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

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

Mark Kotanchek
Evolved Analytics
Midland, MI, 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 **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, ..., x_{M-1} >: such that

- (*E1*) y = γ(x)=c_0+$\Sigma c_i B_i$(x)+**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 $\text{square}(x_{10}+3.427)$. Any trial champion such as $\text{cube}(x_{10}+c_0)$ 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 $\text{cube}(x_{10}+c_0)$ 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 $\|\mathrm{F}\| = 15$, the number of features $\|\mathrm{V}\| = 100$, and *cbit*=18. The size of *universal(3,1,t)* can be computed with the following formula $\|\mathrm{F}\|^7 * (\|\mathrm{V}\| + 2^{cbit})^8$. Therefore $15^7 * (100 + 2^{18})^8 . = 3.82\mathrm{E}{+}51$ 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.