turned out to perform better than both standard crossover and SAC (Uy et al. 2011). The idea of SSC was then extended to mutation, with the definition of Semantic Similarity based Mutation (SSM) (Nguyen et al. 2009b). In Beadle and Johnson (2009) the authors presented a technique known as Semantically Driven Mutation (SDM), which can explicitly detect and apply behavioral changes caused by the syntactic modifications in programs produced by mutation. The SDM algorithm does not accept mutated programs when they are behaviorally equivalent to the original ones.

In Krawiec (2012) the authors proposed a class of crossover operators for GP aimed at making offspring programs intermediate (or medial) with respect to parent programs in the semantic space. The authors pointed out that the prospects of designing a crossover operator that works in the genotype space and behaves geometrically in the corresponding semantic space are gloomy in GP, given the complexity of the genotype-phenotype mapping. Hence, rather than guaranteeing the geometric behavior, their operator tries to approximate it by analyzing the offspring after it has been created. This limitation is overcome by the geometric semantic operators proposed in Moraglio et al. (2012), which are the operators used in this chapter. In Sect. 3 we explain them, while Sect. 4 describes our implementation of those operators (first presented in Vanneschi et al. (2013)) that makes them efficient and usable in practice.

3 Geometric Semantic Operators

While the semantically aware methods cited in Sect. 2 often produced superior performance with respect to traditional methods, many of them are indirect: search operators act on the syntax of the parents to produce offspring, which are then accepted only if some semantic criterion is satisfied. As reported in Moraglio et al. (2012), this has at least two drawbacks: (i) these implementations are very wasteful as they are heavily based on trial-and-error; (ii) they do not provide insights on how syntactic and semantic searches relate to each other. To overcome these drawbacks, Moraglio and coworkers introduced in Moraglio et al. (2012) new operators that directly search the semantic space.

To explain the idea, let us first give an example using Genetic Algorithms (GAs). Let us consider a GA problem in which individuals are strings of prefixed length of real numbers, the target solution is known and the fitness of each individual corresponds to its distance to the target (our reasoning holds for any distance measure used). This problem is clearly a purely theoretical one, given that in real applications the target is unknown. Nevertheless, for understanding the arguments that will follow, it is important to convince oneself that this problem is characterized by a very good evolvability, and it is in general easy to solve for GAs. In fact, for instance, if we use *ball mutation* (i.e. a mutation operator whose effect is a, typically "weak", perturbation of the value of a subset of the coordinates of an individual), any possible individual different from the global optimum has at least one neighbor (individual resulting from its mutation) that is closer than itself to the target, and thus fitter. So, there are no local optima. In other words, the fitness landscape is *unimodal*, which generally implies good evolvability. Similar considerations hold for many types of crossover, including various kinds of geometric crossover (Krawiec 2012).

Now, let us consider the typical supervised learning GP problem of finding a function that maps sets of input data into known target outputs (like regression and classification). The fitness of an individual for this problem is usually a distance between its predicted values and the expected target ones (error measure). Now, let us assume that we are able to find a transformation on the syntax of the individuals, whose effect is just a "weak" random perturbation of some of their predicted values. In other words, let us assume that we are able to transform an individual G into an individual H whose output is like the output of G, except for some values, that are slightly perturbed of a random amount. Under this hypothesis, we are able to map the considered GP problem into the GA problem discussed above, in which ball mutation is used (basically, our syntactic transformation would "correspond" to ball mutation on the semantic space). So, this transformation, if known, would induce a unimodal fitness landscape on every problem like the considered one (e.g. regressions and classifications), making those problems easily evolvable by GP. Similarly, we are also interested in finding transformations on the syntax of GP individuals that correspond, on their semantics, to GA operators with known properties, like for instance GAs geometric crossovers. This idea of looking for such operators is very ambitious and extremely challenging: finding those operators would allow us to directly search the space of semantics. At the same time, for

all supervised learning applications (i.e. where the semantic target is known), this would allow us to work on unimodal fitness landscapes; in other words, the error surface would be unimodal independently from the complexity of the data.

Although not without limitations, (Moraglio et al. 2012) accomplishes this task, defining new operators that have exactly these characteristics. Here we report the definition of the geometric semantic operators as given in Moraglio et al. (2012) for real function domains, since these are the ones used here. For applications that consider other types of data, the reader is referred to Moraglio et al. (2012).

Definition 11.1 (Geometric Semantic Crossover). Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$.

The interested reader is referred to Moraglio et al. (2012) for a proof of the fact that this operator corresponds to a geometric crossover on the semantic space, in the sense that it produces an offspring that stands between its parents in this space. Nevertheless, even without a formal proof, we can have an intuition of it by considering that the (only) offspring generated by this crossover has a semantic vector that is a linear combination of the semantics of the parents with random coefficients included in $[0, 1]$ and whose sum is equal to 1. Moraglio et al. (2012) also prove an interesting consequence of this fact: the fitness of the offspring cannot be worse than the fitness of the worst of its parents. Also in this case we do not replicate the proof here, but we limit ourselves to provide a visual intuition of this property: in Fig. 11.1a we report a simple two-dimensional semantic space in which we draw a target point T (whose coordinates are the target values for each input instance in the training set), the semantics of two parents P_1 and P_2, and of one of their possible offspring O (which by construction is included between its parents). It is clear that, if fitness corresponds to the distance to T, O is better than P_2 (which is the worst parent in this case). The generality of this property is proven in Moraglio et al. (2012). To constrain T_R in producing values in $[0, 1]$, we use the sigmoid function: $T_R = \frac{1}{1+e^{-T_{rand}}}$ where T_{rand} is a random tree with no constraints on the output values.

Definition 11.2 (Geometric Semantic Mutation). Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step ms returns the real function $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions.

Moraglio et al. (2012) prove that this operator corresponds to ball mutation on the semantic space and induces a unimodal fitness landscape. However, even though without a formal proof, it is not difficult to have an intuition of it, considering that each element of the semantic vector of the offspring is a "weak" perturbation of the corresponding element in the parent's semantics. We informally define this perturbation as "weak" because it is given by a random expression centered in zero (the difference between two random trees). Nevertheless, by changing the parameter ms, we are able to tune the "step" of the mutation and thus the magnitude of this perturbation.

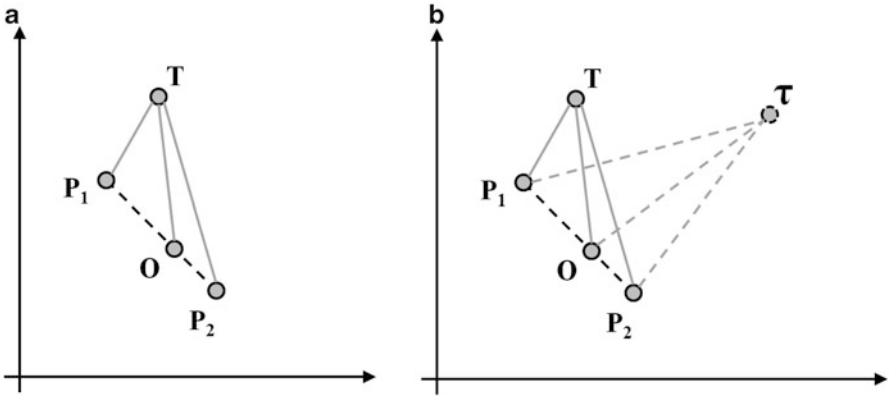


Fig. 11.1 **Plot (a)**: a simple theoretical two-dimensional semantic space that we use to explain some properties of the geometric semantic crossover presented in Moraglio et al. (2012). Geometric semantic crossover produces an offspring that is at least not worse than the worst of its parents. In this simple case, offspring O (which stands between parents P_1 and P_2 by construction) is clearly closer to target T than parent P_2. **Plot (b)**: as discussed in Sect. 5, the geometric properties of geometric semantic operators hold also on test data. For instance, offspring O is not worse than the worst of the parents not only considering the training target T, but also considering any hypothetical test target τ (even though the position of τ is unknown at learning time)

Limitation of Moraglio's Geometric Semantic Operators. We point out that, with each application of these operators, an offspring is produced that contains the complete structure of the parents, plus one or more random trees as its subtrees and some arithmetic operators: the size of the offspring is thus clearly much larger than the size of their parents. The growth in size of the individuals in the population is exponential (as demonstrated in Moraglio et al. (2012)), which makes these operators unusable in practice: after a few generations, the population becomes unmanageable because the fitness evaluation process becomes unbearably slow. The solution that is suggested in Moraglio et al. (2012) consists in performing an automatic simplification step after every generation, in which the programs are replaced by (hopefully smaller) semantically equivalent ones. However, this additional step (which should be performed using one of the existing software packages for the automatic simplification of expressions, like for instance MATLAB or Mathematica, or by implementing a new one from scratch) adds to the computational cost of GP and is only a partial solution to the progressive program size growth. Last but not least, according to the particular language used to code the GP individuals and the used primitives, automatic simplification can be a very hard task. To overcome this limitation, in Vanneschi et al. (2013) we have presented an implementation of these operators which is efficient and does not require any simplification step. This implementation is described in the next section.

4 An Efficient Implementation of Geometric Semantic Operators

Here we present the proposed implementation of a GP system that uses the geometric semantic crossover and mutation described in Sect. 3. We designate this system as GS-GP (which stands for Geometric Semantic GP). This implementation was first introduced in Vanneschi et al. (2013), where it was described quite informally. Here we show the pseudocode and discuss it in more detail. Note that, although we describe the algorithm assuming the representation of the individuals is tree based, the implementation fits any other type of representation. The pseudocode of the proposed implementation is shown in Fig. 11.2. The basic idea of this algorithm is the one of generating the tree structure only of the individuals that are in the initial population and of all the random trees needed to apply the operators. The tree structure of the individuals of the subsequent generations will not be explicitly built. On the contrary, only the information needed to generate these trees is stored in memory. This information is basically a set of memory pointers, and the name of the operators (crossover or mutation) that must be used. For any individual τ, a *reference* (or memory pointer) to τ has been represented as $\&\tau$ in the pseudocode, using a C-likes notation. Furthermore, the semantics of the individuals in the current population are also stored in memory, exploiting the fact that the semantics of the

```
Create an empty repository P of individuals;
Create an empty matrix S of real numbers;
Create an empty matrix S_rand of real numbers;
Create n random individuals and store them in P;
Evaluate the individuals in P and store their semantics in S;
Create an empty structure M;
for each generation:
   Create a new empty matrix S_new of real numbers;
   repeat n times:
      Choose a genetic operator;
      if the chosen operator is crossover then
          Generate a random tree R and store it in P;
          Evaluate R and store its semantics in S_rand;
          Select two individuals T_1 and T_2;
          Store(crossover,⟨&T_1,&T_2,&R⟩)in M;
          Store offspring's semantics in S_new;
      else if the chosen operator is mutation then
          Generate two random trees R_1 and R_2 and store them in P;
          Evaluate R_1 and R_2 and store their semantic in S_rand;
          Select one individual T_1;
          Store(mutation,⟨&T_1,&R_1,&R_2 ⟩)in M;
          Store off spring's semantics in S_new;
      end if
   end repeat
   Replace S by S_new;
   Erase all the rows of P, M and S_rand not referring to ancestors;
end for
```

Fig. 11.2 The pseudocode of our implementation of geometric semantic GP. With n we indicate the population size (number of individuals contained in the population). For any individual τ, with the notation $\&\tau$ we indicate a *reference* (or memory pointer) to τ

offspring of crossover and mutation are easily calculated from the semantics of the ancestors[1] by applying the definition of the operators. This algorithm uses the following data structures:

- A repository of individuals `P`. It stores the tree structures of the individuals in the initial population and of the random trees that are generated by the algorithm along the evolution.
- A matrix of real numbers `S`. The dimension of this matrix is $n \times k$, where n is the size of the population (number of individuals it contains), and k is the number of instances in the training set (number of fitness cases). Each line i of matrix `S` is used to store the semantic vector of the ith individual in the population.
- A matrix of real numbers $\texttt{S}_{\texttt{rand}}$. The dimension of this matrix is $m \times k$, where m is the number of random trees that are generated by the algorithm during the whole execution and k is the number of fitness cases. Each line of this matrix is used to store the semantic vector of a random tree. Either m is known "a priori" or $\texttt{S}_{\texttt{rand}}$ is implemented as a dynamically growing structure of k-dimensional vectors.
- A matrix of real numbers $\texttt{S}_{\texttt{new}}$. The dimension of this matrix is $n \times k$, like matrix `S`. $\texttt{S}_{\texttt{new}}$ is used to store the semantic vectors of the individuals in the "next" population (individuals whose tree structures are not built). At the end of each generation, $\texttt{S}_{\texttt{new}}$ replaces `S`.
- A structure `M`. Basically, this structure provides all the information for reconstructing the individuals that take part in the evolution. It is dynamic, in the sense that its size grows during the evolution, and it contains objects in the following format: $(operator, \langle ancestor_1, ancestor_2, ancestor_3 \rangle)$. The field *operator* is used to know how to compose the ancestors to reconstruct the offspring. It can be either crossover or mutation and it can be implemented using a boolean value or a bit. The fields ancestor_i are *references* (or memory pointers) to individuals.

In the pseudocode shown in Fig. 11.2, selection deserves a particular discussion. In fact, at generation 1 individuals are selected among the ones that are stored in the first n positions of `P` (and they are trees), while at the subsequent generations they are selected from the last n lines in `M` (and they are memory pointers). Analogously, the first n lines of `M` contain pointers to objects in `P`, while the subsequent ones contain pointers to objects in `P` (the random trees) and in previous lines of `M` (the individuals of the previous generation, whose tree structure has not been built).

We finally point out that, for simplicity, the pseudocode of Fig. 11.2 does not take into account the operator of reproduction (i.e. the copy of individuals unchanged into the next population), which is implemented in our system by simply performing a copy of the lines in `M` corresponding to these individuals. The interested reader is referred to Vanneschi et al. (2013) for a numeric example of the implementation proposed here, which should further clarify the functioning of the algorithm in Fig. 11.2.

[1]Both in the text and in the pseudocode of Fig. 11.2, we abuse the term "ancestors" to designate not only the parents but also the random trees used to build an individual by crossover or mutation.

Computational Complexity of the Proposed Implementation. In terms of computational time, we emphasize that the process of calculating the semantics of the individuals is very efficient as it does not require the evaluation of the trees. Indeed, evaluating each individual requires (except for the initial generation) a constant time, which is independent from the size of the individual itself. In terms of memory, structures `P` and `M` grow during the run. However, `P` adds a maximum of $2 \times n$ rows per generation (if all new individuals are created by mutation) and table `M` (which basically contains only memory pointers) adds a maximum of n rows per generation. Even if we never erase the "ex-ancestors" from these structures (and never reuse random trees, which is also possible), we can manage them efficiently for several thousands of generations. Let us briefly consider the cost in terms of time and space of evolving a population of n individuals for g generations. At every generation, we need $O(n)$ space to store the new individuals. Thus, we need $O(ng)$ space in total. Since we need to do only $O(1)$ operations for evaluating new individuals (since the fitness can be computed using the fitness of the parents), the time complexity is also $O(ng)$.

The final step of the algorithm is performed after the end of the last generation. In order to reconstruct the individuals, we may need to "unwind" our compact representation and make the syntax of the individuals explicit. Therefore, despite performing the evolutionary search very efficiently, in the end we may still have to deal with the large trees that characterize the standard implementation of geometric semantic operators. However, most probably we will only be interested in the best individual found, so this unwinding (and recommended simplification) process may be required only once, and it is done offline after the search is finished. This greatly contrasts with the solution proposed by Moraglio et al. of building and simplifying every tree in the population at each generation online with the search process. If we are not interested in the form of the optimal solution, we can avoid the "unwinding phase" and we can evaluate an unseen test input with a time complexity of $O(ng)$. In this case, the individual is used as a "black-box" which, in some cases, may be acceptable. Excluding the time needed to build and simplify the best individual, the proposed implementation allowed us to evolve populations for thousands of generations with a considerable speed up with respect to standard GP.

5 Experimental Study

The objectives of this section are three: the first one is to compare standard GP (ST-GP) with GP that uses geometric semantic crossover and mutation (GS-GP). The second one is to test the performance of GP using only geometric semantic mutation with several mutation steps. The third one is to study a system in which the rates of geometric semantic crossover and mutation, and the mutation step are not decided "a priori", and instead are randomly updated several times during the evolution (this last GP variant is represented as RND from now on). In all these cases, we are interested in studying the performance of the considered GP variants

both on training data and on unseen test cases, in order to have information about their generalization ability. Before discussing the experimental results, let us briefly describe the test problems used and the experimental setting.

Test Problems Used. The first three considered applications are real life problems in the field of pharmacokinetics. They have been chosen because they have been tackled in several previous studies, in particular using GP (see for instance Archetti et al. (2007)). They consist in predicting the value of three different pharmacokinetic parameters of a set of candidate drug compounds based on their molecular structure. The first pharmacokinetic parameter we consider is human oral bioavailability (represented as %F from now on), the second one is plasma protein binding level (or %PPB) and the third one is median lethal dose (or LD50), also informally called toxicity. %F is the parameter that measures the percentage of the initial orally submitted drug dose that effectively reaches the systemic blood circulation after being filtered by the liver. %PPB consists in the percentage of the initial drug dose which binds plasma proteins. LD50 refers to the amount of compound required to kill 50% of the test organisms (cavies). For a more detailed discussion of these pharmacokinetic parameters the reader is referred to Archetti et al. (2007). The datasets we have used are the same as in Archetti et al. (2007): the %F (respectively %PPB and LD50) dataset consists in a matrix composed by 260 (respectively 131 and 234) rows (instances) and 242 (respectively 627 and 627) columns (features). Each row is a vector of molecular descriptor values identifying a drug; each column represents a molecular descriptor, except the last one, that contains the known target values of the considered pharmacokinetic parameter. These datasets can be downloaded from:

http://kdbio.inesc-id.pt/~sara/gptp2013/bioavailability.txt,
http://kdbio.inesc-id.pt/~sara/gptp2013/ppb.txt, and
http://kdbio.inesc-id.pt/~sara/gptp2013/toxicity.txt.

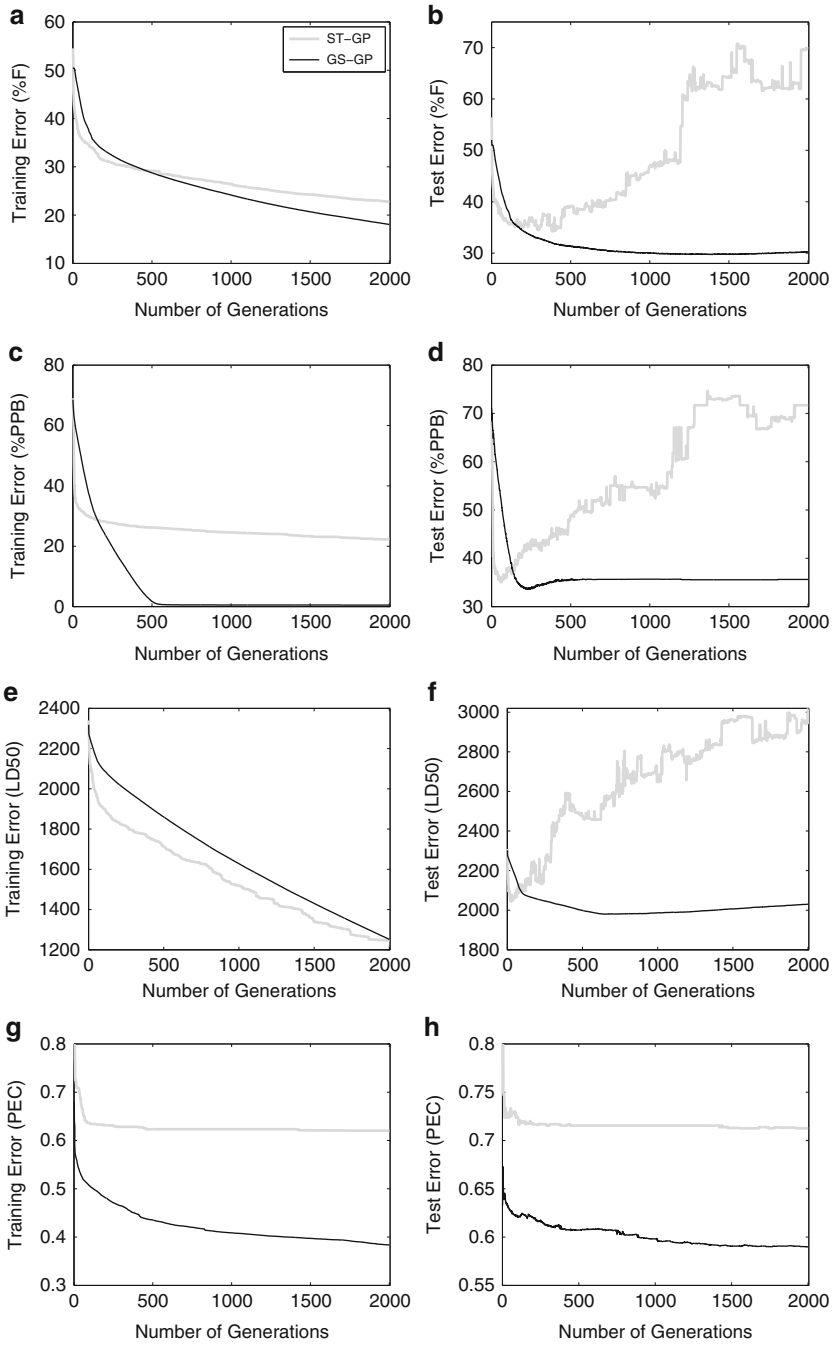
The remaining application we have used in our experiments consists in a regression problem over a dataset containing historical load electricity data and weather information in the south Italy area collected by TERNA (the owner of the Italian transmission grid and responsible for energy transmission and dispatching) in the years between 1999 and 2010. In this problem, the objective is to forecast the energy consumption (or load) in a given day t, using the past samples of the load and the other information provided in the dataset (like weather information) for all the previous days up to day $t - 1$ (1-day ahead forecasting). Data include temperatures, pressure values, wind speed and other weather related information, collected by using detection devices arranged in the area of interest. Data from 1999 to 2006 have been used during the training phase, while the remaining data have been used as test a set to assess the quality of the forecasting. The dataset consists of 45 variables and 240 instances, and it will be called PEC (which stands for Prediction of Energy Consumption) from now on.

Experimental Settings. For all the studied problems, except PEC, training and test sets have been obtained by random splitting: at each different GP run, 70% of the instances have been randomly selected with uniform probability and inserted into the training set, while the remaining 30% formed the test set. A total of 30 runs were performed for all the GP variants that we have studied. All the runs used populations of 100 individuals evolved for 2,000 generations. Tree initialization was performed with the Ramped Half-and-Half method (Koza 1992) with a maximum initial depth equal to 6. The function set contained the four binary arithmetic operators $+$, $-$, $*$, and $/$ protected as in Koza (1992). The terminal set contained a number of variables corresponding to the number of features in each dataset. Selection was done with tournaments of size 4. To create new individuals, ST-GP used standard (subtree swapping) crossover (Koza 1992) and (subtree) mutation (Koza 1992) with probabilities 0.9 and 0.1, respectively. For GS-GP the mutation rate was 0.5. Preliminary tests have shown that the geometric semantic operators require a relatively high mutation rate in order to be able to effectively explore the search space. The ms step used was 0.001 as in Moraglio et al. (2012). For both systems, survival was elitist: the best individual was always copied unchanged into the next generation. No maximum tree depth limit has been imposed during the evolution. The results reported next consist in curves of the fitness (RMSE) on the training and test sets. More particularly, for each generation the training fitness of the best individual, as well as its fitness on the test set (that we call test fitness) was recorded. The curves in the plots report the median of these values for the 30 runs. The median was preferred over the mean because of its higher robustness to outliers. For all the reported experiments, tests of statistical significance were performed. In particular, the Kolmogorov-Smirnov test has shown that, for all of our experiments, the data were not normally distributed and hence a rank-based statistic has been used. The Wilcoxon rank-sum test for pairwise data comparison with Bonferroni correction has been used under the alternative hypothesis that the samples do not have equal medians. Given the length limit imposed to this publication, we report all the obtained p-values as supplementary material available for download at:

http://kdbio.inesc-id.pt/~sara/gptp2013/SupplementaryMaterial.pdf

In the remainder of this manuscript, we will refer to results as *statistically significant* if the corresponding p-value is smaller than the usual significance level $\alpha = 0.01$. In the opposite case, results will be addressed as *not statistically significant*.

Results: Standard GP vs Geometric Semantic GP. Figure 11.3 shows the evolution of training and test fitness for ST-GP and GS-GP for all the studied problems. GS-GP outperforms ST-GP consistently on all the studied problems both on training and test data (with the only exception of the LD50 dataset, where ST-GP outperforms GS-GP on the training set, but is outperformed on the test set). Furthermore, it is very interesting to remark that, for every studied problem, the curve of GS-GP on the test set is quite regular, while, with the only exception of the PEC problem, the one of ST-GP contains several oscillations or worsening jumps. This clearly indicates that GS-GP is able to control overfitting much better than


a
Training Error (%F)
ST–GP
GS–GP
Number of Generations
b
Test Error (%F)
Number of Generations
c
Training Error (%PPB)
Number of Generations
d
Test Error (%PPB)
Number of Generations
e
Training Error (LD50)
Number of Generations
f
Test Error (LD50)
Number of Generations
g
Training Error (PEC)
Number of Generations
h
Test Error (PEC)
Number of Generations

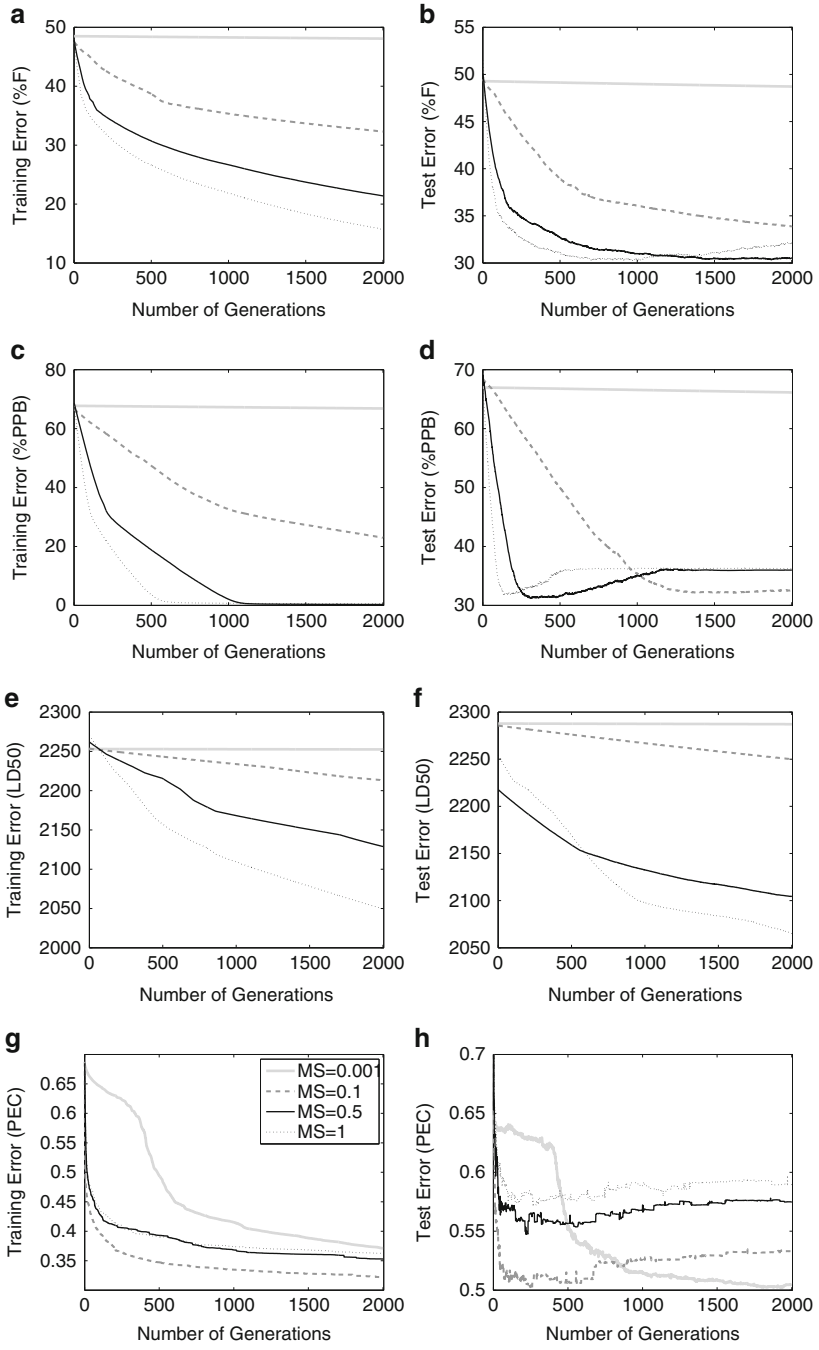
ST-GP, a tendency that was already remarked in Vanneschi et al. (2013) and that is discussed in the final part of this section. According to the Wilcoxon test, all the results discussed so far are statistically significant.

Results: GP with only Geometric Semantic Mutation. Geometric semantic crossover can find a globally optimal solution only if this solution is inside the hypercube delimited in the semantic space by the individuals in the initial population. Given that geometric semantic mutation does not have this limitation, it is worth verifying whether GS-GP really needs crossover or if it can reach the same performance using only mutation. In doing this, it is also interesting to study how the mutation rate (*ms* parameter) influences the performance. The results of this study are reported in Fig. 11.4, where, for each one of the considered applications, we show the results of four different variants of GP using only geometric semantic mutation. The names of the variants are "MS = 0.001", "MS = 0.1", "MS = 0.5" and "MS = 1", according to the *ms* value used. For the pharmacokinetic datasets, "MS = 1" outperforms the other variants both on training and test data (in a statistically significant way, according to the Wilcoxon test). The conclusion seems to be that high mutation steps are better, at least when geometric semantic mutation is used in isolation. However, the results obtained on the PEC dataset contradict this. In fact, on the PEC dataset "MS = 1" is the worst performer. In this problem the best performance is obtained by "MS = 0.1" and "MS = 0.001", with "MS = 0.1" achieving good results very quickly but with relatively high oscillations on the test set, and "MS = 0.001" behaving in a much steadier manner, achieving better results on the test set despite being slower to achieve them on the training set.

Our conclusion is that, although GP using only geometric semantic mutation seems to work well on all the studied datasets, also limiting overfitting in a comparable way to GS-GP, the setting of the *ms* parameter is still an issue. This parameter greatly influences the performance, and the ideal value seems to be problem dependent. For this reason, we have decided to study a GP variant in which both the rates of application of the genetic operators and the mutation step are not decided once "a priori", but instead are randomly generated.

Results: Random Generation of Operator Rates and Mutation Step. This GP variant is characterized by using random rates for crossover and mutation, and random mutation step. More in detail, at each generation, the crossover and mutation rates are randomly chosen with uniform probability in [0, 1] (with the only, obvious, constraint that the sum between the crossover, mutation and reproduction rates

Fig. 11.3 Experimental comparison between standard GP (ST-GP) and Geometric Semantic GP (GS-GP). (**a**): %F problem, results on the training set; (**b**): %F problem, results on the test set; (**c**): %PPB problem, results on the training set; (**d**): %PPB problem, results on the test set; (**e**): LD50 problem, results on the training set; (**f**): LD50 problem, results on the test set; (**g**): PEC problem, results on the training set; (**h**): PEC problem, results on the test set

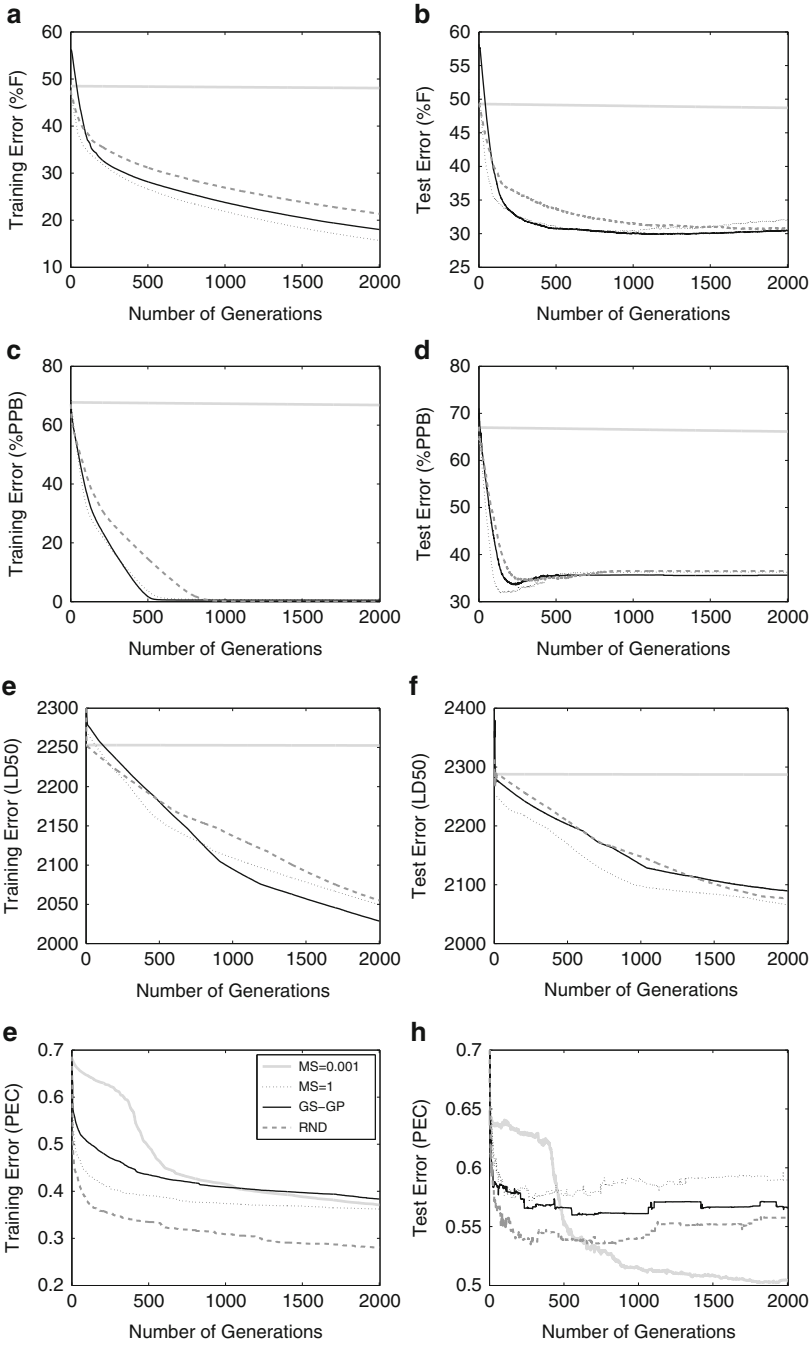

a
Training Error (%F)
Number of Generations
b
Test Error (%F)
Number of Generations
c
Training Error (%PPB)
Number of Generations
d
Test Error (%PPB)
Number of Generations
e
Training Error (LD50)
Number of Generations
f
Test Error (LD50)
Number of Generations
g
Training Error (PEC)
Number of Generations
MS=0.001
MS=0.1
MS=0.5
MS=1
h
Test Error (PEC)
Number of Generations

has to be equal to one). Hence, the probability of applying a specific genetic operator is randomly updated at each generation. As done for the operator rates, also the mutation step ms is generated uniformly at random in $[0, 1]$ but, differently from the operator rates, a new value for this parameter is generated each time the mutation has to be applied, right before the application itself. This random process should have a positive influence on the evolution, especially in later stages of the run, when improving fitness is harder. We identify the new GP variant as RND, and we report its results in Fig. 11.5, comparing them with the ones of GS-GP (exactly as in Fig. 11.3). To facilitate comparisons between all the studied variants, in each plot we also report the best and the worst curve of the corresponding plot in Fig. 11.4. Although debatable, we consider the best mutation step in the PEC problem to be 0.001. Interestingly, we can see that for the pharmacokinetic datasets, the performance of RND is always comparable with (i.e. extremely similar to) the best variant both on training and test data. Furthermore, for the PEC dataset RND is the best performer on training data and the second best performer on test data.

According to the Wilcoxon test, the differences between RND and the best variant are not statistically significant for any of the pharmacokinetic problems both on training and test sets. For the PEC problem, the advantage of RND compared to all the other methods on the training set is statistically significant, and the differences between RND and all the other variants are not statistically significant on the test set. This allows us to conclude that the performance of RND is statistically the same as the best variant for all the studied problems, even outperforming in the training data of the PEC problem. This confirms the suitability of the RND method, at least for the studied applications.

Discussion: Overfitting Control. An interpretation of the excellent performance on test data obtained using geometric semantic operators has already been given in Vanneschi et al. (2013). However, given its novelty and importance, we repeat and extend it here. The main point is: the geometric properties of those operators hold *independently from the data* on which individuals are evaluated, and thus they hold also on test data. In other words, geometric semantic crossover produces an offspring that stands between the parents also in the semantic space induced by test data. As a direct implication, following exactly the same argument as in Moraglio et al. (2012), the offspring is, in the worst case, not worse than the worst of its parents *also on test data*. This is exemplified in Fig. 11.1b where also a hypothetical

Fig. 11.4 Experimental comparison between variants of Geometric Semantic GP (GS-GP) with different mutation rates. (**a**): %F problem, results on the training set; (**b**): %F problem, results on the test set; (**c**): %PPB problem, results on the training set; (**d**): %PPB problem, results on the test set; (**e**): LD50 problem, results on the training set; (**f**): LD50 problem, results on the test set; (**g**): PEC problem, results on the training set; (**h**): PEC problem, results on the test set


a
Training Error (%F)
Number of Generations
b
Test Error (%F)
Number of Generations
c
Training Error (%PPB)
Number of Generations
d
Test Error (%PPB)
Number of Generations
e
Training Error (LD50)
Number of Generations
f
Test Error (LD50)
Number of Generations
e
Training Error (PEC)
MS=0.001
MS=1
GS–GP
RND
Number of Generations
h
Test Error (PEC)
Number of Generations

test point τ is shown[2] and where it is possible to see that offspring O is closer to τ than parent P_1. Let there be no misunderstanding: of course at learning time we do not know where test data are in the semantic space; what we are stating is that, wherever they are, the property holds, since it holds for each and every possible set of data. Regarding mutation, following the same reasoning used for training data, geometric semantic mutation produces an offspring that is a "weak" perturbation of its "parent" also in the semantic space induced by test data, given that this is true for every possible set of data (and the maximum possible perturbation can be expressed as a function of the mutation step ms).

These considerations have an immediate consequence on the behaviour of GS-GP on test data: even though geometric semantic operators do not guarantee a test fitness improvement at every application (and indeed in some cases among the ones discussed above the GS-GP median test error increases from one generation to the next), they at least guarantee that the eventual worsening of the test fitness is *limited*, and limited by an amount that is easily estimated: the test error of the worst parent for crossover and the ms parameter for mutation. In other words, geometric semantic operators are a means for *controlling overfitting*. This guarantees, if not a constant improvement in test error, at least that GS-GP cannot have those "big worsening jumps" in the test fitness plots, as for instance the ones that characterize ST-GP on the %PPB, %F and LD50 problems (Fig. 11.3). We finally point out that without the efficient implementation of the geometric semantic operators presented in Sect. 4 that has allowed us to use them on real life applications, this interesting property would probably have remained unnoticed.

6 Conclusions and Future Work

This chapter contains a deep investigation on the usefulness of geometric semantic operators for real life applications. The study was possible thanks to a new and efficient implementation of these operators that we have recently introduced. The first

Fig. 11.5 Experimental Comparison between Geometric Semantic GP (GS-GP) and a variant in which crossover rate, mutation rate and mutation step are randomly generated (RND). The best and worst curves from Fig. 11.4 are also reported for each plot, in order to facilitate comparisons. (**a**): %F problem, results on the training set; (**b**): %F problem, results on the test set; (**c**): %PPB problem, results on the training set; (**d**): %PPB problem, results on the test set; (**e**): LD50 problem, results on the training set; (**f**): LD50 problem, results on the test set; (**g**): PEC problem, results on the training set; (**h**): PEC problem, results on the test set

[2]Figure 11.1b makes an assumption: that the number of instances of the test set is identical to the one of the training set. Otherwise, the training target T and the test point τ could not be drawn in the same plane, because they would have different dimensions. This hypothesis is false in general, but it is used for simplicity, since it helps us to explain more clearly our hypothesis. Nevertheless, it is worth pointing out that, of course, the argument holds also when this restrictive assumption is false.

result we achieved is a clear confirmation of the fact that Genetic Programming (GP) using these operators (Geometric Semantic Genetic Programming, GS-GP) outperforms standard GP (ST-GP) on training data, which was expected because these operators induce a unimodal error surface. The second one, slightly less expected in the beginning, was that GS-GP also significantly outperforms ST-GP on unseen test data. Investigating the geometric properties of the operators on the semantic space, we have been able to explain why these operators are a concrete mechanism to limit overfitting. Our third achievement was the evidence that GS-GP with no crossover (i.e. using only geometric semantic mutation) can obtain the same performance of, and in some cases even outperform, GS-GP using both crossover and mutation. However, this is true only for some particular values of the mutation step (a parameter that is in the definition of geometric semantic mutation), and the appropriate values for this parameter are problem dependent. Finally, the fourth and last presented achievement was the definition of a new GS-GP variant (called RND), in which crossover rate, mutation rate and mutation step are randomly updated several times during the evolution. The reported experiments show that RND is able to obtain a performance that most of the times is comparable to the performance of the best parameter setting both on training and test data.

Our experiments have been conducted on three problems of pharmacokinetics and one problem of energy consumption forecasting. These problems are important in their respective applicative areas, and one of our current activities consists in interacting with domain experts to investigate and further interpret the solutions produced by GS-GP. Of course, the fact that GS-GP produces either black-box models of data, or models which are quite complex and hard to generate, represents its major limitation. Thus, we are also investigating the possibility of defining new operators that, while respecting the same geometric properties as the ones studied here on the semantic space, do not necessarily generate offspring that are larger than their parents. We hope that these operators, once they are well defined, will produce models that are clearer and easier to understand, possibly with the same quality and using the same amount of computational resources as GS-GP.

Acknowledgements This work was supported by national funds through FCT under contract PEst-OE/EEI/LA0021/2013 and by projects EnviGP (PTDC/EIA-CCO/103363/2008), MassGP (PTDC/EEI-CTP/2975/2012) and InteleGen (PTDC/DTP-FTO/1747/2012), Portugal.

References

Archetti F, Lanzeni S, Messina E, Vanneschi L (2007) Genetic programming for computational pharmacokinetics in drug discovery and development. Genet Program Evolvable Mach 8:413–432

Beadle L, Johnson C (2008) Semantically driven crossover in genetic programming. In: Wang J (ed) Proceedings of the IEEE world congress on computational intelligence, Hong Kong. IEEE Computational Intelligence Society/IEEE, pp 111–116. doi:10.1109/CEC.2008.4630784

Beadle L, Johnson CG (2009) Semantically driven mutation in genetic programming. In: Tyrrell A (ed) 2009 IEEE congress on evolutionary computation, Trondheim. IEEE Computational Intelligence Society/IEEE, pp 1336–1342. doi:10.1109/CEC.2009.4983099

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Krawiec K (2012) Medial crossovers for genetic programming. In: Moraglio A, Silva S, Krawiec K, Machado P, Cotta C (eds) Proceedings of the 15th European conference on genetic programming, EuroGP 2012, Malaga. Lecture notes in computer science, vol 7244. Springer, pp 61–72. doi:10.1007/978-3-642-29139-5-6

McPhee NF, Ohs B, Hutchison T (2008) Semantic building blocks in genetic programming. In: Proceedings of the 11th European conference on genetic programming, EuroGP'08, Naples. Springer, Berlin/Heidelberg, pp 134–145

Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: Parallel problem solving from nature, PPSN XII (part 1), Taormina. Lecture notes in computer science, vol 7491. Springer, pp 21–31

Nguyen QU, Nguyen XH, O'Neill M (2009a) Semantic aware crossover for genetic programming: the case for real-valued function regression. In: Vanneschi L, Gustafson S, Moraglio A, De Falco I, Ebner M (eds) Proceedings of the 12th European conference on genetic programming, EuroGP 2009, Tuebingen. Lecture notes in computer science, vol 5481. Springer, pp 292–302. doi:10.1007/978-3-642-01181-8-25

Nguyen QU, Nguyen XH, O'Neill M (2009b) Semantics based mutation in genetic programming: the case for real-valued symbolic regression. In: Matousek R, Nolle L (eds) 15th international conference on soft computing, Mendel'09, Brno, pp 73–91

Uy NQ, Hoai NX, O'Neill M, McKay B (2010) The role of syntactic and semantic locality of crossover in genetic programming. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G (eds) 11th international conference on parallel problem solving from nature, PPSN 2010, Krakow. Lecture notes in computer science, vol 6239. Springer, pp 533–542. doi:10.1007/978-3-642-15871-1-54

Uy NQ, Hoai NX, O'Neill M, McKay RI, Galvan-Lopez E (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet Program Evolvable Mach 12(2):91–119. doi:10.1007/s10710-010-9121-2

Vanneschi L, Castelli M, Manzoni L, Silva S (2013) A new implementation of geometric semantic GP applied to predicting pharmacokinetic parameters. In: Proceedings of the 16th European conference on genetic programming, EuroGP'13, Vienna. Springer, pp 205–216

Chapter 12
Evaluation of Parameter Contribution to Neural Network Size and Fitness in ATHENA for Genetic Analysis

Ruowang Li, Emily R. Holzinger, Scott M. Dudek, and Marylyn D. Ritchie

Abstract The vast amount of available genomics data provides us an unprecedented ability to survey the entire genome and search for the genetic determinants of complex diseases. Until now, Genome-wide association studies have been the predominant method to associate DNA variations to disease traits. GWAS have successfully uncovered many genetic variants associated with complex diseases when the effect loci are strongly associated with the trait. However, methods for studying interaction effects among multiple loci are still lacking. Established machine learning methods such as the grammatical evolution neural networks (GENN) can be adapted to help us uncover the missing interaction effects that are not captured by GWAS studies. We used an implementation of GENN distributed in the software package ATHENA (Analysis Tool for Heritable and Environmental Network Associations) to investigate the effects of multiple GENN parameters and data noise levels on model detection and network structure. We concluded that the models produced by GENN were greatly affected by algorithm parameters and data noise levels. We also produced complex, multi-layer networks that were not produced in the previous study. In summary, GENN can produce complex, multi-layered networks when the data require it for higher fitness and when the parameter settings allow for a wide search of the complex model space.

Keywords Grammatical evolution • Neural networks • Data mining • Human genetics • Systems biology • XOR model

R. Li (✉) • E.R. Holzinger • S.M. Dudek • M.D. Ritchie
Center for Systems Genomics, Pennsylvania State University, University Park, PA, USA
e-mail: rvl5032@psu.edu; emily.r.holzinger@vanderbilt.edu; sud23@psu.edu; marylyn.ritchie@psu.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7_12,

1 Introduction

With the rapid advancement of the genomics field, huge amounts of biological data are being generated. One beneficiary of the technology advancement is the exponential growth of genotype data, which measures single nucleotide variations (SNVs) across the genome. In fact, recent assays provide up to five-million SNVs on every person for relatively low cost. The most widely used method for analyzing the genotyping data has been the genome-wide association study (GWAS). GWAS employs a statistical test at each individual locus across the genome to determine whether the locus is associated with the outcome (phenotype). GWAS has successfully revealed the genetic determinants for many complex human diseases (Hindorff et al. 2009). However, analysis of the genotyping data has been a great challenge because of the complex underlying relationships that exist in the data and many of the current GWAS analyses do not test the effect of interactions among multiple loci on the phenotype. Performing an exhaustive search of all possible combinations of loci is also not feasible with the current computational power, as the combinatorics for five-million SNVs explodes. To mine the missing genetic variations of the data, we utilized and modified machine-learning techniques to search for interactions among genetic factors (Motsinger-Reif et al. 2001; Breiman 2001). We developed the ATHENA package, which utilizes grammatical evolution neural networks (GENN), to uncover the genetic network models underlying the disease or phenotype.

Previously, we reported that the final network models produced by grammatical evolution neural networks (GENN) with three variables were simple, 1-layer neural networks (Ritchie et al. 2012). Various parameter optimizations were attempted, but yielded little change in the depth of the neural networks. While 1-layer networks decrease the probability of finding false positives, it also raises the question of whether GENN can successfully model more complex interactions that are typically found in genomics data. Theoretically, GENN models should be quite capable of building multilayer network structures, but empiric evidence based on Ritchie et al. (2012) led to concern. Specifically, three hypotheses were considered: (1) the GENN grammar is biased toward 1-layer networks, (2) the parameters selected for the GENN analysis were not capable of building multi-layer networks, (3) the data simulation models did not warrant multi-layer networks.

The goal of this study is to examine these hypotheses and determine if one or more of them are true. To further examine these ideas, we assessed factors that may influence the network structure and true model detection. We used two types of simulated effect models to investigate the effect of grammar, population size, number of generations, and maximum grammar tree depth on network structure and detection power. Our findings suggest that the previously reported bias towards 1-layer network size was not due to a limitation of the GENN algorithm or an error in the program. Rather, combinations of grammar and maximum network depth affect network structure and detection power. Furthermore, the amount of

noise (or non-informative variables) in the dataset also plays a role in networks identified by GENN. As such, it was a combination of parameter optimization and data simulation models that led to the results in Ritchie et al. (2012). In this chapter, we will explain the compendium of simulations performed to address this question, demonstrate the results, and discuss future directions.

Grammatical Evolution Neural Networks

Neural networks (NNs) were designed to imitate neurons in the brain so that the networks can process information in parallel. NNs are widely used data mining methods in scientific research to detect underlying models in data to predict the desired outcome. NNs consist of nodes that can receive inputs from other nodes or from external independent variables. Each input is associated with a coefficient (or weight) which is multiplied and then the NN processes the weighted inputs through some activation function to produce an output signal (Skapura 1995). Generally, the most popular method for training feed-forward multilayer NN; is the gradient descent algorithm back-propagation (BPNN). BPNN randomly initializes weights associated with each node and gradually adjusts the weights with the goal of minimizing an error function (Skapura 1995). However, if the underlying fitness landscape is unknown, BPNN is an insufficient optimization method. In genomics studies of complex diseases, the fitness landscape is always unknown and complex. Thus, in order to avoid defining the fitness landscape a priori, a method has been proposed to apply genetic programming to optimize the structures and weights of the NN (Koza and Rice 1991). A version of genetic programming neural networks (GPNN) has been implemented specifically for genetic association studies (Motsinger et al. 2006).

Grammatical evolution neural networks (GENN), an extension of GPNN, uses GE as the evolutionary algorithm. GE is a type of genetic programming (O'Neill and Ryan 2001, 2003) that uses Backus-Naur Form (BNF) grammar to create a model based on a genetic algorithm. The grammar translates an array of bits into a model, e.g. NN, based on its set of rules. At each generation, the fitnesses of the NNs are evaluated and the fittest networks are more likely to be selected to reproduce in the following generation. The genetic algorithm evolves for a specified number of generations and outputs the most optimal solution in the final generation. GENN optimizes variable selections, node coefficients, the number of hidden layers, and the number of nodes per layer simultaneously (Turner et al. 2010), so it can be applied to any data sets regardless of their underlying fitness landscape.

2 Methods

Data Simulation

We simulated the XOR model because it is commonly used as a benchmark for neural network models; in addition, it also describes a potential type of epistasis interaction that may be observed in biological data. There are also several bioengineering and biochemical applications that have utilized the XOR relationship in designing the experiment (Pearson et al. 2011; Privman et al. 2010). In addition, in the XOR model, neither of the two predictor variables have a main effect; rather, interactions between the variables determine the outcome. Thus, the model cannot be solved with 1-layer additive neural networks. Lastly, the result of experimenting with the XOR model can be extended to genotyping data because of the similarities of data formats. We simulated datasets under the XOR model with two possible outcomes (such as case and control). The model is detailed in Table 12.1. Random variables were generated using genomeSIMLA (Edwards et al. 2008) such that both outcomes contain 1,000 individuals (for a total of 2,000 individuals in the dataset).

In order to detect the effect of noise on network structure and detection power, we simulated two different types of datasets. The first type consisted of only two predictor variables with no noise variables (xor). The second type consisted of 100 predictor variables – two functional variables and 98 non-functional, or noise, variables (xor+noise). For both types of datasets, the two functional variables perfectly predicted the binary outcome using the XOR model shown in Table 12.1. While this is unrealistic for complex trait epistasis in biology, it gives us a clean benchmark to explore these hypotheses.

ATHENA

The Analysis Tool for Heritable and Environmental Network Associations (ATHENA) is a versatile software package that includes various analysis techniques. One of the modeling methods in ATHENA is GENN, which uses grammatical evolution to optimize artificial neural networks (ANNs). The GENN algorithm has previously been described in detail (Ritchie et al. 2012). The algorithm is briefly described as follows:

Table 12.1 Description of the XOR model

Phenotype	VARIABLE1	VARIABLE2
0	0	0
1	1	0
1	0	1
0	1	1

Step 1: The data are equally divided into 5 parts with 4/5 for training and 1/5 for testing. Different non-overlapping training and testing data are used for 5-fold cross validations.

Step 2: Under population size constraint, a random population of binary strings are generated to be ANNs using a Backus-Naur grammar. The ANNs are guaranteed to be functional per sensible initialization (O'Neill and Ryan 2001, 2003). During sensible initialization, an expression tree is created using the specified grammar by randomly selecting grammar rules to construct the tree. The software recursively checks the expression tree to make sure the selected rule would not make the expression tree exceed the maximum depth (Maxdepth) allowed. Half of the expression trees are built to the maximum depth, and the other half are built with a random depth less than the maximum depth. Finally, the expression trees are converted into corresponding codons. This step concurrently occurs at all demes (computer CPUs).

Step 3: All ANNs are evaluated with training data, and the solutions with highest balanced accuracies are selected for crossover and reproduction. The new population is composed of mutated original solutions and new random solutions.

Step 4: Step 3 is repeated until it reaches the set generation number. Migrations of the best solutions occur at specified intervals between CPUs.

Step 5: The best solution at the final generation is tested on the testing data, and the balanced accuracy is recorded.

Step 6: Steps 2–5 are repeated each time with a different set of training and testing data.

We ran the GENN algorithm within ATHENA using the parameters settings in Table 12.2.

The parameters for generations, population size, and maxdepth are selected through empirical studies. Different grammar sets are designed to test the effects of different search spaces: add (linear), bool2 (linear, logical), bool, (linear, logical, multiplicative). Each combination of the values for the varying parameters (36 unique combinations) was assessed using 10 xor datasets and 10 xor+noise datasets for a total of 720 experiments.

3 Results

Maximum depth of the grammar tree (Maxdepth) specifies the maximum number of layers of the grammar tree prior to translating into a neural network. A grammar tree consists of grammar expressions that direct how to make the neural network. Higher Maxdepth would more likely produce a more complex grammar tree, which in turn correlates with the higher complexity of the neural networks. With a Maxdepth of 9, the average accuracies of models for both types of datasets were approximately 75–85% with an average neural network depth of 1. When the Maxdepth was 12, the average accuracies for the xor datasets reached 100%, indicating that the maximum

Table 12.2 ATHENA GENN parameters

	Parameter	Values	Description
	Demes	16	Number of demes. Evolution occurs within each deme on an individual node with scheduled migration between demes
Constant parameters	Migrations	Every 25 generations	Networks with the highest fitness migrate between demes
	Probability of cross-over	0.9	Probability of exchanging genetic material between solutions
	Probability of mutation	0.01	Probability of random mutation
	Fitness metric	Balanced accuracy	(Sensitivity + specificity)/2
	Selection	Roulette	The probability of selection is proportional to its fitness
	Min genome size	20	Minimum size of binary genome at initialization
	Max genome size	15,000	Maximum size of binary genome at initialization
	Backpropagation	Every 25 generations	Frequency of back propagation
	Cross-validation	5 fold	4/5th of the data was used as training and 1/5th of the data was used as testing
Varying parameters	Grammar	Bool	Operators: {+, −, ∗, /, OR, NOR, AND, NAND}
		Bool2	Operators: {+, −, OR, NOR, AND, NAND}
		Add	Operators: {+, −}
	Generations	300, 550, 800	Number of generations
	Deme population size	1,000, 2,000	The number of solutions per deme
	Maxdepth	9, 12	Maximum depth of grammar tree
Datasets	Dataset type	xor, xor+noise	Type of dataset used to test the effect of the varying parameters

depth of the grammar tree was correlated with model detection. Higher Maxdepth also allowed production of multilayer neural networks (Fig. 12.1).

GENN grammar specifies how the nodes connect to each other, e.g. multiplying grammar multiplies the inputs and then feeds it into the activation function. We compared three sets of grammar in this study: Bool {+, −, ∗, /, OR, NOR, AND, NAND}, bool2 {+, −, OR, NOR, AND, NAND}, and add {+, −}. Upon random initializing of the population, each individual was generated based on its grammar set. Thus, individuals had access to different building blocks when they have different grammars. Models built with bool2 grammar achieved higher accuracy

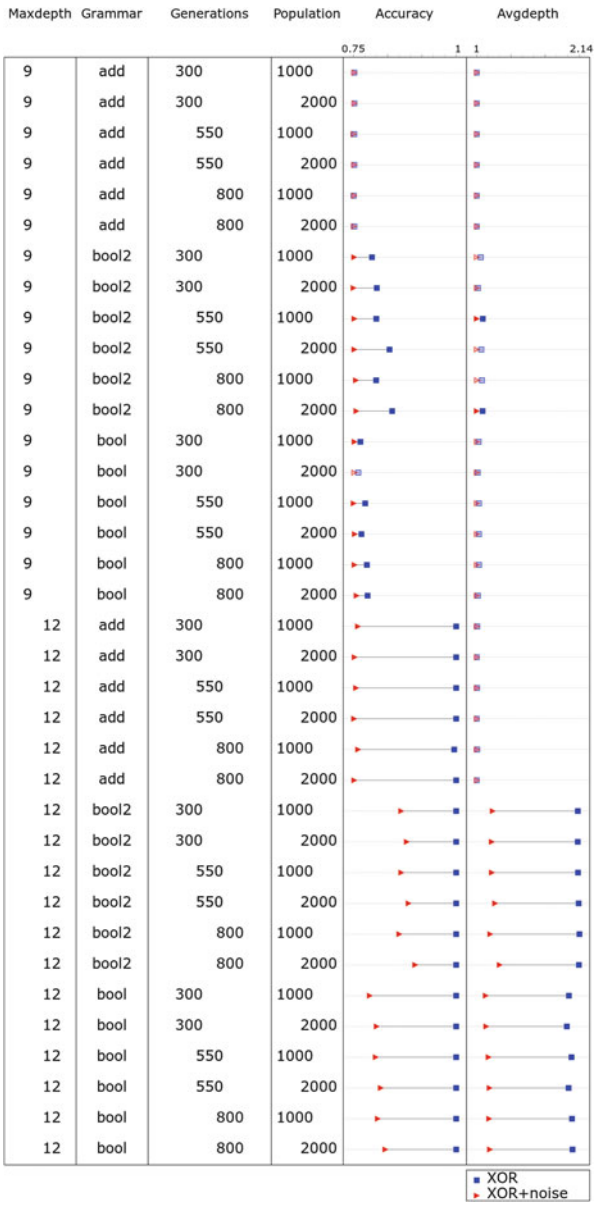


Fig. 12.1 Average accuracy and average depth of neural networks

than that of bool and add grammar. Using bool2 or bool grammar in conjunction with a Maxdepth of 12 allowed the production of multi-layer neural networks (Fig. 12.1).

The noise level of the datasets fell into two distinct groups. One group of data only contained two functional variables (xor), which can perfectly predict the simulated XOR model. The other group contained the same two variables and 98 randomly generated noise variables (xor+noise). The differences between the two

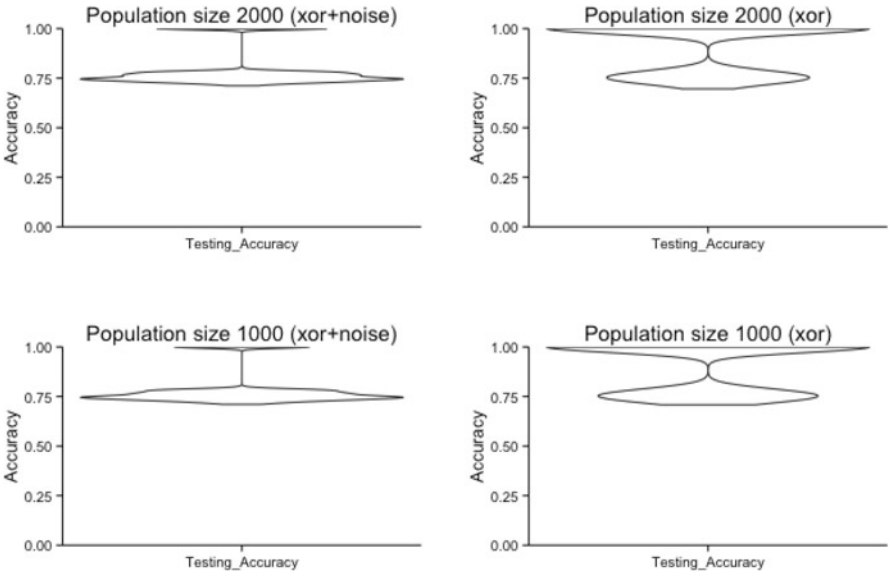


Fig. 12.2 Comparison of population sizes on model detection

groups are clear as shown in Fig. 12.1. In all cases, data sets without noise signals achieved equal or higher prediction accuracies and generated more complex neural networks than xor+noise data sets.

To further understand the effect of each GENN parameter on model detection, we compared the accuracies for each set of GENN parameters on two types of models. Xor data set produced more accurate models than that of the xor+noise data sets regardless of the population size. Also, there was no clear difference between population size of 1,000 and 2,000 on model detection (Fig. 12.2). Similarly with population size, longer generations of evolution did not affect model detection. The prediction accuracies were only affected by data noise level (Fig. 12.3). The maximum depth of the grammar tree significantly improved model detection for both types of data. In particular, under Maxdepth of 12, xor data achieved perfect predictability in almost all data sets (Fig. 12.4). In the order of add grammar, bool grammar, and bool2 grammar, model detection has shown an improvement in both data types. This is evident by the increasingly higher number of models at 100% accuracy in the above order (Fig. 12.5).

From these results, we drew three main conclusions. First, the addition of noise variables results in overall lower testing accuracy, which is not surprising and in some ways serves as a positive control experiment. Second, higher maximum grammar tree depth resulted in more predictive models for both dataset types. Third, grammar types have an effect on accurate modeling for both dataset types.

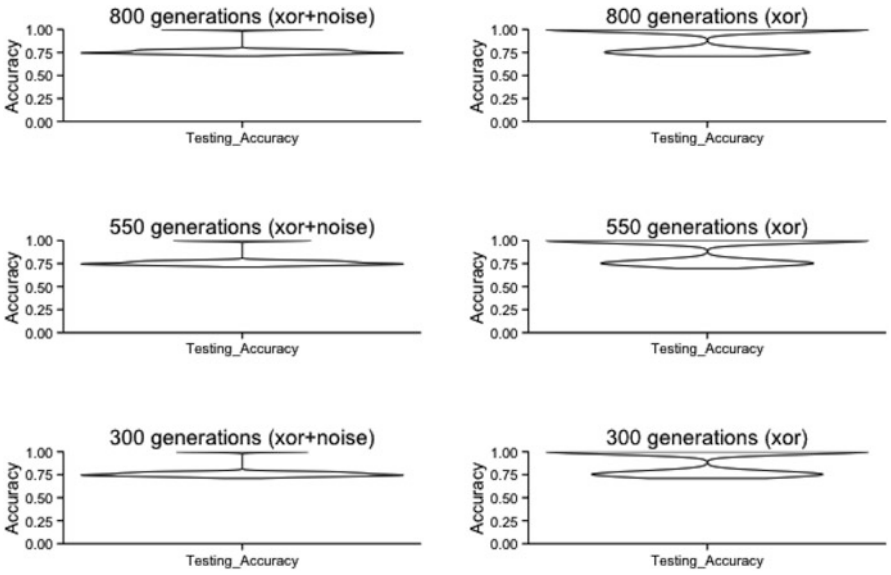


Fig. 12.3 Comparison of generations on model detection

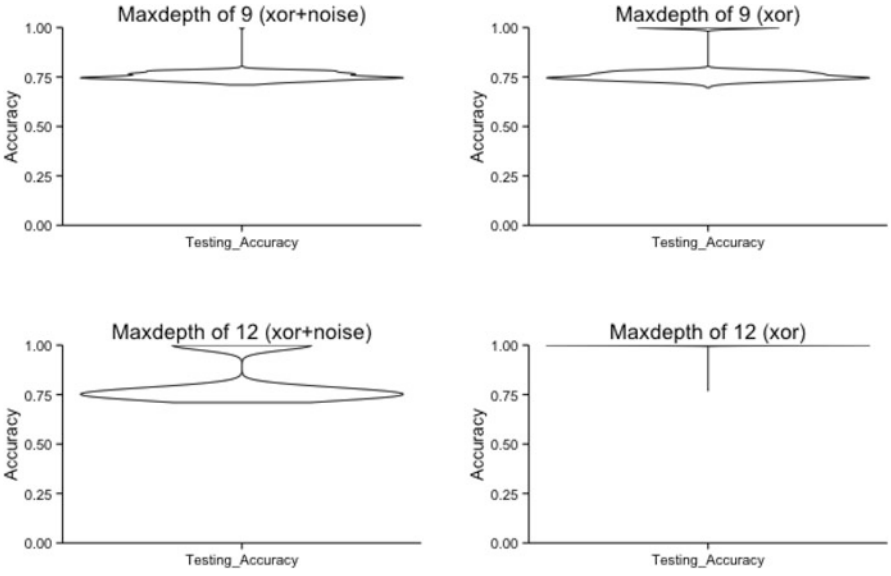


Fig. 12.4 Comparison of maximum depth of grammar tree on model detection

To further investigate the factors that influence network size, we studied the effect of each parameter across generations. The average depths of the neural network for xor datasets were higher than that of xor+noise datasets as shown by darker

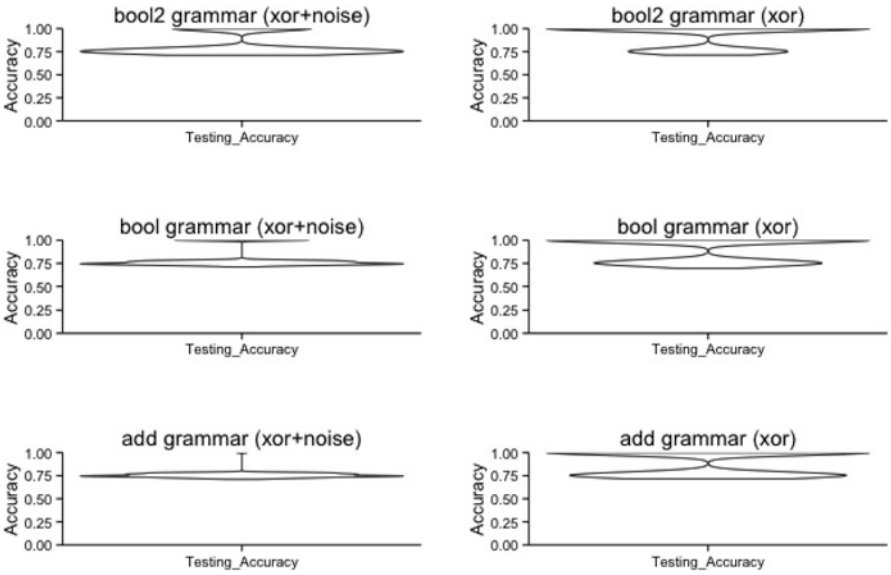


Fig. 12.5 Comparison of grammars on model detection

shades at the top portion of the panels. However, there was no clear difference of network depth between the two population sizes (Fig. 12.6). Similarly, xor dataset produced more complex neural networks than xor+noise dataset regardless of the number of generations. The depth of the network stayed relatively flat after the initial oscillation (Fig. 12.7). The maximum depth of the grammar tree had a clear effect on the network structure. With a Maxdepth of 9, GENN produced mostly single layer neural networks. Increasing the Maxdepth to 12 resulted in more complex neural networks (Fig. 12.8). With add grammar, neither data set produced complex neural networks. Add grammar also differs from the other two grammars in that the depth drops very quickly and never recovered. Both bool and bool2 grammar were able to produce multilayer neural networks (Fig. 12.9).

As previously reported, the average depth of the networks decreased at the beginning of the evolution. Except for experiments where the maximum depth was 9 or with add grammar, the average neural network depth increased after the initial drop. Datasets without noise variables generally had higher average network depth, indicating that the noise level was a strong determinant of network depth. Higher maximum depth, bool and bool2 grammar were also correlated with higher average network depths.

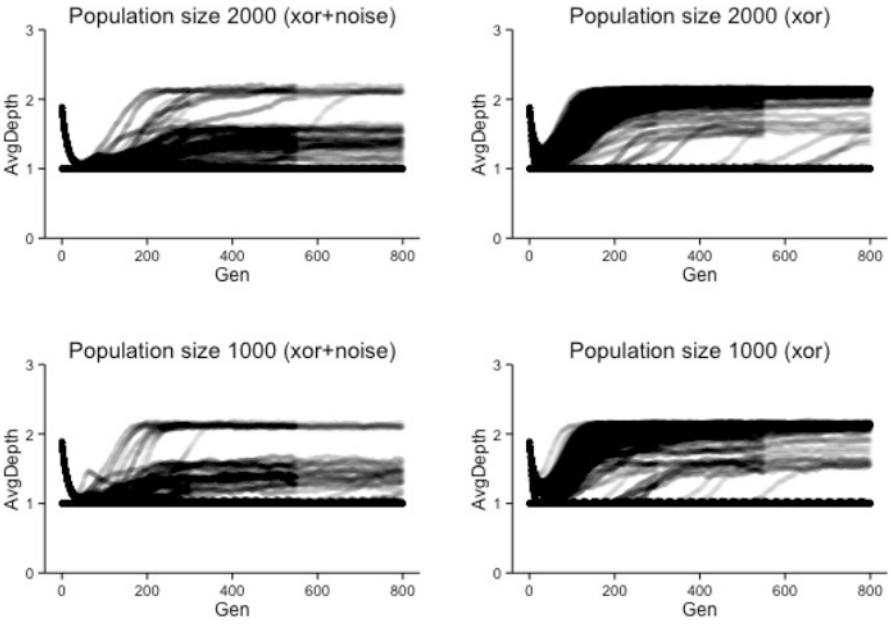


Fig. 12.6 Comparison of population sizes on average network sizes

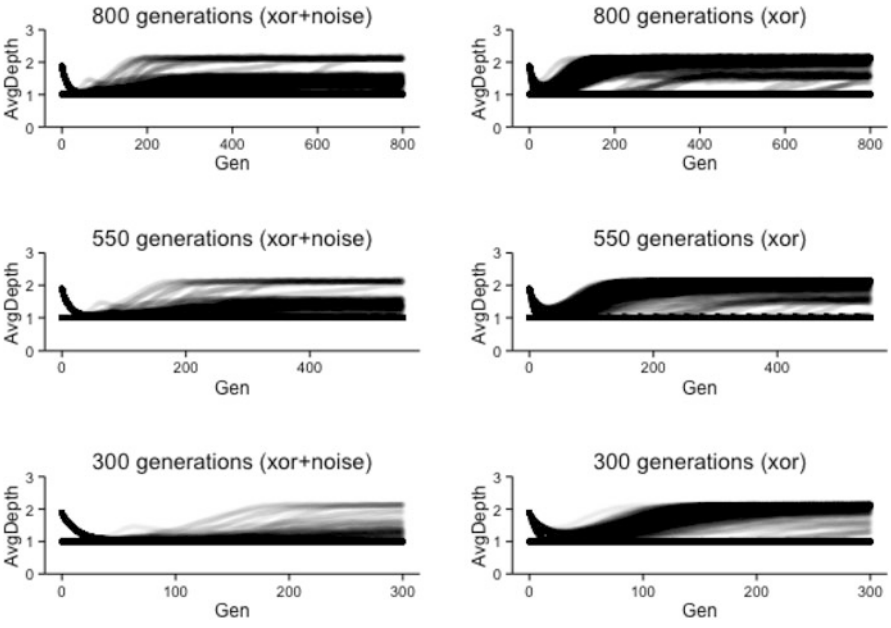


Fig. 12.7 Comparison of generations on average network sizes

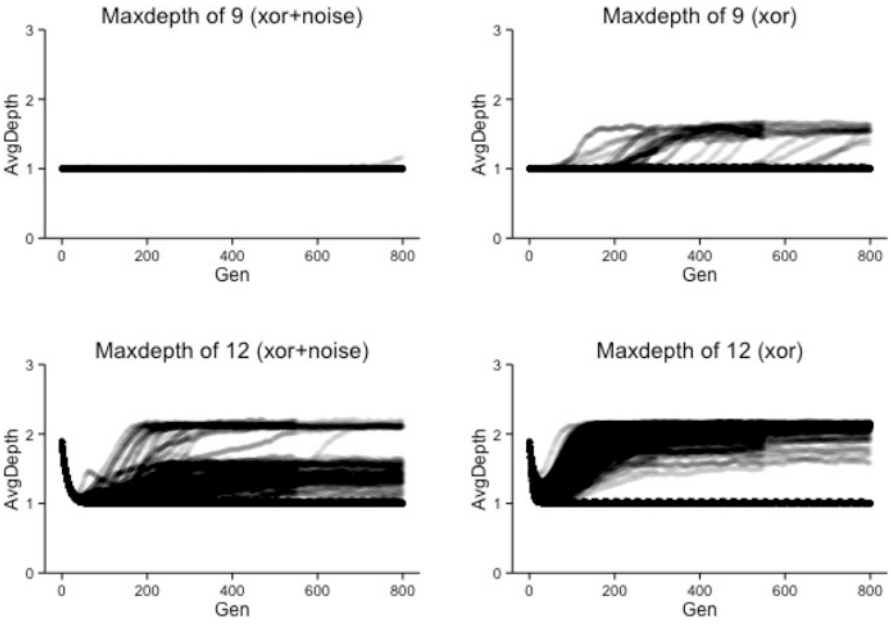


Fig. 12.8 Comparison of maximum depth of the grammar tree on average network sizes

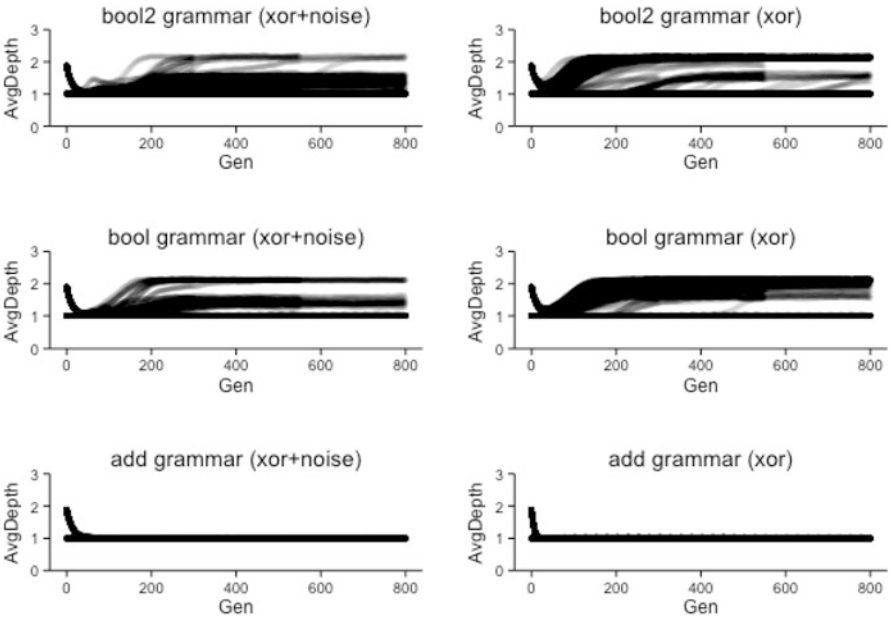


Fig. 12.9 Comparison of grammars on average network sizes

4 Discussion

In the previous report (Ritchie et al. 2012), we observed that GENN preferentially produced 1-layer neural networks based on the three variables simulated in the model. The reason for producing 1-layer networks remained unclear as to whether it was due to the inherited characteristics of GENN, the GENN parameter settings, or the complexity of the simulated datasets. While simple 1-layer networks offer the advantage of better interpretability, it might not be sufficient to model complex biological processes because there are non-additive epistasis effects in many biological systems (Gibson et al. 2004; Andrew et al. 2012). To be sure that GENN can produce a multilayer network if it is required by data, we experimented with GENN parameters to understand under which parameter combinations obtain multilayer networks. We believe that parameter settings should not favor either 1-layer networks or multilayer networks because the evolution process will determine the most fit model structures. However, when the parameter settings are limiting or unnecessarily expanding the search space, it will have a great impact on the generated models. Most importantly, our goal is to apply GENN on biological data with unknown underlying relationships. Through this experimentation, we can eliminate biases due to parameter settings and have more confidence in our results. In this study, we simulated two types of xor models: xor and xor+noise to test the effect of various GENN parameters on model detection and network structures. These results have shown that the noise level in a dataset was the most significant determinant of model detection and network depth. In noisy data, GENN has to perform both variable selection and network modeling as opposed to only network modeling in the noiseless data. The existence of the noise variables decreased the probability of true variable detection, which in turn made it more difficult to maintain multilayer networks through evolution because most of the variables are non-informative. Maximum depth of the grammar tree was also an important factor in model detection. In our simulations, lower Maxdepth limited the possibility of producing multilayer networks, which in turn limited the ability of GENN to produce multilayer network structures necessary to detect interaction effects. Lastly, different grammars also affected model detection and network depth. Bool and bool2 grammar outperformed add grammar because they included more operators that can detect interaction relationships among variables. However, having more operators did not produce the best model as evidenced by the better modeling with bool2 grammar compared to bool grammar. If the additional operators did not add more informative variable relationships, it will unnecessarily increase the search space, which could explain the lower accuracies produced by bool grammar. In conclusion, we have determined that GENN does not inherently produce 1-layer networks. The combination of noise level, maximum grammar tree depth and grammar determines the model detection and network sizes in GENN. Due to the simplicity of the XOR model, the highest average NNs depth is only around 2.

In future experimentation, we can simulate more complex models and use higher Maxdepth value. Software optimization might be needed because higher Maxdepth significantly requires more computational resources during the evolution process.

References

Andrew AS, Hu T, Gu J, Gui J, Ye Y, Marsit CJ, Kelsey KT, Schned AR, Tanyos SA, Pendleton EM, Mason RA, Morlock EV, Zens MS, Li Z, Moore JH, Wu X, Karagas MR (2012) HSD3B and gene-gene interactions in a pathway-based analysis of genetic susceptibility to bladder cancer. PLoS One 7(12):e51301

Breiman L (2001) Random forests. Mach Learn 45:5–32

Edwards T, Bush W, Turner S, Dudek S, Torstenson E, Schmidt M, Martin E, Ritchie M (2008) Generating linkage disequilibrium patterns in data simulations using genomesimla. Evol Comput Mach Learn Data Min Bioinform 4973:24–35

Gibson G, Riley-Berger R, Harshman L, Kopp A, Vacha S, Nuzhdin S, Wayne M (2004) Extensive sex-specific nonadditivity of gene expression in drosophila melanogaster. Genetics 167:1791–1799. 104026583

Hindorff LA, Sethupathy P, Junkins HA, Ramos EM, Mehta JP, Collins FS, Manolio T (2009) Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. Proc Natl Acad Sci USA 106:9362–9367

Koza JR, Rice JP (1991) Genetic generation of both the weights and architecture for a neural network. In: International joint conference on neural networks, IJCNN-91, Washington State Convention and Trade Center, Seattle, vol II. IEEE Computer Society, pp 397–404

Motsinger AA, Lee SL, Mellick G, Ritchie MD (2006) GPNN: power studies and applications of a neural network method for detecting gene-gene interactions in studies of human disease. BMC Bioinform [electronic resource] 7(1):39–39

Motsinger-Reif AA, Dudek SM, Hahn LW, Ritchie MD (2001) Comparison of approaches for machine-learning optimization of neural networks for detecting gene-gene interactions in genetic epidemiology. Genet Epidemiol 32:325–340

O'Neill M, Ryan C (2001) Grammatical evolution. IEEE Trans Evol Comput 5(4):349–358

O'Neill M, Ryan C (2003) Grammatical evolution: evolutionary automatic programming in a arbitrary language. Volume 4 of genetic programming. Kluwer, Boston

Pearson B, Lau K, Allen A, Barron J, Cool R, Davis K, DeLoache W, Feeney E, Gordon A, Igo J, Lewis A, Muscalino K, Madeline P, Penumetcha P, Rinker V, Roland K, Zhu X, Poet J, Eckdahl T, Heyer L, Campbell A (2011) Bacterial hash function using DNA-based XOR logic reveals unexpected behavior of the LuxR promoter. IBC 3, article no 10:1–10

Privman V, Zhou J, Halamek J, Katz E (2010) Realization and properties of biochemical-computing biocatalytic XOR gate based on signal change. J Phys Chem B 114:13601–13608

Ritchie MD, Holzinger ER, Dudek SM, Frase AT, Chalise P, Fridley B (2012) Meta-dimensional analysis of phenotypes using the Analysis Tool for Heritable and Environmental Network Associations (ATHENA): challenges with building large networks. In: Riolo R et al (eds) Genetic programming theory and practice X. Springer, New York

Skapura D (1995) Building neural networks. ACM, New York

Turner SD, Dudek SM, Ritchie MD (2010) Grammatical evolution of neural networks for discovering epistasis among quantitative trait loci. In: Pizzuti C, Ritchie MD, Giacobini M (eds) 8th European conference on evolutionary computation, machine learning and data mining in bioinformatics (EvoBIO 2010), Istanbul. Volume 6023 of lecture notes in computer science, pp 86–97. Springer

Index

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, DOI 10.1007/978-1-4939-0375-7,