Genetic and Evolutionary Computation

Rick Riolo
William P. Worzel
Mark Kotanchek *Editors*

Genetic Programming Theory and Practice XII

Springer

Genetic and Evolutionary Computation

Series Editors

David E. Goldberg
John R. Koza

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

Rick Riolo • William P. Worzel • Mark Kotanchek
Editors

Genetic Programming Theory and Practice XII

Springer

Editors
Rick Riolo
Center for the Study of Complex Systems (CSCS)
University of Michigan
Ann Arbor
Michigan
USA

Mark Kotanchek
Evolved Analytics
Midland
Michigan
USA

William P. Worzel
Evolution Enterprises
Milan
Michigan
USA

ISSN 1932-0167
Genetic and Evolutionary Computation
ISBN 978-3-319-16029-0 ISBN 978-3-319-16030-6 (eBook)
DOI 10.1007/978-3-319-16030-6

Library of Congress Control Number: 2015936144

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

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

Preface

This book is based on the material presented at the Twelfth Workshop on Genetic Programming Theory and Practice by the Center for the Study of Complex System at the University of Michigan in Ann Arbor on May 8th–10th, 2014. The purpose of this workshop is to promote the exchange of ideas between theorists working on genetic programming and people applying genetic programming to real-world problems. It is designed to encourage speculative presentations with a lot of time for discussion between presentations and focuses on the underlying principles of Genetic Programming (GP) and methods of application used to get the best results. Each chapter of this book was sent to two other participants *before* the workshop who responded with comments and suggestions, thus allowing the authors to revise and expand the work presented at the workshop. After the workshop, the authors revised the work based on comments and discussions from the participants. In many cases the involvement of reviewers, and the general discussions during workshop is considered the most important aspect of the the workshop.

In addition to the GP researchers and practitioners, there were three keynote speakers who presented on a related area of study at the start each day. Traditionally, these have involved a biological component as GP is inspired by the mechanisms of natural selection, a presentation on a related field of computer science, often related to machine learning, and a presentation by someone from industry describing the use of a cutting edge technology in a practical application. This year, on the first day, Dr. Chao Cheng of Dartmouth University presented *Application of Machine-Learning Methods to Transcriptional Regulation by Histone Modifications and Transcription Factors* where he discussed his group's use of machine learning and genetic programming in the analysis of molecular biology, and in particular, the mechanisms of genetics. On the second day, Paco Nathan presented *Nine Decades of Machine Learning: GP in the context of Big Data and contemporary open source* which described the history of machine learning, its growing importance in the analysis of 'Big Data' and the potential for GP in this space. Finally, Theresa Kotanchek presented *Materials Innovation and the Next Manufacturing Renaissance*, a description of her tenure as the VP of Sustainable Technology at Dow Chemical Company, the difficulties and successes of applying machine learning and GP to the development of

more energy efficient manufacturing processes, and the future use of GP in designing new materials.

In addition to the presentations and keynote speakers, we expanded the format of the workshop this year to include "whiteboard sessions"—discussions led by a participant of the workshop on a topic of ongoing research. The goal of these sessions was to expand the speculative nature of the workshop so that people could present and discuss topics on which their thinking was not yet complete in order to solicit new ideas and suggestions on these topics. Though the results of these sessions do not appear in this book, they provided another avenue of discussion for ideas that were not quite ready for general release. The hope is that these sessions will take advantage of the collected experience and knowledge of the participants and will lead to future papers and presentations at GP Theory and Practice.

Here is a list of the whiteboard sessions and the people who led the discussions:

- *Evolving Arbitrary Software*—Led by Lee Spector
- *Mobile Computing and Evolutionary Computing*—Led by Moshe Sipper
- *Application Spaces and Opportunities*—Led by William P. Worzel

October 2014 Rick Riolo, William P. Worzel and Mark Kotanchek

Acknowledgements

We would like to thank all of the participants for again making GP Theory and Practice a successful workshop. As always, it produced a lot of high energy, interesting and topical discussions, debates and speculations. The keynote speakers added a lot of food for thought and raised some interesting questions about GPs place in the world. We would also like the thank our financial supporters for making the continued existence of GP Theory and Practice possible. These include:

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

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 and after the workshop itself. Linda Wood also provided invaluable assistance finding a venue and handling other arrangements. 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. Kala Groscurth provided editorial and other assistance to RR. Jennifer Malat 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.

Contents

Contributors

Michael Affenzeller Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria

Julio Albinati DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Omri Bernstein Cognitive Science, Hampshire College, Amherst, MA, USA

Bogdan Burlacu Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria

Mauro Castelli NOVA IMS, Universidade Nova de Lisboa, Lisboa, Portugal

Chao Cheng Department of Genetics, Institute for Quantitative Biomedical Sciences, and Norris Cotton Cancer Center, Geisel School of Medicine at Dartmouth, Hanover, NH, USA

Casey S. Greene The Geisel School of Medicine at Dartmouth, One Medical Center Drive, Lebanon, NH, USA

Thomas Helmuth School of Computer Science, University of Massachusetts, Amherst, MA, USA

Douglas P. Hill The Geisel School of Medicine at Dartmouth, One Medical Center Drive, Lebanon, NH, USA

Babak Hodjat Sentient Technologies Holdings Limited, San Francisco, CA, USA

Karthik Kannappan School of Computer Science, University of Massachusetts, Amherst, MA, USA

Michael Kommenda Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria

Michael F. Korns Analytic Research Foundation, Henderson, Nevada

Gabriel Kronberger Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

William La Cava Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA, USA

Duncan MacLean Google Inc, Mountain View, CA, USA

Jason H. Moore The Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA

Luiz Otávio V.B. Oliveira DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Fernando E.B. Otero School of Computing, University of Kent, Chatham Maritime, UK

Gisele L. Pappa DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Stefano Ruberto GSSI, Gran Sasso Science Institute, L'Aquila, Italy

Hormoz Shahrzad Sentient Technologies Holdings Limited, San Francisco, CA, USA

Sara Silva BioISI, FCUL, University of Lisbon, Lisbon, Portugal

Moshe Sipper Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Lee Spector School of Cognitive Science, Hampshire College, Amherst, MA, USA

Leonardo Vanneschi NOVA IMS, Universidade Nova de Lisboa, Lisboa, Portugal

Stefan Wagner Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

Stephan M. Winkler Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg, Austria

Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria

Jake Wisdom Cognitive Science, Hampshire College, Amherst, MA, USA

William P. Worzel Evolution Enterprises, Milan, MI, USA

Chapter 1
Application of Machine-Learning Methods to Understand Gene Expression Regulation

Chao Cheng and William P. Worzel

1.1 Introduction

For a long time molecular biology studies were known to be time-consuming and labor-intensive. The amount of data generated was mostly at small-scale and easy to be analyzed. This situation has changed radically with the development of microarray technologies in middle 1990's, which can quantify the expression levels of tens of thousands of genes simultaneously. More recently, owing to the advent of the next-generation sequencing technologies, an enormous amount of biological data has been produced. A single massively parallel sequencing platform (e.g. Illumina HiSeq) is able to generate terabytes of raw data in one day. These high-throughput technologies have been used in several large-scale projects such as ENCODE (the Encyclopedia of DNA Element) (ENCODE Consortium 2012), TCGA (The Cancer Genome Atlas) (Kandoth et al. 2013) and Roadmap Epigenomics (Chadwick 2012).

Specifically, ENCODE is a research project sponsored by the US National Human Genome Research Institute (NHGRI) in 2003 (ENCODE Consortium 2012). It was launched as a follow-up to the human genome project (HGP), which was an international project initiated in 1990 and completed in 2003. In 2001 this project published the whole human genome sequence containing approximately 3.3 billion base-pairs (Lander et al. 2001; Venter et al. 2001). In the human genome, only 1.5 % of DNA encodes proteins, the function of the remaining 98.5 % of the human genome is unknown. The goal of ENCODE project is to utilize high-throughput technologies to identify functional elements in the human genome and determine their roles in regulating gene expression. Functional elements are regulatory units of the human

C. Cheng (✉)
Department of Genetics, Institute for Quantitative Biomedical Sciences, and Norris Cotton Cancer Center, Geisel School of Medicine at Dartmouth, Hanover, NH, USA
e-mail: chao.cheng@dartmouth.edu

W. P. Worzel
Evolution Enterprises, 214 W. Main St., 48160 Milan, MI, USA

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_1

genome including promoters, enhancers, insulators, silencers and transcription factor binding sites, etc. Obtaining a catalogue of these elements is an essential step to understand how the human genome is organized and regulated. Considering the fact that certain experiments are impossible or difficult to carry out in humans, the ENCODE project is extended to three model organisms: fly, worm (modENCODE) and mouse (mouse ENCODE) (Gerstein et al. 2010; Stamatoyannopoulos et al. 2012).

These projects have produced an enormous amount of data. For instance, the ENCODE project has generated more than 2600 genomic datasets from RNA-seq, ChIP-seq (Chromatin Precipitation followed by massive parallel Sequencing), CAGE (Cap Analysis of Gene Expression) and other experiments (ENCODE Consortium 2012). RNA-seq is a sequencing based method that can quantify the expression levels of genes in biological samples. ChIP-seq is developed to determine genome wide transcription factor (TF) binding or histone modification events. In a ChIP-seq experiment, an antibody is utilized to collect DNA molecules binding with a TF or enriched for a specific type of histone modification. DNA molecules are then fragmentized and sequenced in a sequencing machine to obtain millions of reads. These reads are mapped to the genome to determine their positions to obtain the binding signals of a TF or the histone modification signals of a specific type of histone mark (e.g. H3K4me3) in a sample at all positions in the genome. The ENCODE project has produced a total of 1,479 ChIP-seq datasets to capture transcription factor binding or histone modification patterns in different human cell lines. Specifically, 1,242 of them are TF binding datasets, accounting for 199 ($> 10\,\%$) of human factors. These datasets provide unprecedented opportunities to elucidate the mechanism of transcriptional regulation in human genome.

Transcription is a highly supervised process, in which two interrelated factors are involved: TFs and histone modifications (Kurdistani et al. 2004; Berger 2007; Farnham 2009). TFs are a specific family of proteins which account for $\sim 10\,\%$ of the proteins encoded by the human genome. They can act as activators, co-activators, repressors, and chromatin remodeling factors, and bind to specific gene regulatory elements (such as promoters, enhancers, silencers, and insulators) to cooperatively induce a unique pattern of gene expression (Maston et al. 2006). In the human genome, DNA is non-covalently associated with histone proteins (H1, H2A, H2B, H3, and H4) to form high order nucleoprotein structures. Histones can be modified biochemically to alter the local chromatin structure. Basically there are two chromatin states: an "open" state is highly accessible to transcription regulators to turn on gene expression, and a "close" state is poorly accessible to transcription regulators and thus gene expression is off. There are many possible histone modification types. For example, H3K4me3 is a modification that adds three methyl- groups to a lysine amino acid of the histone H3. Combinations of different histone modifications result in a "histone code" that ultimately dictate the structural status of chromatin (Strahl and Allis 2000). Along with transcription factors, histone modifications result in a carefully orchestrated and complex level of gene expression regulation.

Machine learning approaches have been widely applied to biological studies. Particularly, the large-scale projects have provided useful data for training and testing models developed to address a variety of biological questions. In this chapter, we will introduce several examples in which machine learning models are applied to analyze

the data from ENCODE and modENCODE. First, we will describe a quantitative model that relates gene expression levels with TF binding and histone modification signals. Second, we will show how we use machine learning methods to predict TF binding sites and tissue specific enhancers based on histone modification patterns and sequence features. Third, we will introduce a machine learning method to predict cell cycle regulated genes. Finally, we will discuss the advantages and limitations of genetic programming techniques in biological applications.

1.2 Application of Machine Learning to Predict Gene Expression from TF Binding and Histone Modifications

Gene expression is under precise regulation by TFs and histone modifications (HMs). In this example, we construct predictive models to quantify the relationship between gene expression levels and TF binding and histone modification signals (Cheng et al. 2011b, 2012; Cheng and Gerstein 2012). With these models, we aim to investigate how much variation of gene expression levels can be explained by TF binding and histone modification signals, respectively. We have tested the models in multiple species ranging from yeast to human. Here we show the results using data obtained from mouse embryo stem cells (mESCs). Specifically, the data contain ChIP-seq profiles for 12 TFs (E2f1, Esrrb, Klf4, Nanog, Oct4, Stat3, Smad1, Sox2, Tcfcp2l1, Zfx, c-Myc and n-Myc) and 7 histone modifications (H3K4me1, H3K4me2, H3K4me3, H3K4me9, H3K20me3, H3K27me3 and H3K36me3), as well as gene expression data from RNA-seq (Mikkelsen et al. 2007; Chen et al. 2008; Cloonan et al. 2008). The models take TF binding and/or histone modification signals as the "input" and relate them to the "output": expression levels of genes.

Figure 1.1 shows the schematic diagram we used for predicting gene expression levels from TF binding data. Given the genomic binding data for a total of M TFs, our prediction model contains the following steps. First, we divided the DNA region around the transcriptional start site (TSS) of genes into 80 bins, each of 100 bp in size. For each bin, we calculated the average signal for each chromatin feature (i.e. TFs), resulting in a matrix of G∗M, here G is the total number of genes and M is the number of TFs. Second, in each bin we constructed a model that used the binding signal of all TFs as predictors to predict the expression levels of genes (the first layer models). Finally, the predicted expression values in all bins are combined by a second layer model to make the final prediction. We applied several supervised machine learning approaches including Random Forest (RF), Support Vector Regression (SVR) and multivariable linear regression model (MLR).

We evaluated the predictive accuracy of models using a cross-validation method. Specifically, we randomly selected 2000 genes as training data and the rest as test data. We trained the bin-specific models or the two-layer model with the training data, and then applied it to predict the expression levels of genes in the test data. The correlation between the predicted values and the experimentally measured expression levels of genes in the test data were calculated as the predictive accuracy.

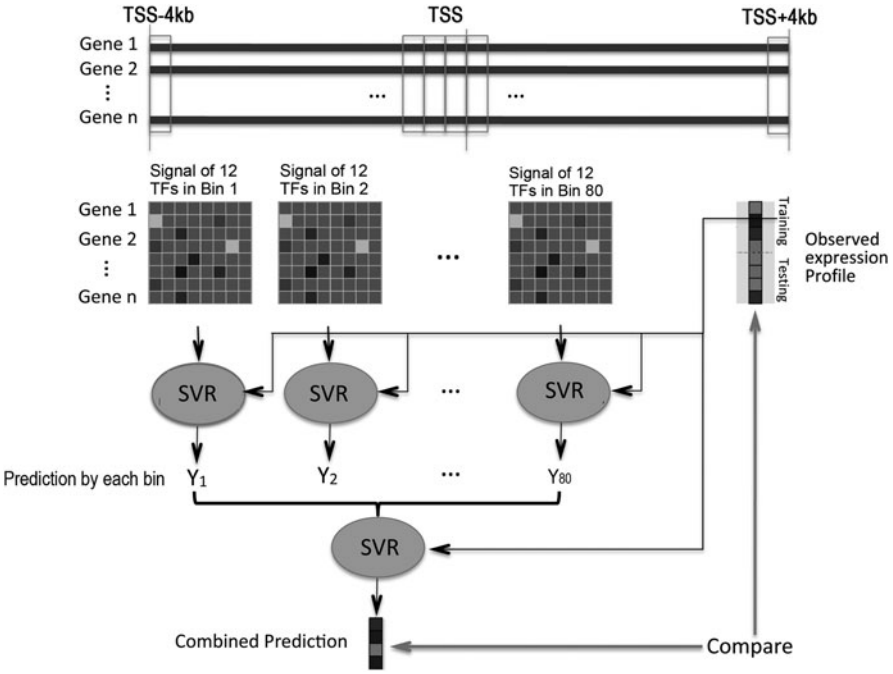


Fig. 1.1 A two-layer supervised model for predicting gene expression levels based on TF binding signals. DNA regions surrounding the Transcription Start Site (TSS) are divided into 80 bins, each of 100 bp in size. In the first layer, a model is constructed for each bin to predict the expression values of genes using binding signals of multiple TFs as predictors. The predicted values from all bins are then combined in the second layer model to make the final prediction of gene expression

We build three sets of models using TF binding signals only (TF models), histone modification signals only (HM models) and a combination of them (TF+HM models), respectively. Different machine learning methods achieved comparable predictive accuracy. Here we focus on the results from the SVR method. Figure 1.2 shows the predictive accuracy of the three sets of models in 80 bins around the TSS as well as in 80 bins around the transcriptional terminal site (TTS). As shown, the highest predictive power ($R = 0.71$) of the TF models is achieved at the TSS, which individually accounts for 50 % of the variation in gene expression. In contrast, the highest predictive bins of the HM models are within the transcribed region immediately downstream of the TSS, which achieves similar prediction accuracy as the best prediction by TF models. As shown, the TF models and HM models show very different patterns in their prediction accuracy profiles—the TF models show best prediction accuracy at a narrow DNA region at the TSS and the predictive power decays quickly away from it; while the HM models are highly predictive to expression across a broad range of transcribed regions that extends from upstream of the TSS to downstream of the TTS. Moreover, the TF+HM models only obtain prediction

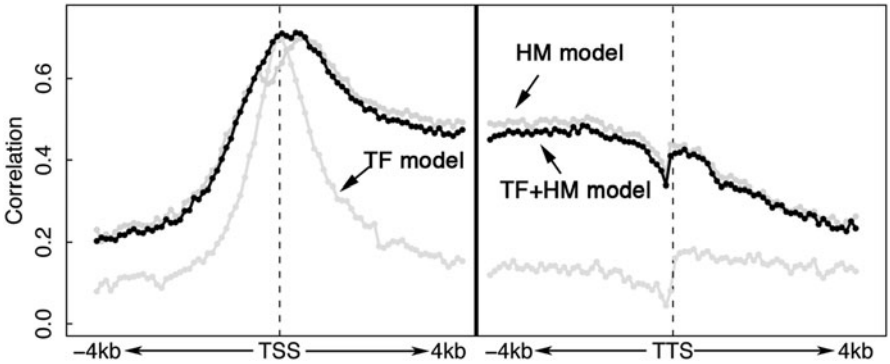


Fig. 1.2 The prediction accuracy of three set of models. The predictive accuracy of the TF models, the HM models and the TF+HM models, in each of the 160 bins (80 bins surrounding TSS and 80 bins surrounding TTS of genes) are shown. Accuracy is measured as the correlation between predicted and measured expression levels. TSS: Transcription Start Site; TTS: Transcription Terminal Site

accuracy similar to that of the HM models or the TF models across all bins, suggesting that the TF binding signal and HM signal are redundant for gene expression prediction.

In addition to the mouse data, we have applied the histone modification model to four other species including yeast, worm, fly and human using data generated by ENCODE, modENCODE and previous publications. Our results indicate that in all these organisms, about 50 % of gene variation can be explained by TF binding or histone modification signals in the TSS-proximal DNA region.

1.3 Application of Machine Learning to Predict Transcription Factor Binding Sites and Enhancers

Identification of TF binding sites (TFBSs) is critical for investigating transcriptional regulation of genes. The specific binding of a TF is mediated by the recognition of its binding motif represented as a position-weighted matrix (PWM) (Stormo 2000). TFBSs in a genome can be predicted by computational methods or determined by experimental approaches. Computationally, TFBSs can be predicted by searching DNA sequences for the associated PWM of a TF. Experimentally, ChIP-seq and ChIP-chip have been widely used to identify the genomic binding sites of TFs (Ren et al. 2000; Johnson et al. 2007).

Histone modifications can modulate the accessibility of DNA regions and affect the recruitment of TFs (Li et al. 2007). We thus developed machine learning approaches to predict TFBSs based on histone modification signals (Gerstein et al. 2010; Cheng et al. 2011a; Yip et al. 2012). Figure 1.3 shows an example of using modENCODE worm data to predict the binding sites of a TF, HLH-1. As shown,

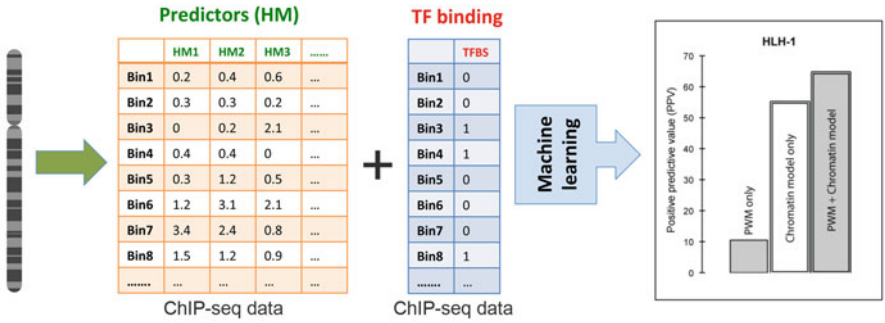


Fig. 1.3 Application of Machine Learning to Predict TF Binding sites in C. Elegans Genome. C. elegans genome is divided into continuous small bins each of 100 bp in size (Left panel). Average histone modification signals of multiple types were calculated in each bin. These histone modification signals are utilized as predictors to predict whether a bin is bound by a specific TF. 1 and 0 indicate binding and non-binding events, respectively. The positive predictive values of the PWM model, the Chromatin model and the PWM+Chromatin model for HLH-1 are shown in the right panel

the worm genome was divided into small continuous bins of 100 bp in size, and the average signals for 13 histone modification types in early embryonic cells were calculated. These signals were used as predictors to predict whether the bins are bound with HLH-1. Histone modification signals and HLH-1 binding signals are determined by ChIP-chip experiment.

Figure 1.3 shows the positive predictive values (PPVs) of three models: PWM model, Chromatin model, and the PWM+Chromatin model. The PWM model simply examines the existence of HLH-1 binding motif in each bin and identifies bins with one or more motifs as the positive binding bins of HLH-1. The Chromatin model combines 13 histone modification signals as predictors using Support Vector Machine (SVM). The PWM+Chromatin model identify HLH-1 binding bins as those that are classified as positive by the Chromatin SVM model and contain one or more HLH-1 motifs. The accuracy of the models are assessed by comparing predictions with the HLH-1 ChIP-chip data. As shown, only 10 % of HLH-1 motif-containing bins are true positives. In contrast, the Chromatin only model achieves a much high accuracy with PPV=58 %. Moreover, the accuracy can be further improved when PWM information is included (the PWM+Chromatin model). These results indicate that chromatin structures are highly informative for predicting TF binding sites.

A specific type of DNA elements called enhancers play crucial roles in the regulation of tissue specific gene expression. An enhancer is a short (50–1500 bp) region of DNA that can be bound with TFs to activate the transcription of genes (Pennacchio et al. 2013). Unlike promoters that localize in DNA regions proximal to TSS of genes, enhancers are distant cis-acting elements, often localize in intergenic or intronic region, and regulate expression of genes a few mb away. Active enhancers are known to be associated with certain histone marks (e.g. H3K4me1 and H3K27ac) (Creyghton et al. 2010). Thus we developed machine learning models to predict tissue

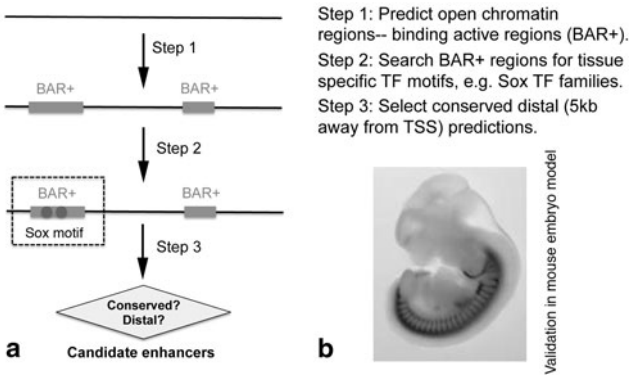


Fig. 1.4 Prediction of Human Tissue Specific Enhancers. (a) A three-step procedure for predicting embryo specific enhancers in human. **(b)** Experimental validation of predicted enhancers in a mouse embryo system. The dark regions mark the tissues in which the tested enhancer is active

specific enhancers in human using ENCODE data (ENCODE Consortium 2012). In the models, we adopt the following steps to predict tissue specific human enhancers (Fig. 1.4a) (Yip et al. 2012).

First, we constructed a Random Forest model to predict the binding active regions (BAR) across the whole human genome. A BAR is defined as the genomic region where transcription factors tend to bind, which is associated with open chromatin structure highly accessible to TFs. To train the model, 100 bp bins overlapping with the TF binding peaks were collected as positive examples, and non-positive bins were randomly sampled from the whole genome as negative examples. The model predicted a list of candidate BAR bins in the human genome, presumably from DNA regions with open chromatin structure. Second, we merged adjacent bins into longer regions, and identified the merged BARs that are enriched in the binding motifs of embryo specific TFs (e.g. SOX and OCT family members) to predict embryo specific enhancers. Finally, we selected candidate enhancers from these regions that are conserved across different species and localize at least 5kb away from any known TSSs.

In order to assess the accuracy of our predictions, we selected 6 candidate enhancers out of the positive predictions and validated their enhancer activity in a mouse embryo system. In this system, enhancers were inserted into an expression vector on the upstream of a lacZ reporter gene fused with an hsp68 promoter, and then transfected into day E11.5 embryos of transgenic mice (Pennacchio et al. 2006). If an enhancer is active in some tissues, it will initiate the transcription of the report gene and display blue colors in these tissues. Figure 1.4b shows an example of enhancer that is active in the spine tissues. Among the 6 tested predictions, 5 (83 %) were found to have enhancer activities in various tissues with high reproducibility.

1.4 Application of Machine Learning to Predict Cell Cycle Regulated Genes

Cell cycle is an important biological process in which a series of events take place in a cell leading to its division and duplication to produce two daughter cells. The cell cycle is under precise regulation at different levels (Orlando et al. 2008). At the transcriptional level it has been shown that a series of TFs act at different phases of the cell cycle and coordinate the sequential transcription of cell cycle genes (Simon et al. 2001; Cheng and Li 2008). In humans, more than a thousand genes exhibit periodic expression patterns during the cell cycle. The periodic expression pattern of cell cycle genes is encoded in cis in their promoters and can be manifested in trans by the TFs that bind to them. In other words, genes that are bound and regulated by cell cycle regulators are likely to be cell cycle genes.

Based on this rationale, we constructed Random Forest models to predict cell cycle genes (Cheng et al. 2013). In the models, we utilized the ENCODE ChIP-seq-derived TF-binding data and TRANSFAC-derived motif matching data as predictors (Fig. 1.5a). More specifically, we calculated the regulatory scores for all human genes to obtain 424 TF binding profiles. These binding profiles represent binding strength of TFs to human genes in a number of different cell lines. In addition, we examine the existence of all TRANSFAC TF binding motifs in the promoters of genes (from TSS to upstream 1kb), resulting in a total of 546 motif matching score profiles. To train the model, we used the cell cycle and non-cell cycle genes identified from a time course data that measured gene expression at multiple time points of the cell cycle in HeLa cells (Whitfield et al. 2002). Out of the 424 TF binding profiles, 46 are from HeLa cells, while the rest are from other human cell lines.

We constructed three models to classify cell cycle versus non-cell cycle genes using Random Forest method. In a TF model the trans TF-binding features were used as predictors; in a Motif model the cis motif features are used as predictors; and a TF+motif model uses a combination of all of the features. The performance of these models was evaluated by 10-fold cross-validation. Our results suggest that both TF binding features and motif features are informative for cell cycle gene prediction– the TF model achieves a prediction accuracy AUC=0.768 and the motif model achieved AUC=0.642 (Fig. 1.5b). This suggests that the trans- TF binding features are more informative than the cis- motif features. Moreover, a combination of both sets of features (the TF+motif model) can significantly improve the prediction accuracy, leading to an AUC=0.861. This indicates that the trans- information captured by ChIP-seq data and the cis- information provided by the motif analysis complement each other in cell cycle prediction.

Apart from the Random forest model, we also implemented other machine learning methods, including support vector machine (SVM) and penalized logistic regression (PLS). The prediction accuracy of these models is slightly lower than the Random Forest model. In addition, results from these models confirmed the finding and conclusions from the Random Forest model, e.g. higher predictive accuracy of TF binding features than motif features.

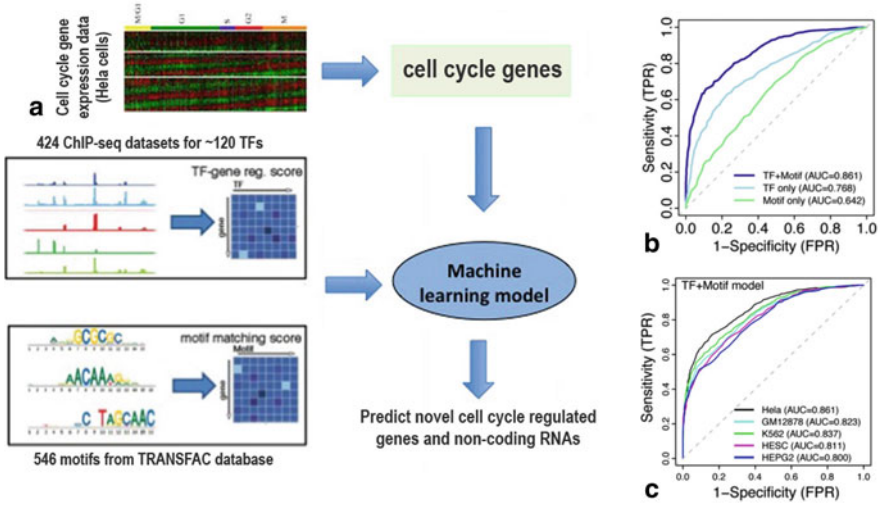


Fig. 1.5 Application of Machine Learning Models to Predict Cell Cycle Genes. (a) The schematic diagram of our cell cycle prediction model. **(b)** The ROC curve of three models: TF only model, Motif only model, and TF+Motif model. **(c)** TF binding data from HeLa cells achieves the highest prediction accuracy for predicting cell cycle regulated genes in HeLa cells

In the 424 TF binding profiles, there are 68 from GM12878, 94 from K562, 37 from HESC and 55 from HEPG2 cell lines, respectively. We thus examined the cell line specificity of our cell cycle gene prediction model. If cell cycle regulation is cell line specific, we would expect the best prediction accuracy using HeLa TF binding profiles; and otherwise a similar accuracy throughout different cell lines. As shown in Fig. 1.5c, indeed the highest prediction accuracy was achieved when matched data are used– the TF binding features from the HeLa cell line are used for predicting HeLa cell cycle genes. Thus, these results suggest that cell cycle genes are cell line specific and exhibit differences in their regulation between cell lines.

1.5 Application of Genetic Programming to Biological Studies

Genetic programming (GP) and genetic algorithm (GA) have been widely used in different fields of biological studies including drug design, association studies and cancer research, etc (Ghosh and Bagchi 2009; Worzel et al. 2009; Khan and Alam 2012). GP is distinct from other common machine learning algorithms used in bioinformatics. In this section, we will discuss about the advantages and limitations of GP (Mitra et al. 2006; Moore and White 2006).

1.5.1 Advantages of Genetic Programming

Interpretability Many machine learning methods such as SVM and Neural Networks do not provide human readable results. In contrast, the final output of GP consists of easily readable rules. For example, when GP is applied to classify cancer subclasses, the resulting rules are expressed as executable classifier programs that define tangible relationships between the most influential genes. That is, the results of GP are easily interpretable. The findings may be interpreted in the biological context of these genes and provide new testable working hypotheses. While hierarchical clustering provides visually intuitive results, it does not provide exact relationships among the features. Classification and regression trees (CART) output a binary decision tree, which is interpretable but it fails to provide clear insights into the relationships among features—these relationships become less explicit and difficult to discern when the tree grows larger. Moreover, CART algorithms are normally felt to be greedy, leading to a locally optimized solution (Eggermont et al. 2004). GP takes a more global view by searching a larger space for solution trees so it is likely to have better performance.

Automatic Feature Selection GP can also select features automatically without any need to pre-filter or limit them based on what is known about a system. This property of GP is favorable, since filtering may create an incomplete and biased dataset that become not representative of a complex biological system. Basically, the "curse of dimensionality" affects all classification algorithms but the problem of dimensional reduction is more important in some classical algorithms, e.g. hierarchical clustering and Neural Networks, which do not scale easily to larger numbers of variables. Feature selection is then an important step before the application of these algorithms, which may lead to loss of information that is critical for the success of them.

Small Set of Selected Features GP is capable of identify a small set of features that are most influential to the response variable. This property is especially useful when GP is applied to biomedical problems, e.g. identify a gene set for predicting clinical outcome in cancer. Without compromising their predictability, GP can usually limit the complexity of the classifiers and generate robust but simple rules containing only a few genes. A smaller gene set means lower cost in a clinical diagnosis, which makes it affordable to more customers. Moreover, Occam's Razor suggests that the smaller the solution, the more robust it is. Overfitting is the price paid for producing overly complex models, so GP's ability to make concise selections of features helps to discover robust solutions.

Non-linear Relationships GP can choose variables from a large list and then combine them in a non-linear, readable way. This is a powerful character of GP, since many biological systems have non-linear relationships between genes or proteins. SVM outputs non-linear classifiers but it is limited to the kernel selected. CART implements nonlinearity in a pseudo sense as they split the data and tackle each

partition separately, but it is not as succinct as the rules produced by GP in capturing the non-linear relationships among genes.

Missing Values GP can incorporate very diverse data sets that contain different types of variables and can also handle missing values in the data. Missing values in a dataset can be problematic as even a small amount of missing data may result in a considerable loss in performance. One can process the missing values by imputation or by replacing them with a constant. This may introduce bias in the data. GP alleviates this problem by leveraging the ability of the system to select features. During fitness assessment, a sample with missing data may be considered as misclassified by a rule and thus decrease the fitness of the rule. As a consequence, variables with a large percentage of missing values are not favored for picking up by the GP system. This approach allows for maximum use of the available data without making any unwarranted assumptions about missing data.

Multiple Data Types GP can integrate categorical, integer, and real valued data seamlessly to produce rules that are conditional on state information. For example, patient demographics or status (e.g., gender or disease state) can be used to produce conditional classifiers where different biomarkers are used depending on these conditional factors. Similarly, a predictor could be different depending on where a cell is in the cell cycle.

Continuous Functions GP can also be adapted to create continuous functions from time series data. This allows GP to produce dynamic models which could describe many biological processes. The growing volume of data in biology, combined with the increasing ability to measure cell states, may make it possible to create empirical models that give insight into dynamic processes (Kotanchek et al. 2012).

1.5.2 Limitations of Genetic Programming

Computationally Intensive GP is computationally intensive that requires a large amount of machine time. The estimated machine time increases with increasing complexity of the problem, and increase in the dimensions and number of samples. This can be resolved by using parallel computing– segmenting the problem into parts and then processing them simultaneously on different processors with synchronization. Strikingly, GP is particularly tractable for parallel computing techniques as there are several natural ways to distribute execution onto different machines (Andre and Koza 1996).

Inconsistent Rules from Different Runs As GP is a stochastic process that depends highly on the initial control parameter settings, it does not guarantee an optimal solution in all runs. Therefore, it should be repeated several times with different settings to ensure that the system has not fallen into a local optima. GP may also output several rules that are quite different but perform equally well. One can create

an ensemble classifier to utilize these rules to make final prediction. To further refine these rules, one should refer to the biological functionality of the features (genes) and validate them in independent datasets.

1.5.3 Possible Uses of Genetic Programming to Predict Gene Expression Levels Based on TFs and HMs

As described above, GP is a powerful and flexible machine learning approach. When applied to the question of the effect of TFs and HMs to gene expression levels, they can be used in place of the SVR or other machine learning algorithms with the difference that GP would produce human readable results, clearly showing the interaction between TFs, HMs, and the combination of the two in controlling gene expression levels.

It is also possible that a two-step process would not be necessary in GP as it could be used to combine binding values for each bin to produce a gene expression predictor. This could be done by using the set of MxG matrices as inputs (essentially creating an NxMxG input matrix) to evolve a predictor for the gene expression levels. The form of this predictor would create functional rules that combine TF signals from each bin to predict gene expression levels.

1.5.4 Predicting Binding Sites and Enhancers Using Genetic Programming

Similarly, predicting binding sites using GP would allow all inputs described in Fig. 1.3 to be used (PWM, Histone binding signals and HLH-1) to produce a predictive model. GP would select from these inputs to produce the best combination based on the training set.

Discovering enhancers is a different problem. In this case GP would replace the Random Forest model. GP generally performs better than Random Forest models because it combines a global search with a local optima and, because there are a population of solutions, an ensemble can be discovered based on the best predictors across multiple runs or folds.

The question of location and extent of an enhancer is an interesting one. Certainly fusing bins together is an easy and effective way to proceed but adapting (Kotanchek et al. 2006) suggests another option where a Pareto optimal solution is used to find those predictors that are both concise and predictive. Adapting this approach to the problem at hand, it may be possible for GP to find both location and size of an enhancer where length of sequence associated with the enhancers along with the ability to predict a location of an enhancer may make it possible to do away with binning altogether. One may think of this as allowing GP to open and close a sliding window to find the best location and length for an enhancer.

1.5.5 Using GP to Predict Cell Cycle Genes

As with the earlier studies, GP can be used as another machine learning approach to classify genes are being cell-cycle related genes. However another more intriguing possibility exists. Koza et al. (2001) used GP to model both the topology and the dynamics of known metabolic pathways that included feedback loops. Though this was based on part of an already well characterized part of a phospholipid cycle, it provides an interesting opportunity given the increased data and computing power available.

Given the nature of cell cycle and the approach described in Sect. 1.4 to characterize the interaction of TFs, HMs, promoters and enhancers, and the ability to capture time course data for these and correlate them with gene expression data, it would provide a good test of the ability of GP to build de novo models of a complex biological system.

References

Andre D, Koza J (1996) A parallel implementation of genetic programming that achieves super-linear performance. Proceedings of the international conference on parallel and distributed processing techniques and applications, CSREA Press, Sunnyvale:A.H.R.

Berger S (2007) The complex language of chromatin regulation during transcription. Nature 447(7143):407–412

Chadwick L (2012) The NIH roadmap epigenomics program data resource. Epigenomics 4(3):317–324

Chen X, Xu H, Yuan P, Fang F, Huss M, Vega V, Wong E, Orlov Y, Zhang W, Jiang J (2008) Integration of external signaling pathways with the core transcriptional network in embryonic stem cells. Cell 133(6):1106–1117

Cheng C, Gerstein M (2012) Modeling the relative relationship of transcription factor binding and histone modifications to gene expression levels in mouse embryonic stem cells. Nucleic Acids Res 40(2):553–568

Cheng C, Li L (2008) Systematic identification of cell cycle regulated transcription factors from microarray time series data. BMC Genomics 9:116

Cheng C, Shou C, Yip K, Gerstein M (2011a) Genome-wide analysis of chromatin features identifies histone modification sensitive and insensitive yeast transcription factors. Genome Biol 12(11):R111

Cheng C, Yan K, Yip K, Rozowsky J, Alexander R, Shou C, Gerstein M (2011b) A statistical framework for modeling gene expression using chromatin features and application to modencode datasets. Genome Biol 12(2):R15

Cheng C, Alexander R, Min R, Leng J, Yip K, Rozowsky J, Yan K, Dong X, Djebali S, Ruan Y (2012) Understanding transcriptional regulation by integrative analysis of transcription factor binding data. Genome Res 22(9):1658–1667

Cheng C, Ung M, Grant G, Whitfield M (2013) Transcription factor binding profiles reveal cyclic expression of human protein-coding genes and non-coding rnas. PLoS Computational Biol 9(7):e1003132

Cloonan N, Forrest A, Kolle G, Gardiner B, Faulkner G, Brown M, Taylor D, Steptoe A, Wani S, Bethel G (2008) Stem cell transcriptome profiling via massive-scale mrna sequencing. Nat Methods 5(7):613–619

Creyghton M, Cheng A, Welstead G, Kooistra T, Carey B, Steine E, Hanna J, Lodato M, Frampton G, Sharp P (2010) Histone h3k27ac separates active from poised enhancers and predicts developmental state. Proceedings of the National Academy of Sciences of the United States of America 107(50):21,931–21,936

Eggermont J, Kok J, Kosters W (2004) Genetic programming for data classification:partitioning the search space. Proceedings of the 2004 ACM symposium on Applied computing ACM Press, Nicosia, pp 1001–1005

ENCODE Project Consortium (2012) An integrated encyclopedia of DNA elements in the human genome. Nature 489(7414):57–74

Farnham P (2009) Insights from genomic profiling of transcription factors. Nat Rev Genet 10(9):605–616

Gerstein M, Lu Z, Nostrand EV, Cheng C, Arshinoff B, Liu T, Yip K, Robilotto R, Rechtsteiner A, Ikegami K (2010) Integrative analysis of the caenorhabditis elegans genome by the modencode project. Science 330(6012):1775–1787

Ghosh P, Bagchi M (2009) Qsar modeling for quinoxaline derivatives using genetic algorithm and simulated annealing based feature selection. Curr Med Chem 16(30):4032–4048

Johnson D, Mortazavi A, Myers R, Wold B (2007) Genome-wide mapping of in vivo protein-dna interactions. Science 316(5830):1497–1502

Kandoth C, McLellan M, Vandin F, Ye K, Niu B, Lu C, Xie M, Zhang Q, McMichael J, Wyczalkowski M (2013) Mutational landscape and significance across 12 major cancer types. Nature 502(7471):333–339

Khan M, Alam M (2012) A survey of application: genomics and genetic programming, a new frontier. Genomics 100(2):65–71

Kotanchek M, Smits G, Vladislavleva E (2006) Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In: Riolo RL, Soule T, Worzel B (eds) Genetic programming theory and practice IV, genetic and evolutionary computation, vol 5. Springer, Ann Arbor, pp 167–185. doi:10.1007/978-0-387-49650-4–11

Kotanchek ME, Vladislavleva E, Smits G (2012) Symbolic regression is not enough: It takes a village to raise a model. In: Riolo R, Vladislavleva E, Ritchie MD, Moore JH (eds) Genetic programming theory and practice X, genetic and evolutionary computation. Springer, Ann Arbor, pp 187–203. doi:10.1007/978-1-4614-6846-2-13, http://dx.doi.org/10.1007/978-1-4614-6846-2–13

Koza JR, Mydlowec W, Lanza G, Yu J, Keane MA (2001) Automatic synthesis of both the topology and sizing of metabolic pathways using genetic programming. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt HM, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 57–65. http://www.cs.bham.ac.uk/ wbl/biblio/gecco2001/koza-gecco2001.pdf

Kurdistani S, Tavazoie S, Grunstein M (2004) Mapping global histone acetylation patterns to gene expression. Cell 117(6):721–733

Lander E, Linton L, Birren B, Nusbaum C, Zody M, Baldwin J, Devon K, Dewar K, Doyle M, FitzHugh W (2001) Initial sequencing and analysis of the human genome. Nature 409(6822):860–921

Li B, Carey M, Workman J (2007) The role of chromatin during transcription. Cell 128(4):707–719

Maston G, Evans S, Green M (2006) Transcriptional regulatory elements in the human genome. Annu Rev Genomics Hum Genet 7:29–59

Mikkelsen T, Ku M, Jaffe D, Issac B, Lieberman E, Giannoukos G, Alvarez P, Brockman W, Kim T, Koche R (2007) Genome-wide maps of chromatin state in pluripotent and lineage-committed cells. Nature 448(7153):553–560

Mitra A, Almal A, George B, Fry D, Lenehan P, Pagliarulo V, Cote R, Datar R, Worzel W (2006) The use of genetic programming in the analysis of quantitative gene expression profiles for identification of nodal status in bladder cancer. BMC Cancer 6:159

Moore J, White B (2006) Genome-wide genetic analysis using genetic programming: the critical need for expert knowledge. In: Riolo RL, Soule T, Worzel B (eds) Genetic programming theory and practice IV, Springer, genetic and evolutionary computation, vol 5, pp 11–28

Orlando D, Lin C, Bernard A, Wang J, Socolar J, Iversen E, Hartemink A, Haase S (2008) Global control of cell-cycle transcription by coupled cdk and network oscillators. Nature 453(7197):944–947

Pennacchio L, Ahituv N, Moses A, Prabhakar S, Nobrega M, Shoukry M, Minovisky S, Dubchak I, Holt A, Lewis K (2006) In vivo enhancer analysis of human conserved non-coding sequences. Nature 444(7118):499–502

Pennacchio L, Bickmore W, Dean A, Nobrega M, Bejerano G (2013) Enhancers: five essential questions. Nat Rev Genet 14(4):288–295

Ren B, Robert F, Wyrick J, Aparicio O, Jennings E, Simon I, Zeitlinger J, Schreiber J, Hannett N, Kanin E (2000) Genome-wide location and function of dna binding proteins. Science 290(5500):2306–2309

Simon I, Barnett J, Hannett N, Harbison C, Rinaldi N, Volkert T, Wyrick J, Zeitlinger J, Gifford D, Jaakkola T (2001) Serial regulation of transcriptional regulators in the yeast cell cycle. Cell 106(6):697–708

Stamatoyannopoulos J, Snyder M, Hardison R, Ren B, Gingeras T, Gilbert D, Groudine M, Bender M, Kaul R, Canfield T (2012) An encyclopedia of mouse dna elements (mouse encode). Gen Biol 13(8):418

Stormo G (2000) Dna binding sites: representation and discovery. Bioinformatics 16(1):16–23

Strahl B, Allis C (2000) The language of covalent histone modifications. Nature 403(6765):41–45

Venter J, Adams M, Myers E, Li P, Mural R, Sutton G, Smith H, Yandell M, Evans C, Holt R (2001) The sequence of the human genome. Science 291(5507):1304–1351

Whitfield M, Sherlock G, Saldanha A, Murray J, Ball C, Alexander K, Matese J, Perou C, Hurt M, Brown P (2002) Identification of genes periodically expressed in the human cell cycle and their expression in tumors. Mol Biol Cell 13(6):1977–2000

Worzel W, Yu J, Almal A, Chinnaiyan A (2009) Applications of genetic programming in cancer research. Int J Biochem Cell Biol 41(2):405–413

Yip K, Cheng C, Bhardwaj N, Brown J, Leng J, Kundaje A, Rozowsky J, Birney E, Bickel P, Snyder M (2012) Classification of human genomic regions based on experimentally determined binding sites of more than 100 transcription-related factors. Genome biol 13(9):R48

Chao Cheng is an Assistant Professor of Genetics at the Geisel School of Medicine of Dartmouth College. He is also affiliated with the Institute for Quantitative Biomedical Science and serves as a member of the Norris Cotton Cancer Center at Dartmouth College. He received his Ph.D degree in Computational Biology and Bioinformatics from University of Southern California in 2006. Dr. Cheng's research focuses on the development and application of computational and bioinformatics methods for elucidating the regulatory programs underlying normal biological processes and human diseases.

William P. Worzel is one of the original organizers of the first GP Theory and Practice workshop along with Rick Riolo. He is an entrepreneur and a consultant, whose fundamental interests are in understanding the evolutionary mechanisms of GP (and nature) in order to create better GP systems and apply them to new problems

Chapter 2
Identification of Novel Genetic Models of Glaucoma Using the "EMERGENT" Genetic Programming-Based Artificial Intelligence System

Jason H. Moore, Casey S. Greene and Douglas P. Hill

2.1 Introduction

Primary open-angle glaucoma (POAG) is a common eye disease that is characterized by an increase in intraocular pressure that, if untreated, can lead to a decrease in vision or even blindness due to progressive nerve damage. Family studies have shown that POAG has a heritable component with siblings of those affected having 5-10 times the risk of randomly selected people from the same population (Wang et al. 2010). As recently reviewed (Cooke Bailey et al. 2013), at least five genomic regions have been associated with POAG in large genome-wide association studies (GWAS). While these initial genetic studies provide some clues they do not come close to explaining the variability in risk due to genetic variation. A significant limitation of these genetic studies is that they consider each genetic variant or polymorphisms individually ignoring both genomic and ecological contexts that are likely to influence how a particular region of the genome impacts risk through a complex hierarchy of biological systems. The goal of the present study is to more fully explore the genotype-phenotype relationship in a genome-wide genetic study of POAG that embraces, rather than ignores, the complexity of the disease. It is our working hypothesis that alternatives to the one genetic variant at a time framework implemented using parametric statistical approaches such as logistic regression will reveal interesting new associations that will spark future work in this area leading to new predictive models and perhaps increased biological understanding that will open the door to new treatments.

J. H. Moore (✉) · C. S. Greene · D. P. Hill
The Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, 19104-6021, USA
e-mail: jhmoore@upenn.edu

C. S. Greene
e-mail: casey.s.greene@dartmouth.edu

D. P. Hill
e-mail: douglas.hill@Dartmouth.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_2

The availability of big data in disciplines such as human genetics has rekindled an interest in artificial intelligence (AI) and machine learning methods that are capable of identifying complex relationships among measured features such as genetic variation. This need for more powerful modeling approaches coupled with the availability of inexpensive high-performance computing and improved human-computer interaction (HCI) technology means that the timing is perfect to employ these methods for looking at large-scale human genetics data. A key feature of AI research is the desire to create a computational system that can reason or make decisions as a human would. This is in significant contrast to the current GWAS strategy that carries out each genetic analysis without any input from a human or their abundant expert knowledge base. The goal of the present study is to utilize an AI approach to the genetic analysis of POAG that explores complex relationships in the data in a manner that is much more consistent with how humans would approach manual data analysis given effectively infinite time.

We introduce here the Exploratory Modeling for Extracting Relationships using Genetic and Evolutionary Navigation Techniques (EMERGENT) algorithm as an AI approach to the genetic analysis of common human diseases. At the heart of EMERGENT is a symbolic discriminant analysis (SDA) approach (Moore et al. 2002; Moore et al. 2007) that performs classification using models constructed from a list of possible mathematical functions and a list of features or attributes. This base classification method is appealing because it makes no assumptions about the functional form of the model. This is in contrast to methods such as logistic regression that first assume a particular model to which all data are fitted. It is the base assumption of EMERGENT that human genetics data is sufficiently complex that we do not know what etiological models for diseases like POAG should look like beyond what has been learned from rare Mendelian diseases like cystic fibrosis where it is much easier to pin down the genetic cause. At this level, EMERGENT is like any other machine learning method that takes features as inputs and that produces a classifier for prediction. The goal of course is to optimize the selection of functions and features to maximize the classification accuracy of the model and to do so in a manner that more closely mimics human problem-solving. It is this last goal that distinguishes EMERGENT from other machine learning methods like decision trees or neural networks.

The framework EMERGENT uses for representing models and for exploring an effectively infinite model space is based on genetic programming (GP). Genetic programming is an automated computational discovery tool that is inspired by Darwinian evolution by natural selection (Banzhaf et al. 1998; Koza 1992). 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 more specifically (Moore et al. 2007; Moore et al. 2008b; Ritchie et al. 2003). This is an attractive approach to the genetic analysis problem because it is inherently flexible, stochastic, parallel and easily

adapted to exploit expert knowledge of the type that humans would employ in their own modeling strategies. Genetic programming falls within the scope of AI as a computational intelligence method.

The key to EMERGENT as an AI approach is two-fold. First, as described above, GP provides a flexible way to represent solutions such as symbolic discriminant functions as computer programs. Second, GP provides a stochastic and parallel search based on the principles of evolution by natural selection. These first two characteristics are at the heart of the EMERGENT algorithm and are important for model discovery. Finally, while the algorithm is discovering good models, we want the system to learn how to generate good models. This final characteristic is a meta-layer that is inspired by how humans solve problems. That is, not only can we as humans solve a complex problem but we can learn at the same time strategies that make solving the problem faster and easier. To accomplish this we have implemented a type of GP called computational evolution that learns how to generate new models while it is learning what a good model is. Computational evolution has been previously reviewed (Banzhaf et al. 2006) and has been previously employed in GP systems such as PushGP (Spector and Robinson 2002). The key to our own implementation of computational evolution within the EMERGENT framework is the ability of the system to learn to use different sources of expert knowledge to help guide the search for new models. This is intended to mimic how humans draw on past experience to solve a problem.

In the next sections we outline our previous work with the EMERGENT system, the details of the EMERGENT algorithm as employed here and then its application to a genome-wide genetic study of POAG. Our results suggest both interesting and novel genetic associations for POAG that have not been previously reported.

2.2 History of the EMERGENT Framework

Development of the EMERGENT framework (formally described generically as a Computational Evolution System or CES) described in detail below has proceeded in multiple steps. Moore et al. developed the GP-based symbolic discrimination analysis (SDA) method for flexible classification of disease using genomics data (Moore et al. 2002). This was later extended to developing models of genetic variation predictive of common disease endpoints (Moore et al. 2007). This approach was extended to include some of the features of a computational evolution system (Banzhaf et al. 2006). We developed a hierarchical, spatially-explicit GP-based system 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 (Greene et al. 2009a, b; Moore et al. 2008a; Moore et al. 2008c; 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 EMERGENT toward valid gene-gene interaction models (Pattin et al. 2010). We also showed how visualization of EMERGENT results could improve the modeling process (Moore et al. 2011). More recently, we have demonstrated how Pareto optimization (Moore et al. 2012) and measures of interestingness (Moore et al. 2013) can be used to improve the search for novel models. Here, we demonstrate how the EMERGENT framework derived from the studies summarized above can be used to look at the genetics of POAG in a novel manner.

2.3 Methods

In this section, we first present a summary of our EMERGENT AI framework for open-ended genetic analysis of complex human diseases. We then discuss our application of this approach to POAG.

2.3.1 Overview of the EMERGENT Framework

In Fig. 2.1, we provide a graphical overview of EMERGENT, 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 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). EMERGENT includes a source of expert knowledge (Fig. 2.1E) that can be used with the solution operators and as part of Pareto optimization (Fig. 2.1F). EMERGENT 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.

2.3.2 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

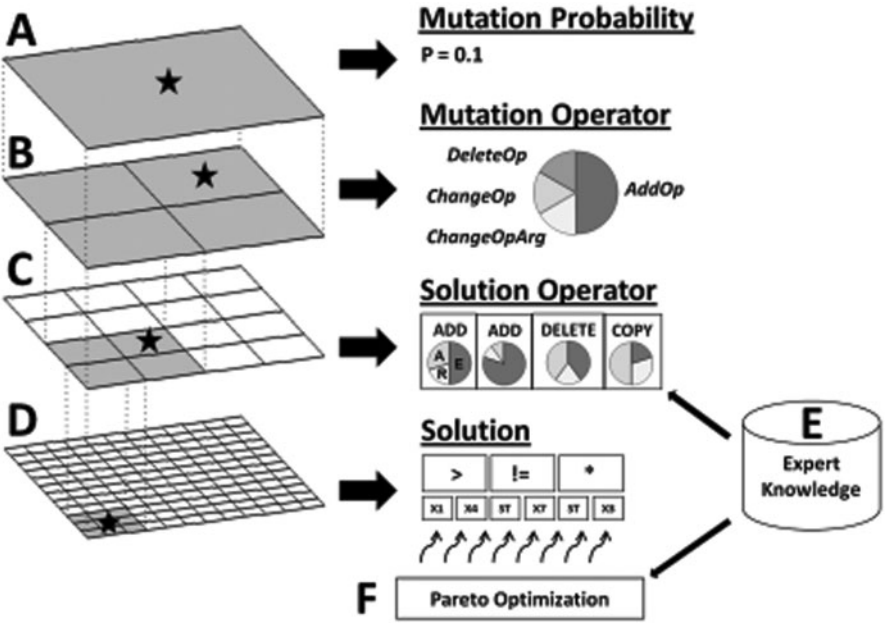


Fig. 2.1 Visual overview of our computational evolution system for discovering symbolic discriminant functions that differentiate disease subjects from healthy subjects using information about single nucleotide polymorphisms (SNPs). The hierarchical structure is shown on the left while some specific 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 EMERGENT, 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

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

$$A = (1/2)(TP/(TP + FN) + TN/(TN + FP)$$

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 features (i.e. genetic variants) 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 EMERGENT 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). As described in detail (Moore et al. 2013), we used classification accuracy, the number of features or attributes in the model (i.e. complexity) and interaction information (Moore et al. 2006) 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 (described in more detail below). First all dominated solutions and solutions evaluating to a constant are removed from competition. A solution is dominated if there exists any solution that is equal to it or better for all Pareto criteria, and better in at least one criterion. 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. In order to prevent convergence on solutions of a single length, higher selection probability is assigned to a solution if there are relatively fewer solutions of that length in the lattice. 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 one to 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. A seed may be specified for the random number generator to make results deterministic and reproducible. Optionally, some of the initial population may be replaced by solutions selected from a database. This selection is also deterministic.

2.3.3 Solution Operators

EMERGENT 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 EMERGENT recognizes. In this study we have used two types of EK, Association EK and Attribute EK. Association EK is used to help EMERGENT 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 486,726 attributes have over 1.1×2^{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 1440 most strongly interacting pairs, a tiny fraction of the total. These are the building blocks and the way they are influenced by Association EK. Pre-computer ReliefF scores were used as Attribute EK. Both of these pre-processing steps are described in additional detail below.

1. ADD: Inserts a randomly generated element into the solution at a randomly selected position.
   a. Attribute EK: The selection of attributes in the new element is biased toward those favored in the Attribute EK file.
   b. Association EK: The existing attribute just upstream of the new element is taken into account. The selection of new attributes is biased toward others in the same cluster.
2. ALTER: Modifies either the function or an argument of a randomly selected element.
   a. Attribute EK: If an attribute argument is chosen, its selection is biased toward those favored in the Attribute EK file.
   b. Association EK: If an attribute argument is chosen, the nearest upstream attribute is taken into account as in the ADD operator.
3. COPY: Within a randomly selected neighboring solution, randomly selects an element and inserts it into a randomly selected position in the focal solution.
   a. Attribute EK: The element chosen to copy has an attribute among the most favored in the Attribute EK file.
   b. Association EK: No effect.
4. DELETE: Removes an element from a randomly selected position.
   a. Attribute EK: The element chosen for deletion has an attribute among the least favored in the Attribute EK file.
   b. Association EK: No effect.
5. REPLACE: Within a randomly selected neighboring solution, randomly selects a source position. In the focal solution, REPLACE 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.

a. Attribute EK: The source position is chosen to have an attribute among the most favored in the Attribute EK file.
b. Association EK: No Effect.

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 a 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 although this would be an interesting extension to explore in a future study.

2.3.4 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. 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).

2.3.5 Primary Open Angle Glaucoma (POAG) Data

The data used in this study came from the Glaucoma Gene Environment Initiative (GLAUGEN) study (Cornelis et al. 2010) that included approximately 1272 subjects with POAG and 1057 healthy controls. A total of 657,366 single-nucleotide polymorphisms (SNPs) were measured across the human genome in these subjects. The data were obtained through application to the dbGaP resource (Mailman et al. 2007) at the National Institutes of Health (accession phs000308.v1.p1). Filtering out the SNPs with very little genetic variation (minor allele frequency < 0.05) or those that were missing 10 % or more of their values left 486,726 SNPs for analysis. Missing values were imputed using a frequency-based method. 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 (POAG vs healthy).

2.3.6 Pre-Processing, Experimental Design and Post-Processing

The goal of this study was to apply EMERGENT to the genetic analysis of POAG. We first pre-processed the data by estimating the interaction information for all pairs of SNPs as described previously (Fan et al. 2011; Hu et al. 2011; Moore et al. 2006). We considered pairs of SNPs that have higher interaction information more interesting (Association EK). We also pre-processed the data by running the ReliefF machine learning algorithm as reviewed by Moore et al. (2010). These pre-processed measures of interestingness were used as expert knowledge (Association EK and Attribute EK, respectively) in the CES solution modifiers. Also, the interaction information measure was used as an additional axis in the three-way Pareto optimization.

Each EMERGENT run was conducted with a 36×36 grid of solutions for 2000 generations. The EMERGENT system was implemented in a hierarchical framework inspired by the age-layered population structure algorithm or ALPS (Hornby 2006). Here, we implemented a depth 11 binary tree where each node represents an EMERGENT 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. We reported the best models discovered at depths 9, 10 and 11. The interaction information among the top features was used to build a network for statistical interpretation (Hu et al. 2011; Hu et al. 2013).

2.4 Results

The EMERGENT algorithm was run a total of 1024 independent times at the lowest level of the tree shown at the bottom of Fig. 2.2. Symbolic discriminant models of genetic variation from the final Pareto front constructed from sets of two runs were used to seed a new set of 512 runs. This process was repeated according to the tree shown resulting in in a final Pareto optimal set of models at the top of the tree. An advantage of using Pareto optimization is that multiple measures of model quality or interestingness can be used in the model discovery process and for final model selection. We decided to focus on picking final models that had the highest accuracy for classification of POAG, had a moderate number of features (i.e. SNPs) to guard against overfitting and that maximized the consistency of model discovery at levels 9, 10 and 11 of the tree of EMERGENT runs. We found that models with six features were highly consistent across multiple final runs. The top of Fig. 2.2 summarizes simplified versions of the symbolic discriminant models that met our selection criteria. The accuracies of these best models ranged from 0.611 to 0.615. Note that the function $X1 = X6$ and $X8 \; != X18$ showed up in all four best models from level nine and in the best models from the subsequent higher-level runs. In addition, $X3 \; != X16$ and $X12 > X7$ showed up consistently. As shown in Fig. 2.2, each of these two-feature functions formed one of three atomistic modules that came together with an overall function is interpreted as the sum of at least two of the three simplified functions. The consistency of these models each derived from hundreds

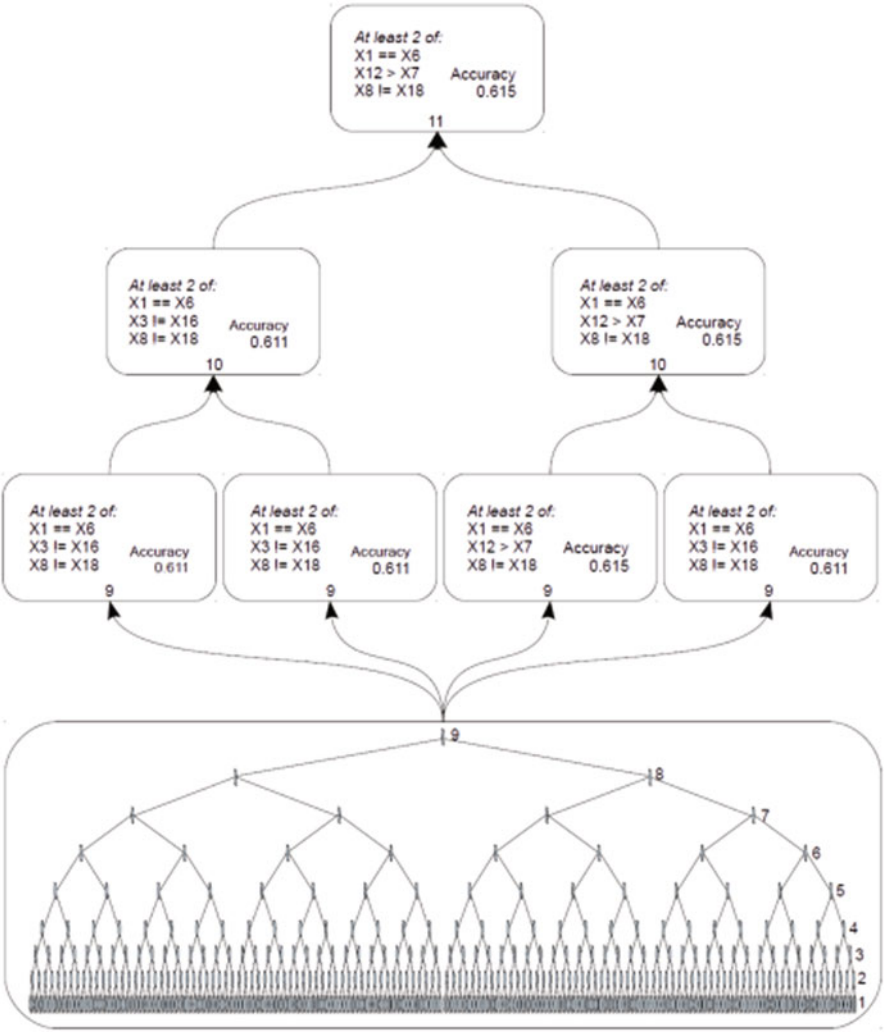


Fig. 2.2 The EMERGENT algorithm was first run 1,024 times with different random seeds (level 1 of the tree). The Pareto optimal results from pairs of runs were used to seed 512 additional independent runs. This hierarchical organization of the runs continued until there were four runs at level 9, two at level 10 and one final run at level 11. Shown are simplified versions of the best models for levels 9, 10 and 11. Note that we chose models with six features from simpler and more complex models along the Pareto front because this model size gave the most consistent results across independent runs

of independent runs with different random seeds was reassuring given the stochastic nature of the EMERGENT algorithm and the effectively infinite search space. Fig. 2.3 shows the version of the final best model prior to simplification. Note that the Boolean result of $X8$!= $X18$ is added to the Boolean result of $X12 > X7$. This result, with

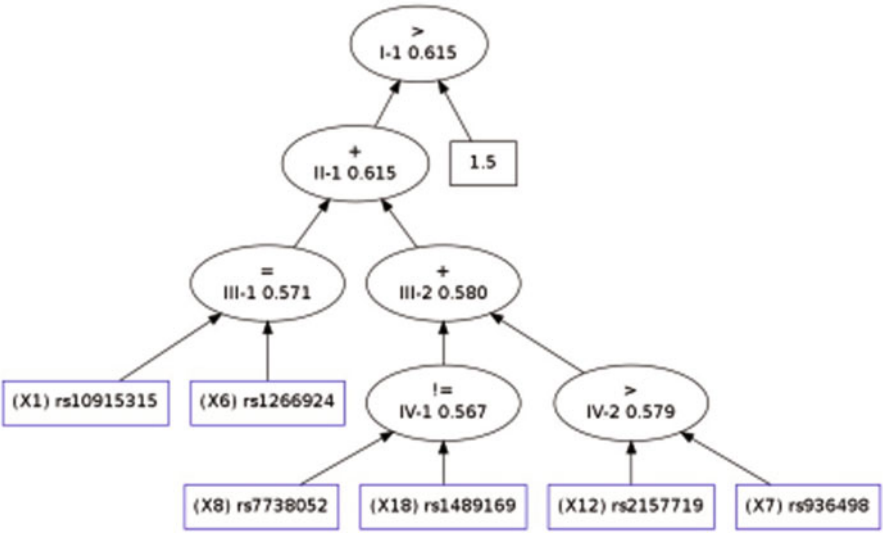


Fig. 2.3 The overall best model discovered by EMERGENT using three-way Pareto optimization and two sources of expert knowledge. The model includes features or attributes (*rectangles*), constants (*squares*) and mathematical functions or nodes (*ovals*). Each model outputs a discriminant score for each 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

possible values of $\{0, 1, 2\}$, is added to the Boolean result of $X1 = X6$ yielding possible values of $\{0, 1, 2, 3\}$. These values were then compared to a discovered threshold of 1.5 using the $>$ function. A result of one would be returned from this last function if any two or all three of the two-feature functions lower in the tree returned a one. Otherwise it would return a zero. These final values represent the predicted values of the class variable where a one indicates a human subject with glaucoma and a zero indicates a health control. The other best models were of the same form.

Figure 2.4 provides a simple clustered heatmap summarizing whether each best model piece or the overall best model correctly classified each subject as a case or a control. Here, each column (blended together) represents a human subject with the cell of the heatmap color coded light grey for a correct classification of disease and dark grey for incorrect. As indicated above, the overall best model had a classification accuracy of 0.615. This can be seen on the bottom row of the heatmap where more than half of the cells are shaded light grey. Interestingly, the light grey of the overall best model corresponds to human subjects that were also correctly classified from at least two of the model pieces. This pattern of classification across the model pieces is consistent with the symbolic discriminant function described in detail above.

A central goal of this study was to identify models of SNP to disease susceptibility relationships that are new, novel and interesting. Our strategy for interpreting the best model was two-fold. First, we performed function mapping (Moore et al. 2007)

Fig. 2.4 Clustered heatmap summarizing whether each best model piece (e.g. $X1 == X6$) or the overall best model correctly classified each subject as a case or a control. Here, each column (blended together) represents a human subject with the cell of the heatmap color coded light grey for a correct classification of disease and dark grey for incorrect

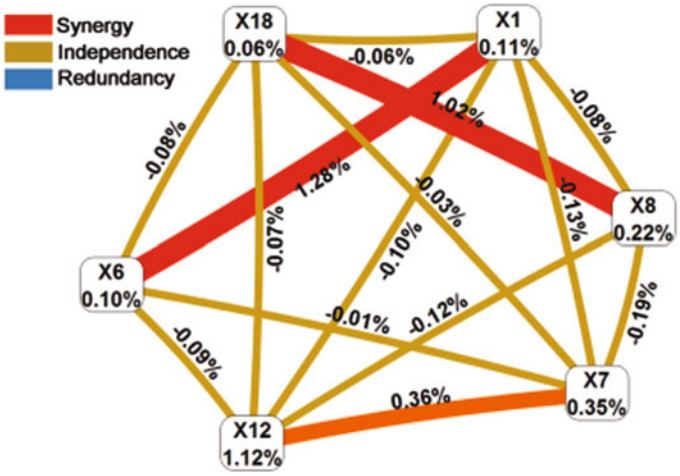


Fig. 2.5 Interaction graph providing a statistical interpretation of individual and pairwise SNP effects on disease susceptibility. The numbers in this graph represent the percentage of entropy or uncertainty about disease status (i.e. class) that is removed by information about the genotypes of individual SNPs (nodes) and the joint effects of pairs of SNPs (edges). Larger positive numbers indicate synergy (*red* and *orange lines*), numbers close to zero indicate independence (*gold lines*) while larger negative numbers indicate redundancy or correlation

to examine the statistical relationships between the individual SNPs and the output of each function in the symbolic discriminant tree. Here, we are primarily interested in the degree of non-additive interaction or synergy between the SNPs and nodes that is associated with disease as measured by interaction information. Second, we performed a biological interpretation by examining the genes closest to the six SNPs identified in the best model. Figure 2.5 shows the result of the synergy analysis in the form of an interaction graph. The numbers in this graph represent the percentage of entropy or uncertainty about disease status (i.e. class) that is removed by information about the genotypes of individual SNPs (nodes) and the joint effects of pairs of SNPs (edges). Larger positive numbers indicate synergy (red and orange lines), numbers

Table 2.1 Most relevant pathology and disease association for the best model according to the SNP and closest gene

Feature	SNP	Closest gene	Relevant pathology	Disease association
X1	rs10915315	AJAP1	neuroblastoma	Migraine, dental caries
X6	rs1266924	PKHD1	Kidney diseases	Glaucoma, weight, prostate cancer
X7	rs936498	HTR1B	Visual cortex	None
X8	rs7738052	MARCKS	Retina development	None
X12	rs2157719	CDKN2B-AS1	Retinal ganglion cells	Glaucoma, aneurysm
X18	rs1489169	CTIF	mRNA processing	Taxane treatment

close to zero indicate independence (gold lines) while larger negative numbers indicate redundancy or correlation. Note that SNP pairs $X8 - X18$ and $X1 - X6$ had evidence of stronger synergistic interaction while $X12 - X7$ had moderate evidence of interaction. Also note that the strongest genetic effect involved the synergistic interaction of the $X1 - X6$ pair. The joint effect of these two SNPs was much stronger than the sum of the effects of the two individual SNPs and stronger than the effect of $X12$ that had the strongest independent effect on POAG susceptibility.

Table 2.1 summarizes the features in the best model, their SNP numbers, the closest gene that they map to, their biological function that might relate to POAG, and prior genetic evidence for their involvement in POAG or any other disease according to the National Human Genome Research Institute (NHGRI) catalog of genome-wide association study (GWAS) results (Welter et al. 2014). Two of the genes are known to be associated with glaucoma from previous large-scale studies. The others represent potential novel findings. We used two functional genomics tools to facilitate biological interpretation of the gene-gene relationships. The first tool we used was Integrative Multi-species Prediction (IMP, imp.princeton.edu) software that brings together experimental data from thousands of sources to infer gene-gene relationships (Wong et al. 2012). This analysis did not reveal any evidence for biological relationships among this set of genes. We then used a modified version of IMP called Genome-scale Integrated Analysis of gene Networks in Tissues (GIANT, giant.princeton.edu) that performs the same analyses using data weighted to specific tissue types. We repeated the analysis using an eye tissue specific network and found that the VEGF gene was the gene most highly connected to these query genes. We assessed the connectivity between the VEGF gene and the genes found by EMERGENT using a permutation test and found that the association was statistically significant ($p = 0.016$). The VEGF gene is a vascular endothelial growth factor involved in the development of new blood vessels and is known to play a role in eye diseases including glaucoma. In fact, VEGF is being actively pursued as a target of new drugs for treating eye diseases such as glaucoma (Horsley and Kahook 2010; Osaadon et al. 2014; SooHoo et al. 2014).

2.5 Summary and Discussion

Human genetics has quickly transitioned into the big data realm with genome-wide association studies or GWAS (Bush and Moore 2012; Hirschhorn and Daly 2005; Wang et al. 2005) and whole-genome sequencing (Dewey et al. 2014) that can routinely measure hundreds of thousands to millions of single nucleotide polymorphisms (SNPs) and other genetic variants across the human genome in large population-based studies of human disease. The goal is to determine which SNPs are associated with disease susceptibility. This serves two purposes. First, we hope to use models of SNP variation to predict who is at risk of developing disease in the future. Second, we hope that the SNPs point to gene regions that might be informative for identifying new drug targets. One approach to modeling the relationship between SNP genotype and disease phenotype is to fit univariate models using parametric statistical methods such as logistic regression. The advantage of this approach is that the results are simple and easy to interpret and can be generated quickly using desktop computing. A major disadvantage is that univariate approaches by design ignore environmental and other genetic factors that likely modulate individual SNP relationships with determining risk (Moore 2003; Moore and Williams 2005, 2009; Moore et al. 2010; Tyler et al. 2009).

Alternatives to the parametric statistical modeling paradigm include machine learning and artificial intelligence. Machine learning methods such as multifactor dimensionality reduction (MDR) have improved power to detect combinations of SNPs that have non-additive or synergistic effects on disease risk (Hahn et al. 2003; Ritchie et al. 2001; Ritchie et al. 2003). These model-free and nonparametric approaches have the potential to reveal genetic effects missed by forcing the data through a preconceived mathematic framework such as regression. Artificial intelligence-based methods take this a step further by providing framework by which the algorithms themselves can learn from the model discovery process. The ultimate goal is to develop computational frameworks that can solve complex modeling problems much as a human would. To this end, we introduced here the Exploratory Modeling for Extracting Relationships using Genetic and Evolutionary Navigation Techniques (EMERGENT) algorithm that uses genetic programming (GP) and, more specifically, a variation called computational evolution (Banzhaf et al. 2006) that provides the complexity that GP needs to learn how to solve hard problems. Our previous work in this area was preliminary with a focus on developing and evaluating different features such as how to incorporate expert knowledge and Pareto optimization. The study presented here is the first full-scale application of EMERGENT to discovering genetic models associated with susceptibility to a complex human disease in big data.

We applied EMERGENT to the genetic analysis of genome-wide genetic data for primary open-angle glaucoma (POAG). We first pre-processed the data using two different machine learning algorithms. First, we pre-computed all pairwise interaction information scores to measure the synergy between SNPs. Second, we applied the ReliefF algorithm to assign weights to each SNP that reflect their independent

and/or synergistic effects. These measures were provided to EMERGENT as expert knowledge to help bias the search toward those SNPs that are likely to have synergistic interactions which scores high on our "interestingness" spectrum (Moore et al. 2013). We then ran EMERGENT thousands of times using a hierarchical design that aggregates results to seed new runs. This produces a small set of final results that can be compared and interpreted. Models were selected from a Pareto optimal front that balanced model complexity, accuracy of the symbolic discriminant classifier and interestingness as measured by interaction information. We were encouraged to see that EMERGENT generated highly consistent models at the highest levels of aggregation each generated by hundreds of independent runs. These models each consisted of six SNPs mapping to respective closest genes. Not only did these models classify well with accuracies above 0.6 but they were modular with simple mathematical relationships that captured complex non-additive relationships in addition to known univariate effects. Biological interpretation using functional genomics data aggregated from thousands of publically-available experiments revealed that each of the genes identified had inferred biological interactions with the VEGF gene that is an actively investigated drug target for treating glaucoma (Horsley and Kahook 2010; Osaadon et al. 2014; SooHoo et al. 2014). Although not experimental proof, these results provide an important layer of evidence that EMERGENT identified a novel set of genes that may influence glaucoma risk through VEGF and its affiliated genetic network.

This study represents one of the first comprehensive artificial intelligence analyses of a genome-wide genetic study of a common complex disease. Our EMERGENT algorithm was able to identify a compact and biologically relevant model of POAG with individual genes that have not previously been directly tied to the disease. Our focus here was on the discovery of new and novel associations. There are several next steps. First, replication of the model, pieces of the model, the genes or the gene pathways in independent data would strengthen their validity. Second, additional biological validation through bioinformatics analysis of existing data and/or directed experimental studies in eye cells or animal models of glaucoma would greatly strengthen the result. Finally, a more thorough investigation of whether the genes identified here represent new drug targets for POAG is needed (Clark and Yorio 2003). Computational approaches like this are needed to provide alternate ways of looking at data and their relationships that complement the parametric statistical approach that has dominated this domain.

Acknowledgements The work on glaucoma was supported by NIH grant EY022300. Aspects of the algorithm development were also supported by NIH grants LM009012, LM010098, and AI59694. Computational infrastructure was partially supported by NIH grants P20 GM103506 and P20 GM103534. 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 Publishers Inc., San Francisco

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

Bush WS, Moore JH (2012) Chapter 11: Genome-wide association studies. PLoS Comput Biol 8(12):e1002822. doi:10.1371/journal.pcbi.1002822

Clark AF, Yorio T (2003) Ophthalmic drug discovery. Nat Rev Drug Discov 2(6):448–459. doi:10.1038/nrd1106

Cooke Bailey JN, Sobrin L, Pericak-Vance MA, Haines JL, Hammond CJ, Wiggs JL (2013) Advances in the genomics of common eye diseases. Hum Mol Genet 22(R1):R59–65. doi:10.1093/hmg/ddt396

Cornelis MC, Agrawal A, Cole JW, Hansel NN, Barnes KC, Beaty TH, Bennett SN, Bierut LJ, Boerwinkle E, Doheny KF, Feenstra B, Feingold E, Fornage M, Haiman CA, Harris EL, Hayes MG, Heit JA, Hu FB, Kang JH, Laurie CC, Ling H, Manolio TA, Marazita ML, Mathias RA, Mirel DB, Paschall J, Pasquale LR, Pugh EW, Rice JP, Udren J, van Dam RM, Wang X, Wiggs JL, Williams K, Yu K (2010) The gene, environment association studies consortium (geneva): maximizing the knowledge obtained from gwas by collaboration across studies of multiple conditions. Genet Epidemiol 34(4):364–372. doi:10.1002/gepi.20492

Dewey FE, Grove ME, Pan Cea (2014) Clinical interpretation and implications of whole-genome sequencing. JAMA 311(10):1035–1045. doi:10.1001/jama.2014.1717. http://dx.doi.org/10.1001/jama.2014.1717/data/Journals/JAMA/929845/joi140017.pdf

Fan R, Zhong M, Wang S, Zhang Y, Andrew A, Karagas M, Chen H, Amos CI, Xiong M, Moore JH (2011) Entropy-based information gain approaches to detect and to characterize gene-gene and gene-environment interactions/correlations of complex diseases. Genet Epidemiol 35(7):706–721. doi:10.1002/gepi.20621

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

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, ACM, New York, NY, GECCO '09, 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, pp 19–36

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, pp 19–36

Hahn LW, Ritchie MD, Moore JH (2003) Multifactor dimensionality reduction software for detecting gene-gene and gene-environment interactions. Bioinformatics 19(3):376–382

Hirschhorn JN, Daly MJ (2005) Genome-wide association studies for common diseases and complex traits. Nat Rev Genet 6(2):95–108. doi:10.1038/nrg1521

Horn J, Nafpliotis N, Goldberg DE (1994) A niched pareto genetic algorithm for multiobjective optimization. In: Evolutionary Computation, 1994. IEEE world congress on computational intelligence, proceedings of the first IEEE conference on, pp 82–87 vol.1. doi: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, ACM, New York, NY, GECCO '06, pp 815–822. doi:10.1145/1143997.1144142

Horsley MB, Kahook MY (2010) Anti-vegf therapy for glaucoma. Curr Opin Ophthalmol 21(2):112–117. doi:10.1097/ICU.0b013e3283360aad

Hu T, Sinnott-Armstrong NA, Kiralis JW, Andrew AS, Karagas MR, Moore JH (2011) Characterizing genetic interactions in human disease association studies using statistical epistasis networks. BMC Bioinform 12:364. doi:10.1186/1471-2105-12-364

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. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262111705

Mailman MD, Feolo M, Jin Y, Kimura M, Tryka K, Bagoutdinov R, Hao L, Kiang A, Paschall J, Phan L, Popova N, Pretel S, Ziyabari L, Lee M, Shao Y, Wang ZY, Sirotkin K, Ward M, Kholodov M, Zbicz K, Beck J, Kimelman M, Shevelev S, Preuss D, Yaschenko E, Graeff A, Ostell J, Sherry ST (2007) The ncbi dbgap database of genotypes and phenotypes. Nat Genet 39(10):1181–1186. doi:10.1038/ng1007-1181

Moore J (2003) The ubiquitous nature of epistasis in determining susceptibility to common human diseases. Human Heredity 56:73–82

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

Moore JH, Williams SM (2005) Traversing the conceptual divide between biological and statistical epistasis: systems biology and a more modern synthesis. Bioessays 27(6):637–646. doi:10.1002/bies.20236

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

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, Barney N, Tsai CT, Chiang FT, Gui J, White BC (2007) Symbolic modeling of epistasis. Hum Hered 63(2):120–133. doi:10.1159/000099184

Moore JH, Andrews PC, Barney N, White BC (2008a) 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, Springer, lecture notes in computer science, vol 4973, pp 129–140

Moore JH, Barney N, White BC (2008b) Solving complex problems in human genetics using genetic programming: the importance of theorist-Practitionercomputer interaction. Springer US, pp 69–85

Moore JH, Greene CS, Andrews PC, White BC (2008c) Does complexity matter? artificial evolution, computational evolution and the genetic analysis of epistasis in common human diseases. In: Riolo RL, Soule T, Worzel B (eds) Genetic programming theory and practice VI, genetic and evolutionary computation. Springer, Ann Arbor, pp 125–145. doi:10.1007/978-0-387-87623-8-9

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, pp 153–171. doi:10.1007/978-1-4614-1770-5-9

Moore JH, Hill DP, Sulovari A, Kidd L (2012) Genetic analysis of prostate cancer using computational evolution, pareto-optimization and post-processing. In: Riolo R, Vladislavleva E, Ritchie MD, Moore JH (eds) Genetic programming theory and practice X, genetic and evolutionary computation. Springer, Ann Arbor, pp 87–101. doi:10.1007/978-1-4614-6846-2-7. http://dx.doi.org/10.1007/978-1-4614-6846-2–7

Moore JH, Hill DP, Saykin A, Shen L (2013) Exploring interestingness in a computational evolution system for the genome-wide genetic analysis of alzheimer's disease. In: Riolo R, Moore JH, Kotanchek M (eds) Genetic programming theory and practice XI, genetic and evolutionary computation. Springer, pp 31–45. doi:10.1007/978-1-4939-0375-7-2

Osaadon P, Fagan XJ, Lifshitz T, Levy J (2014) A review of anti-vegf agents for proliferative diabetic retinopathy. Eye (Lond) 28(5):510–520. doi:10.1038/eye.2014.13

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) Exploitation of linkage learning in evolutionary algorithms. Springer. (chap 10: Sensible initialization of a computational evolution system using expert knowledge for epistasis analysis in human genetics), pp 215–226

Ritchie M, Hahn L, Roodi N, Bailey L, Dupont W, Parl F, Moore J (2001) Multifactor dimensionality reduction reveals high-order interactions among estrogen metabolism genes in sporadic breast cancer. Am J Hum Genetics 69:138–147

Ritchie MD, White BC, Parker JS, Hahn LW, Moore JH (2003) Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases. BMC Bioinform 4:28. doi:10.1186/1471-2105-4-28

SooHoo JR, Seibold LK, Kahook MY (2014) The link between intravitreal antivascular endothelial growth factor injections and glaucoma. Curr Opin Ophthalmol 25(2):127–133. doi:10.1097/ICU.0000000000000036

Spector L, Robinson A (2002) Genetic programming and autoconstructive evolution with the push programming language. Genet Program Evolvable Mach pp 7–40

Tyler AL, Asselbergs FW, Williams SM, Moore JH (2009) Shadows of complexity: what biological networks reveal about epistasis and pleiotropy. Bioessays 31(2):220–227. doi:10.1002/bies.200800022

Wang WYS, Barratt BJ, Clayton DG, Todd JA (2005) Genome-wide association studies: theoretical and practical concerns. Nat Rev Genet 6(2):109–118. doi:10.1038/nrg1522

Wang X, Harmon J, Zabrieskie N, Chen Y, Grob S, Williams B, Lee C, Kasuga D, Shaw PX, Buehler J, Wang N, Zhang K (2010) Using the utah population database to assess familial risk of primary open angle glaucoma. Vision Res 50(23):2391–2395. doi:10.1016/j.visres.2010.09.018

Welter D, MacArthur J, Morales J, Burdett T, Hall P, Junkins H, Klemm A, Flicek P, Manolio T, Hindorff L, Parkinson H (2014) The nhgri gwas catalog, a curated resource of snp-trait associations. Nucleic Acids Res 42(Database issue):D1001–6. doi:10.1093/nar/gkt1229

Wong AK, Park CY, Greene CS, Bongo LA, Guan Y, Troyanskaya OG (2012) Imp: a multi-species functional genomics portal for integration, visualization and prediction of protein functions and networks. Nucleic Acids Res 40(Web Server issue):W484–90. doi:10.1093/nar/gks458

Jason H. Moore is the Third Century Professor of Genetics and of Community and Family Medicine at the Geisel School of Medicine of Dartmouth College where he serves as founding Director of the Institute for Quantitative Biomedical Science, Associate Director of the Norris-Cotton Cancer Center and Associate Director of the SYNERGY Clinical and Translational Science Institute. Dr. Moore has published more than 350 peer-reviewed papers, editorials and book chapters. His research program focuses on the development and evaluation of computational methods for the

genetic analysis of common human diseases. His work crosses artificial intelligence, machine learning and statistics. He is an elected Fellow of the American Association for the Advancement of Science (AAAS) and a Kavli Fellow of the National Academy of Sciences. He is the founding Editor-in-Chief of the journal BioData Mining

Casey S. Greene is an Assistant Professor of Genetics and member of the Institute for Quantitative Biomedical Sciences at the Geisel School of Medicine of Dartmouth College, Hanover, NH .

Douglas P. Hill is a software engineer in the Institute for Quantitative Biomedical Sciences at Dartmouth Medical School, USA.

Chapter 3
Inheritable Epigenetics in Genetic Programming

William La Cava and Lee Spector

3.1 Introduction

In 1815, Lamarck postulated that organisms acquired adaptations from their environments during their lifetime and these adaptations were passed along to the offspring they produced. Four decades later, Darwin showed evidence that organisms instead evolve traits over millions of years through a combination of random mutation, natural selection, and the transmission of genetic information to their progeny, rather than through lifetime adaptations (Darwin 1872). Four decades after that, Baldwin reconciled the two ideas by asserting that despite the inheritance of purely genetic material, adaptive changes during an organism's life affected selection pressures and thus reproductive dynamics, ensuring that the emergent adaptive ability of a genotype could be selected for without the information explicitly being transferred in addition to DNA.

The subsequent discovery of somatic and germ cell architecture and the biological mechanisms for genetic inheritance caused Lamarck's ideas to be discredited in biology. This did not stop researchers from studying the incorporation of Lamarckism and Baldwinian evolution into genetic algorithms (GAs) (Gruau and Whitley 1993; Whitley et al. 1994; Ross 1999; Giraud-Carrier 2002). In Lamarckian evolution for GAs, a local search mechanism is implemented to update the population genomes each generation. The Baldwin effect is achieved using the same local search to update the population fitness landscape, but the actual genotype changes are discarded. Whitley showed that both methods were quite useful for improving results for function optimization problems (Whitley et al. 1994). He noted that Lamarckism tended

W. La Cava (✉)
Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA USA
e-mail: wlacava@umass.edu

L. Spector
School of Cognitive Science, Hampshire College, Amherst, MA 01002-3359, USA
e-mail: lspector@hampshire.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_3

to improve the speed of convergence but for some cases became stuck in the same local optima as the standard GA, whereas the Baldwinian evolution strategy was better able to find global optima.

Various Lamarckian updating methods have been implemented for genetic programming (GP) as well, such as equation tree snipping (Bongard and Lipson 2007), reinforcement learning (Mingo and Aler 2007), and parameter updating (Iba 2008; Topchy and Punch 2001). In symbolic regression in particular, concurrent parameter updating, either by stochastic hill climbing or gradient methods, is common due to the vast floating-point search space, and it has been shown to improve convergence on a number of benchmark symbolic regression problems (Kommenda et al. 2013).

Researchers have been able to exploit the fact that, unlike in nature, little or no phenotypic-genotypic mapping has to occur in the computational scheme since the system being evolved is either identical to the genotype or the mapping of phenotypic traits to genotype is straightforward. Previously it was assumed that the Lamarckian implementation was extra-biological because phenotypic adaptations in nature did not have a known physical mechanism for influencing their genotypic origins (Ross 1999). Today, however, physical mechanisms are known to exist and have been demonstrated in many studies. The studies constitute the growing field of epigenetics, a term that refers broadly to the ways in which gene expressions are regulated and inherited (Jablonka and Lamb 2002; Holliday 2006). Recent studies have not only shown that environmental factors influence gene expression in organisms (Dias and Ressler 2013), but also that epigenetic mechanisms may be inheritable (Turner 2000; Kaati et al. 2002; Pogribny et al. 2004; Dias and Ressler. 2014).

We present a GP method that captures this understanding of epigenetics as a layer of environmentally influenced, evolving gene regulation that interacts with the genotype to produce the phenotype. This system captures the advantages of Lamarckian updating without changing the genotype, and yet preserves inheritable phenotypic improvements in offspring, unlike Baldwinian evolution.

3.2 Background

3.2.1 GP Representation

Biological systems benefit from transmitting and evolving structurally complex systems at the more flexible genome level. The key to preserving this flexibility in GP is to create a mapping from the genotype, i.e. the computer encoding, to the phenotype, e.g. the candidate equation[1]. In this way the genotypic search can be free to vary through evolutionary processes and still produce constrained phenotypes. As shown in Ryan (1996), lower constraints on evolutionary search generally improve results. This is the goal of developmental GP approaches such as genotype-phenotype mapping and gene expression programming (Banzhaf 1994; Ferreira 2001).

[1] Note that these definitions distinguish between the program, the resulting equation, and its fitness, unlike in traditional GP.

Classical GP (Koza 1992) represents individuals using tree structures, and most practitioners of symbolic regression follow this trend today, as White discovered in his community survey (White et al. 2012). In addition to tree representations, other methods have been proposed, for example stack-based GP (Salman et al. 1985; Spector 2001; Ferreira 2001), linear GP and directed acyclic graphs (Brameier and Banzhaf 2007; Schmidt and Lipson 2007), tree adjunct grammars (Hoai et al. 2002), and Cartesian GP (Miller and Thomson 2000). Despite the succinct representation of nested equations that tree structures provide, they have disadvantages when applied to evolutionary computation. For example, the tree structure makes it difficult to deliver uniform variation among instructions because of the effect that the size and shape of the trees have on the probability of change wrought by standard mutation and crossover. Whereas in nature chromosomal crossover occurs on homologous or nearly homologous sections of chromosomes between parents in a somewhat (but not purely) uniform manner, standard GP crossover consists of the swapping of random subtrees between parents. In standard GP, subtree mutation also occurs at a random node location. Therefore the probability of mutation and crossover for terminals and nodes in a tree is a positive function of its depth. With linear GP, this probability can be made more uniform. Motivated by evidence that uniformity improves evolutionary search (Page et al. 1999), the ULTRA operator (Spector and Helmuth 2013) was developed that converts nested expressions into linear representations, applies quasi-uniform crossover, repairs the program parentheses, and converts the representation back into a tree structure. The need for the 'R' (signifying repair) in the ULTRA operator highlights a second disadvantage of tree representations: random manipulation of tree structures at the level of instructions and literals can easily make them syntactically invalid, and therefore controls must be in place to ensure that operations such as mutation, crossover and initialization result in executable programs.

In our work we present a developmental linear genetic programming tool that evolves equations with an instruction set that is syntax-free with respect to program validity by using the principles of stack-based, post-fix equation encoding with all instruction ordering constraints removed. In this way, genotypes are unconstrained except by initial size during the run, and this opens the door for our investigation of epigenetics. In most GP systems, it is possible for distinct genotypes to result in the same phenotype, resulting in higher search flexibility. By applying epigenetic logic to the genotype, we are able to achieve the reverse as well: different phenotypic expression from identical genotypes. This could provide a path for increased diversity in a population and avoid premature convergence related to bottle-necking, i.e. genotypic convergence. We will investigate how this affects the performance of genetic programming in symbolic regression. The proposed epigenetic methods can also be thought of as a new local search extension to GP that provides a mechanism for modifying equation form by affecting phenotypic development.

3.2.2 *Epigenetics*

There are two main mechanisms by which epigenetics take place in mammalian DNA: cytosine methylation (Jones and Takai 2001) and histone modification (Turner 2000). Methylation provides a mechanism for silencing portions of genetic code, and as such is able to determine whether certain genes are expressed during transcription. Histones, meanwhile, are structures around which base pairs of DNA are wrapped, and their modification affects the winding of these base pairs, which provide an epigenetic mechanism by affecting the accessibility of genes for transcription.

These processes result in clusters of genetic material that are not expressed in the phenotype (a.k.a. non-coding segments or introns). Introns are studied in the genetic programming world, and initially were linked to bloat, a phenomenon in which programs grow very large, become resistant to behavioral change, and drain computer resources without improvement. However, there have been several studies on the effects of non-coding segments in evolutionary algorithms (EAs) that find that moderate levels of introns can improve EA performance by reducing the destructive effect of crossover operations while maintaining blocks of effective code (Nordin et al. 1995; Brameier and Banzhaf 2007). In Nordin et al. (1995) and Brameier and Banzhaf (2007), introns are either explicitly declared in place of genes or defined as such afterwards by observing their behavior. In our work, we instead impose an epigenetic condition on each instruction that functions as an on/off switch to determine whether or not the instruction will execute within the genotype. Therefore, silenced genes do not affect computation time during genotype to phenotype conversion, but are present as structural elements during mutation and crossover. Epigenetic activation and silencing is learned each generation using a stochastic hill climber and co-evolves with the corresponding genotypes.

There has been work to simulate epigenetics in GP: Tanev (Tanev and Yuta 2008) developed a genetic programming approach that simulated histone modification for use in a predator-prey problem. This approach did not include inheritance of epigenetic properties during evolution, which Tanev considered to be unrealistic at the histone level. Studies of artificial ontogeny (Bongard and Pfeifer 2001; Fontana 2011) encompass some of the aspects of epigenetics, particularly the evolution of phenotype control from a decoupled genotype. Still, an epigenetic approach has not been presented that attempts to model two salient characteristics: its ability to update based on environmental changes, and its ability to be inherited.

We make the assumption that epigenetic traits are inheritable, and focus on doing so in a generic way that can be readily applied to any genetic programming system. While the method is generic, a flat, syntax-free representation like the one presented in this paper is advantageous for epigenetic switching because genotypic regulation can be applied uniformly and easily since it does not require syntactic repair. With other representations, the probability of epigenetic modifications may not be uniform, and they may have to undergo a repair step to guarantee execution.

Many researchers in the field of GP see the incorporation of meta-genetic biological functions as an open issue in GP, with O'Neill et al. noting that "... it can be safely predicted that epigenetic effects will be important if GP will adopt development as a scalability mechanism." (ONeill et al. 2010)

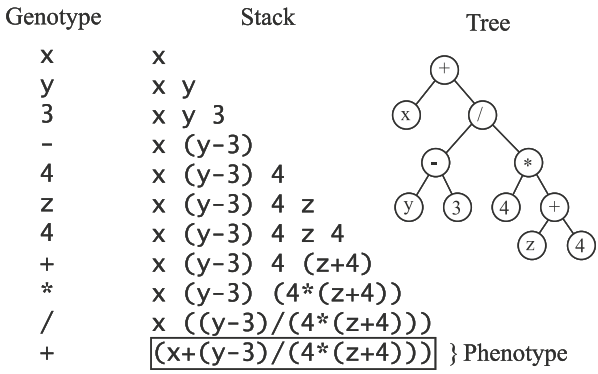


Fig. 3.1 Example encoding of $x + \frac{y-3}{4(z+4)}$ with the steps of stack execution and equivalent tree representation

3.3 Methods

3.3.1 Developmental Linear Genetic Programming

The GP system created for this research is called Develep and the source code is available online (La Cava 2014a). We represent programs as linear genotypes written in Reverse Polish Notation (RPN) or "post-fix" notation, as demonstrated in Figure 3.1. The implementation functions by pushing and pulling floating point numbers on and off of the stack. For instance, a number or variable instruction will push a floating point value to the stack, while a binary (i.e. arity 2) operation will pull two numbers off of the stack, perform its operation on them, and push a new value back to the stack. This basic representation scheme is used in other GP systems (Salman et al. 1985; Ferreira 2001; Spector and Robinson 2002).

In order to guarantee execution safety in the context of arbitrary execution order, the instructions are specialized to handle situations in which the stack does not have the correct number of elements for an operation to occur. For example, binary functions are ignored if the stack is not at least two elements long. At the end of genotype execution, the top element of the stack constitutes the equation, the phenotype, that is then used for fitness assessment. Therefore execution is safe with respect to the resultant stack length. We distinguish between the genotype, that builds the stack, and the phenotype, that is the top element of the resultant stack.

3.3.2 Epigenesis

3.3.2.1 Epigenetic Hill Climber

Epigenesis is achieved by creating binary switches associated with the instructions comprising the genotypes of each individual, and the array of these switches is referred to as an epiline. During genotype execution, only instructions from the genotype with a true value in the corresponding epiline are executed. Individuals can

be initialized with any combination of active and inactive epiline values, and each generation the population undergoes one iteration of epigenetic hill climbing (EHC) to optimize the epiline.

```
gen ⟵ genotype of individual
epi ⟵ epigenetic line of individual
phen ⟵ phenotype equation of individual
L ⟵ length of genotype
sr ⟵ switching rate
for number of hill climbing iterations do
    epiTemp ⟵ epi
    for i ∈ L do
        if rand() < sr then
            /* flip the on/off state of marked indices in the
               epiline                                          */
            epiTemp(i) ⟵ !epi(i)
        end
    end
    /* get equation from genotype with updated gene expression
       */
    phenTemp ⟵ GenToPhen(gen,epiTemp)
                                          /* update equation   */
    if fitness(phenTemp) < fitness(phen) then
        epi ⟵ epiTemp
        phen ⟵ phenTemp
                                          /* secondary size metric */
    else if fitness(phenTemp) == fitness(phen) and size(phenTemp) < size(phen) then
        epi ⟵ epiTemp
        phen ⟵ phenTemp
    else
    end
end
```

Algorithm 1 shows the EHC algorithm for updating the epigenetic properties of an individual in the population. During the hill climbing process, a small percentage of epiline values are switched on or off and the genotype is re-executed with the new epiline. For example, compare the phenotype of Fig. 3.1 to that generated with epigenetic switching in Fig. 3.2. A different equation can be expressed by an identical genotype in this way. The resulting equation is evaluated and kept (along with the new epiline) if it results in a better fitness for the individual, or if it results in a smaller equation size without changing the fitness. Equation size is measured by the number of characters in the equation string. Using string length is a simple but crude measure of equation size, and while more sophisticated measures could be used, calculating string length is very lightweight and inherently penalizes inefficient equation representations, such as having lots of nesting or many separate constants in the phenotype.

The EHC is applied after fitness evaluation and before selection. Figure 3.3 shows the implementation within the Develep instruction set. After an initial fitness

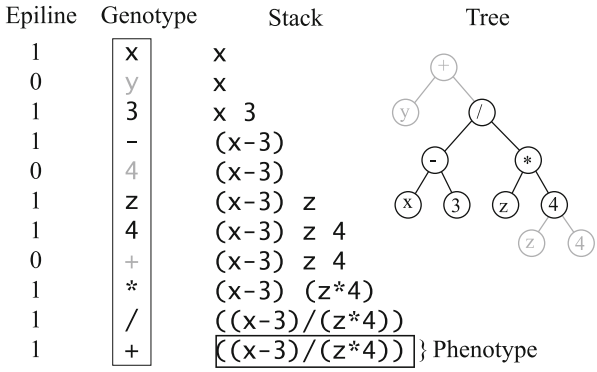


Fig. 3.2 Epigenetics added to the original encoding of $f = x + \frac{y-3}{4(z+4)}$, which in this example results in $f = \frac{x-3}{(z*4)}$ Note that this requires the replacement of the '+' operator in the tree representation with the 4 operand. Thus a repair step would be required for tree implementation

evaluation, one iteration of epigenetic hill climbing is applied. The better fitness value and corresponding epiline and phenotype are kept. The EHC is therefore a local optimization scheme within the scope of genetic material already available to the individual. It can both decrease and increase the expressed length of the genotype. Silenced genes reduce the computational effort of developing the phenotype and evaluating the fitness of the genotype by lowering the number of point evaluations.

3.3.2.2 Evolution of Epiline

The epiline values for an individual are connected to specific genes. During crossover, the child inherits its parents' genes along with the genes' epigenetic states. In the case of swap mutation, the new gene's epigenetic state is chosen probabilistically from the chosen percent of active genes in the initial population (i.e. if the population is

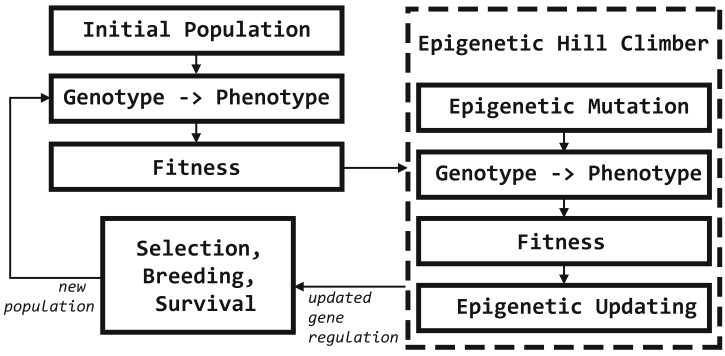


Fig. 3.3 Block diagram of developmental epigenetic programming. After fitness evaluation and before selection, the population undergoes an iteration of epigenetic hill climbing, represented by the *dotted lines* on the *right*. The population then undergoes the typical process of selection, breeding and survival to produce the next generation, and the process is repeated, as shown on the *left*

initialized to be 50 % active, a new gene mutation will have a 50 % chance of being active as well).

It should be noted that modeling the epigenetic inheritance after genetic inheritance is biologically questionable. While it is generally agreed that there are sets of imprinted genes whose epigenetic states are inherited in offspring, there is a reprogramming stage in embryonic development during which many epigenetic values are erased and reapplied. The way that this reprogramming functions and whether it is itself inherited is not well understood. We assume genotypic analog for epigenetic inheritance in this work due to the ease with which we can embed the epigenesis in our representation, and the fact that these values go through a small amount of reprogramming thanks to the EHC.

3.3.2.3 Evolutionary Parameters

Crossover and mutation are used as genetic operators, with a 90 % rate of crossover and 10 % rate of mutation. During crossover, a random point is picked in each parent, and the tails of the two parents are swapped (a.k.a. one-point crossover). The mutation operator is a pointwise operator inspired by Bongard's hill climber (Bongard and Lipson 2007). It can change or delete a randomly selected gene. Whereas all other instructions are replaced randomly from the instruction set, operand instructions (e.g. x, y, 1.73, etc.) are mutated by changing the argument they contain. If the argument is a variable, it is swapped for a randomly picked one. If the argument is a constant, the constant is adjusted by 0 mean Gaussian noise with standard deviation equal to half the constant magnitude or replaced with a random constant, with equal probability. Genes are selected for mutation with a 10 % probability. This probability may seem high for a standard mutation operator, but due to the typing constraints described above, is believed to be appropriate.

We use one of two modern evolutionary methods in our trials: deterministic crowding (DC) or age-fitness Pareto survival (AFP). In deterministic crowding (Mahfoud 1995), children replace the parent with the smallest phenotypic Levenshtein distance if and only if they have a lower fitness than the parent. In age-fitness Pareto survival (Schmidt and Lipson 2011), each individual has an age equal to the number of generations its oldest genes have been in the population. Each generation, a whole separate population of offspring and one new individual are created. These individuals compete with the current population in a tournament of size two that compares age and fitness dominance, and the winners survive to the next generation. In both DC and AFP, parents are randomly selected to produce children.

3.4 Applications

We apply the genetic programming system to a two-state differential equation problem and two benchmark symbolic regression problems. The first two problems consist of the partitioned (Bongard and Lipson 2007) states of the Lotka-Volterra interspecies competition model, defined as

Table 3.1 Runtime settings

Setting	Lotka-volterra	Pagie-1	Nguyen-7
Terminal set	{+, −,*,/,R[−1.0,1.0],x,y}	{+, −,*,/,1.0,x}	{+, −,*,/,exp,log,1.0,x}
Initial program length	[3, 50]	[10, 100]	[10, 100]
Method	DC[a]	AFP[b]	AFP[b]
Pop size	1000	1000	1000
Max generations	5000	5000	5000
Initial % active genes	100	50	50

Brackets indicate ranges of values picked uniformly
R ephemeral random constants
[a]Deterministic Crowding
[b]Age-Fitness Pareto Survival

$$\dot{x} = 3x - 2xy - x^2 \tag{3.1}$$

$$\dot{y} = 2y - xy - y^2 \tag{3.2}$$

The benchmark symbolic regression were chosen from the set of recommended problems emerging from a GP community survey (White et al. 2012). We used the Pagie-1 and Nguyen-7 problems. The Pagie-1 problem is a two-variable system defined as

$$f(x, y) = \frac{1}{1 + x^{-4}} + \frac{1}{1 + y^{-4}} \tag{3.3}$$

The Nguyen-7 problem is defined by the single-variable equation

$$f(x) = log(x + 1) + log(x^2 + 1) \tag{3.4}$$

The settings for all the trials are shown in Table 3.1. The differential equation problems were run using deterministic crowding, whereas the benchmark problems were solved using age-fitness Pareto survival.

Ephemeral random constants were included in the terminal set for the Lotka-Volterra problems, and the Nguyen-7 addded the exp and log functions. For Lotka-Volterra, all genes were active in the initial population. After running a parameter variation study that showed higher rates of beneficial crossover with a mix of active and inactive genes in the inital population (La Cava et al. 2014b), a starting value of 50% active genes was used for the Pagie-1 and Nguyen-7 problems.

We measured a successful run as one achieving

$$\sum_{i=1}^{n} |y^*(i) - \hat{y}(i)| < 0.0001$$

where y^* is the target output, $\hat{y}$ is the equation output, and n is the number of data points. The fitness metric F used during the runs includes the coefficient of determination R^2 and is defined as

$$F = \frac{\frac{1}{n}\sum_{i=1}^{n} |y^*(i) - \hat{y}(i)|}{R^2} \tag{3.5}$$

Table 3.2 Performance comparisons

Problem	Trials	Method	Success rate (%)	MBF	Point evaluations per fitness case	Mean effective size
Lotka-Volterra $\dot{x}$	50	DLGP	100	0.000	*2.7430E07*	*29.63*
	50	DLGP+EHC	100	0.000	*2.0655E07*	*24.69*
Lotka-Volterra $\dot{y}$	50	DLGP	100	0.000	2.5763E07	*30.3595*
	50	DLGP+EHC	100	0.000	2.2529E07	*25.1297*
Pagie-1	100	DLGP	*13*	*0.086*	*3.8605E08*	*68.7337*
	100	DLGP + EHC	*27*	*0.054*	*3.8678E08*	*40.3222*
Nguyen-7	50	DLGP	*72*	*0.0021*	*2.2360E08*	*68.97*
	50	DLGP+EHC	*100*	*0.000*	*2.2781E07*	*20.2493*

Results in *italic* are significant to $p < 0.05$, where p is the non-parametric ranked t-test. (Wineberg and Christensen 1994)

where

$$R^2 = \frac{cov(y^*, \hat{y})^2}{var(y^*)var(\hat{y})} \tag{3.6}$$

For these examples, every successful run was an exact match to the target equation form, with parameters exact to floating-point zero.

A parameter hill climber was used once per generation on each individual in order to perform local search of constant values in the Lotka-Volterra problems. The parameter hill climber perturbs each constant value in the expressed genotype with 0 mean Gaussian noise with standard deviation equal to 10 % of the magnitude of the constant, and keeps the changes if they improve fitness. Likewise, the EHC was run for one iteration each generation with a switching rate of 10 %.

3.5 Results and Discussion

The results are summarized in Table 3.2. We compare results by success rate (exact solution), mean best fitness (MBF), mean number of total point evaluations in the trials, and mean effective program length (number of active instructions) during the trial. The EHC increased success rate on the Pagie-1 problem from 13 to 27 %, and increased success rate on the Nguyen-7 problem from 72 to 100 %; meanwhile, standard DLGP and DLGP+EHC both solved the Lotka-Volterra problem 100 % of the time.

Despite increasing the number of fitness evaluations per generation by implementing the EHC, we found that the total number of point evaluations during a run actually decreased for most problems, with statistically significant decreases in the Lotka-Volterra and Nguyen-7 problem. This is due to a combination of improved success rate (for Nguyen-7) and the lower effective program length. For the Pagie-1

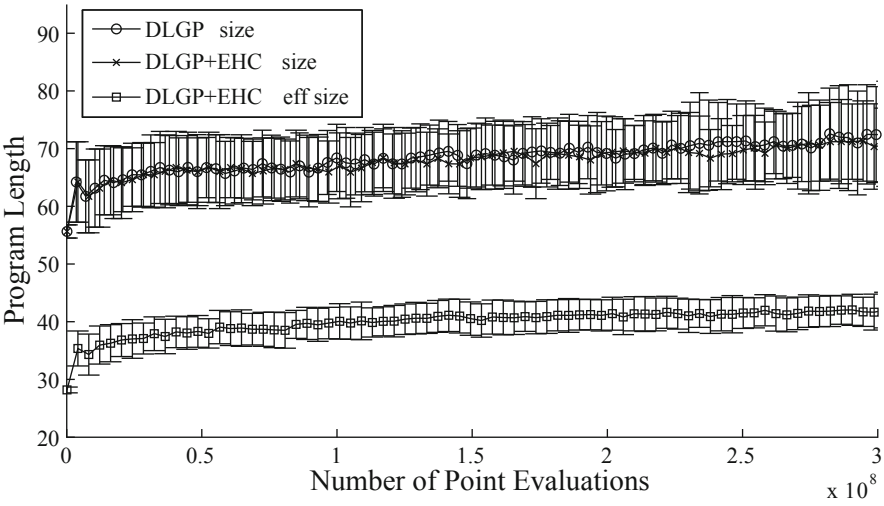


Fig. 3.4 Program size comparison during evolution of Pagie-1, averaged over the trials of each method. Both total and effective length are shown for the regular *DLGP* method and the additional *EHC* implementation

problem there was approximately a 2 % increase in total point evaluations, likely due to the fact that most trials went to the maximum generations without finding a solution. While it is difficult to make a direct comparison to previously published results due to the variability in experimental setup and termination criteria, it is noted that most algorithms do not find (or do not report) as many exact solutions to the Pagie-1 problem (Pagie and Hogeweg 1997; Kommenda et al. 2013; Spector and Helmuth 2013) [eg] or the Nguyen-7 problem (Uy et al. 2011; Krawiec and Pawlak, 2013), especially when using "standard" GP.

We also found that the EHC provided good size control during evolution, as shown in Figs 3.4 and 3.5. While the total genotypic size remains similar to DLGP, with EHC activated the effective size of the programs was 17 % shorter for each Lotka-Volterra state, 41 % shorter for Pagie-1, and 70 % shorter for Nguyen-7. For the Lotka Volterra cases, the smaller initial program sizes and 0 % inactive genes may explain why the size difference is less pronounced. Furthermore, as Fig. 3.5 shows, the EHC addition resulted in final solutions with less bloat, meaning that the expressed genotype of the solution was closer to the smallest possible size by which the exact solutions could be represented.

3.6 Conclusion

We have demonstrated a straightforward way to incorporate some of the key features of epigenetics into linear genetic programming in order to improve symbolic regression performance. We represented two characteristics of epigenesis in this

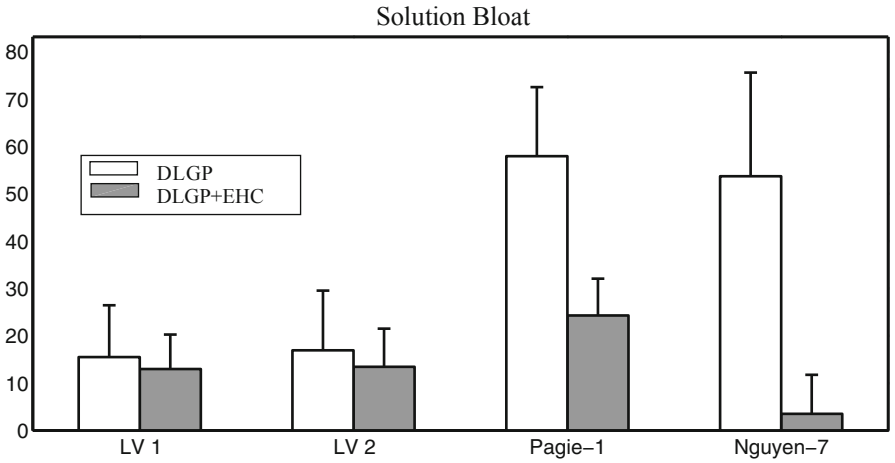


Fig. 3.5 Comparison of solution bloat, which is the difference in size of the solution program and the smallest possible solution program for the exact solution

implementation: (1) dependence on environmental factors by use of the EHC, and (2) inheritability by evolution of epilines with their corresponding genotypes. Unlike previous methods, our system allows offspring to inherit both the learned phenotypic traits of their parents as well as the genotypic underpinning. With this system we demonstrate higher success rates and lower solution bloat for a number of symbolic regression problems, with equivalent or lower computational effort required. This suggests that the epigenetic implementation is able to capitalize on the benefits of Lamarckism (fast convergence) and Baldwinian evolution (finding global optima). We hope this work will provide the basis for further investigation into how epigenetic learning and evolution can interact to improve genetic programming for many applications. Namely, further work should address various levels of epigenetic inheritability, as well as the contributions of environmental factors or inheritance to the improvement in success.

Acknowledgements The authors would like to thank Thomas Helmuth for his insightful feedback and Professor Kourosh Danai for his support of this research, as well as the members of the Hampshire Computational Intelligence Laboratory. This work is partially supported by the NSF-sponsored IGERT: Offshore Wind Energy Engineering, Environmental Science, and Policy (Grant Number 1068864), as well as Grant No. 1017817. 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

Banzhaf W (1994) Genotype-phenotype-mapping and neutral variationa case study in genetic programming. Parallel problem solving from naturePPSN III. Springer, p 322–332. http://link.springer.com/chapter/10.1007/3-540-58484-6-276

Bongard J, Lipson H (2007) Automated reverse engineering of nonlinear dynamical systems. Proc Natl Acad Sci USA 104(24):9943–9948. http://www.pnas.org/content/104/24/9943.short

Bongard JC, Pfeifer R (2001) Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. Proceedings of the genetic and evolutionary computation conference, p 829–836. http://www.cems.uvm.edu/jbongard/papers/geccoBongard2001.pdf

Brameier M, Banzhaf W (2007) Linear genetic programming, vol 1, 1st edn. Springer Berlin

Darwin C (1872) The origin of species by means of natural selection: or, the preservation of favoured races in the struggle for life and the descent of man and selection in relation to sex. Modern Library, London

Dias BG, Ressler KJ (2013) PACAP and the PAC1 receptor in post-traumatic stress disorder. Neuropsychopharmacology 38(1):245–246. doi:10.1038/npp.2012.147. http://www.nature.com/npp/journal/v38/n1/full/npp2012147a.html

Dias BG, Ressler KJ (2014) Parental olfactory experience influences behavior and neural structure in subsequent generations. Nat Neurosci 17(1):89–96. doi:10.1038/nn. http://www.nature.com/neuro/journal/v17/n1/full/nn.3594.html

Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13(2):87–129. arXiv:cs/0102027. http://arxiv.org/abs/cs/0102027

Fontana A (2011) Epigenetic tracking: biological implications. In: Kampis G, Karsai I, Szathmry E (eds) Advances in artificial life. Darwin Meets von Neumann, no. 5777 in Lecture notes in computer science. Springer, Berlin, pp 10–17. http://link.springer.com/chapter/10.1007/978-3-642-21283-3-2

Giraud-Carrier C (2002) Unifying learning with evolution through baldwinian evolution and lamarckism. In Advances in Computational Intelligence and Learning (pp. 159–168). Springer Netherlands

Gruau F, Whitley D (1993) Adding learning to the cellular development of neural networks: evolution and the baldwin effect. Evolut Comput 1(3):213–233. doi:10.1162/evco.1993.1.3.213. http://dx.doi.org/http://dx.doi.org/10.1162/evco.1993.1.3.213

Hoai NX, McKay RI, Essam D, Chau R (2002) Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. Evolutionary computation, 2002. CEC'02. Proceedings of the 2002 Congress on, IEEE, vol 2, p 1326–1331. http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber=1004435

Holliday R (2006) Epigenetics: a historical overview. Epigenetics 1(2). http://www.landesbioscience.com/journals/epigenetics/holliday EPI1-2.pdf

Iba H (2008) Inference of differential equation models by genetic programming. Inf Sci 178(23):4453–4468. doi:10.1016/j.ins.2008.07.029. (special Section: Genetic and Evolutionary Computing)

Jablonka E, Lamb MJ (2002) The changing concept of epigenetics. Ann NY Acad Sci 981(1):82–96. http://onlinelibrary.wiley.com/doi/10.1111/j.1749-6632.2002.t b04913.x/full

Jones PA, Takai D (2001) The role of DNA methylation in mammalian epigenetics. Science 293(5532):1068–1070. doi:10.1126/science.1063852. http://www.sciencemag.org/content/293/5532/1068, PMID: 11498573

Kaati G, Bygren LO, Edvinsson S (2002) Cardiovascular and diabetes mortality determined by nutrition during parents' and grandparents' slow growth period. Eur J Hum Genet 10(11):682

Kommenda M, Kronberger G, Winkler S, Affenzeller M, Wagner S (2013) Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In: Blum C, Alba E, Bartz-Beielstein T, Loiacono D, Luna F, Mehnen J, Ochoa G, Preuss M, Tantar E, Vanneschi L (eds) GECCO '13 companion: proceeding of the fifteenth annual conference companion on genetic and evolutionary computation conference companion. ACM, Amsterdam, p 1121–1128. doi:10.1145/2464576.2482691

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

Krawiec K, Pawlak T (2013) Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. Genet Program Evolvable Mach 14(1):31–63. doi:10.1007/s10710-012-9172-7 . http://link.springer.com/article/10.1007/s10710-012-9172–7

La Cava W (2014a) Develep (Version 1.0.0) [software]. doi:10.5281/zenodo.9824. Retrieved from https://zenodo.org/record/9824

La Cava W, Spector L, Danai K, Lackner M (2014b) Evolving differential equations with developmental linear genetic programming and epigenetic hill climbing. GECCO '14: companion publication of the 2014 genetic and evolutionary computation conference. ACM. doi: http://dx.doi.org/10.1145/2598394.2598491

Mahfoud SW (1995) Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana-Champaign

Miller JF, Thomson P (2000) Cartesian genetic programming. Genetic programming. Springer Berlin Heidelberg, p 121–132. http://link.springer.com/chapter/10.1007/978-3-540-46239-2-9

Mingo J, Aler R (2007) Grammatical evolution guided by reinforcement. IEEE congress on evolutionary computation . CEC 2007, pp 1475–1482. 10.1109/CEC.2007.4424646

Nordin P, Francone F, Banzhaf W (1995) Explicitly defined introns and destructive crossover in genetic programming. In: Rosca JP (ed) Proceedings of the workshop on genetic programming: from theory to real-world applications. Tahoe City, p 6–22. http://web.cs.mun.ca/banzhaf/papers/ML95.pdf

ONeill 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://link.springer.com/article/10.1007/s10710-010-9113–2

Page J, Poli R, Langdon WB (1999) Smooth uniform crossover with smooth point mutation in genetic programming: a preliminary study. Genetic programming, proceedings of EuroGP'99, vol 1598 of LNCS, Springer-Verlag, pp 39–49

Pagie L, Hogeweg P (1997) Evolutionary consequences of coevolving targets. Evol Comput 5(4):401–418. http://www.mitpressjournals.org/doi/abs/10.1162/evco.1997.5.4.401

Pogribny I, Raiche J, Slovack M, Kovalchuk O (2004) Dose-dependence, sex- and tissue-specificity, and persistence of radiation-induced genomic DNA methylation changes. Biochem Biophys Res Commun 320(4):1253–1261. doi: 10.1016/j.bbrc.2004.06.081. http://www.sciencedirect.com/science/article/pii/S0006291X 040 13208

Ross BJ (1999) A lamarckian evolution strategy for genetic algorithms. Pract Handb Genet Algorithms Complex Codin Syst 3:1–16

Ryan C (1996) Reducing premature convergence in evolutionary algorithms. PhD thesis, National University of Ireland

Salman WP, Tisserand O, Toulout B, Stewart M (1985) Forth. Springer-Verlag, New York

Schmidt M, Lipson H (2007) Comparison of tree and graph encodings as function of problem complexity. Proceedings of the 9th annual conference on genetic and evolutionary computation, ACM, New York, GECCO '07, p 674–1679. doi:10.1145/1276958.1277288. http://doi.acm.org/10.1145/1276958.1277288

Schmidt M, Lipson H (2011) Age-fitness pareto optimization. Genetic programming theory and practice VIII, p 129–146. Springer New York. http://link.springer.com/chapter/10.1007/978-1-4419-7747-2–8

Spector L (2001) Autoconstructive evolution: Push, pushGP, and pushpop. Proceedings of the genetic and evolutionary computation conference GECCO-2001, p 137–146

Spector L, Helmuth T (2013) Uniform linear transformation with repair and alternation in genetic programming. Genetic programming theory and practice XI, page in preparation Springer. http://people.cs.umass.edu/thelmuth/Pubs/ultra-gptp-2013-preprint.pdf

Spector L, Robinson A (2002) Genetic programming and autoconstructive evolution with the push programming language. Genet Program Evolv Mach 3(1):7–40. http://link.springer.com/article/10.1023/A:1014538503543

Tanev I, Yuta K (2008) Epigenetic programming:genetic programming incorporating epigenetic learning through modification of histones. Inf Sci 178(23):4469–4481.doi: 10.1016/j.ins.2008.07.027. http://linkinghub.elsevier.com/retrieve/pii/S0020025508002880

Topchy A, Punch WF (2001) Faster genetic programming based on local gradient search of numeric leaf values. Proceedings of the genetic and evolutionary computation conference GECCO-2001, p 155–162. http://garage.cse.msu.edu/papers/GARAGe01-07-01.pdf

Turner BM (2000) Histone acetylation and an epigenetic code. Bioessays 22(9):836–845

Uy NQ, Hoai NX, ONeill M, McKay RI, Galvn-Lpez E (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet Program Evolv Mach 12(2):91–119. http://link.springer.com/article/10.1007/s10710-010-9121-2

White DR, McDermott J, Castelli M, Manzoni L, Goldman BW, Kronberger G, Jakowski W, OReilly UM, Luke S (2012) Better GP benchmarks: community survey results and proposals. Genet Program Evolv Mach 14(1):3–29. doi:10.1007/s10710-012-9177-2. http://link.springer.com/10.1007/s10710-012-9177–2

Whitley D, Gordon VS, Mathias K (1994) Lamarckian evolution, the baldwin effect and function optimization. Parallel problem solving from NaturePPSN III. Springer Berlin Heidelberg, p 5–15. http://link.springer.com/chapter/10.1007/3-540-58484-6-245

Wineberg M, Christensen S (2004) An introduction to statistics for EC experimental analysis. http://www.cis.uoguelph.ca/wineberg/publications/ECStat2004.pdf. Accessed 10 June 2014

William La Cava is a PhD student at University of Massachusetts, Amherst. He received his B.S. and M.Eng. from Cornell University in 2009 and 2010, and went on to work at the National Wind Technology Center in Boulder, Colorado. William is interested in automatic system identification and automatic control design for complex systems. As an NSF student fellow in the UMass IGERT Offshore Wind Energy Program, he is developing symbolic regression methods to create data-based models of offshore wind turbines and bird migration.

Lee Spector is a Professor of Computer Science at Hampshire College and an Adjunct Professor of Computer Science at the University of Massachusetts, Amherst. He received a B.A. in Philosophy from Oberlin College in 1984 and a Ph.D. in Computer Science from the University of Maryland in 1992. He is the Editor-in-Chief of the journal Genetic Programming and Evolvable Machines and a member of the editorial board of Evolutionary Computation. He is also a member of the SIGEVO executive committee and he was named a Fellow of the International Society for Genetic and Evolutionary Computation.

Chapter 4
SKGP: The Way of the Combinator

William P. Worzel and Duncan MacLean

4.1 Introduction

Genetic Programming (GP) has a rich history in both implementation and application. It is an evolutionary algorithm that evolves computer programs and is closely related to Evolutionary Programming and Evolutionary Strategies. GP generally refers to the thread of development that traces its lineage back to Koza (1992) and is characterized by three main aspects: a means of representing a population of functions or programs that will allow evolutionary operations to be applied with relative ease; an implementation of genetic operators, particularly crossover and mutation, that allows two or more individuals to be combined to produce offspring programs, which allows an individual program to change its structure or content, or both; and a way to resolve problems of functionality that arise from crossover and mutation. While there are other aspects of GP that are significant in their operation, these three factors: representation, implementation of genetic operators, and resolution of what may be called "program coherency" are the fundamental characteristics of GP.

There have been many variations in GP implementation starting with "Classic Koza GP" (Koza 1992) which uses program (parse) trees, Automatically Defined Functions (ADFs) (Koza 1994), grammatical evolution of GP (Ryan and O'Neill 1998), and stack based GP (Spector 2001). Spector et al. (2005) extended the Push architecture to include combinators, but has not used them extensively. Briggs and O'Neill (2006); Briggs and O'Neill (2008) described the use of combinators in GP and did some brief benchmark tests but seems to have left the pursuit of this approach behind after publishing two papers. We will describe the use of combinators in GP more thoroughly, explain an unusual benchmark of the system, and describe

W. P. Worzel (✉)
Evolution Enterprises, 214 W. Main St., 48160 Milan, MI, USA
e-mail: billw@evolver.biz

D. MacLean
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
e-mail: dmaclean@acm.org

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_4

some "real world" applications of our SKGP system before describing some future developments that may be of interest to the GP community.

4.2 Combinators

Combinators were first described in Schönfinkel (1967) and were more broadly developed by Haskell Curry (Curry 1929). They belong in the realm of functional programming and provide a way to extract variables from expressions which Turner (1979) demonstrated could be an effective way to implement functional programming languages. He observed that a result from combinatory logic could be used to remove variables from functional programs using combinators that were "compiled" into expressions that can be easily and efficiently evaluated.

The main operation is the "application" of a function to arguments (hence the alternative name for this style of programming, "applicative"). Function application is written in prefix notation via juxtaposition. For example, the function "+" which adds two numbers, would be written

$$+\ 1\ 2$$

which would evaluate to 3. We will write this informally as

$$+\ 1\ 2 \rightarrow 3$$

A technique called "Currying", named for Haskell Curry, but apparently first suggested by Schönfinkel (the term "Schönfinkelling" never caught on), means that we only have to consider functions of one argument.
Consider again the function "+" above.

$$+\ 1$$

can be considered to have as value the function "+1" which adds one to things. In turn, this function is applied to 2 and has a value 3. Hence application associates to the left, so the original expression would be parenthesized as

$$(+\ 1)\ 2.$$

A more complex example is the factorial function, which defined recursively in pseudocode, might be written

$$\text{let } fac\, n = \text{if } (n == 0) \text{ then } 1 \text{ else } n\ *fac(n\ -\ 1)$$

To rewrite this in applicative form, we need to introduce an applicative form of the conditional, "?"

$$?\, true\ x\ y \rightarrow x$$

$$?\, false\ x\ y \rightarrow y$$

Now we can rewrite the definition of factorial thus

$$fac\ n = (?\ (=0\ n)1\ (*\ n(fac\ (-\ n\ 1))))$$

How do we evaluate (fac 3)? We substitute 3 for n (or "bind" n to 3) in the expression on the right hand side. The recursive call to fac will create a new binding for n. This process is relatively simple for this case, but if a function returns a function as a result, then the system would have to create a closure to remember the bindings. This complexity resulted in a reputation for functional languages for inefficiency.

Turner observed that a result from combinatory logic could be used to convert functional programs written in the style we have described into a form from which all variables have been removed, and "compiled" programs that result can be evaluated in a particularly simple way.

The key combinators in this process are as follows:

$$K\ a\ b\ =>\ a$$

$$I\ a\ =>\ a$$

$$S\ a\ b\ c\ =>\ a\ c\ (b\ c)$$

Where S, K and I are combinators and a, b, and c are atoms or expressions. As with most functional programming, combinator expressions are always applied left-to-right so the result of applying the S combinator: $a\ c\ (b\ c)$ may be read as 'a' applied to 'c' and the result of that application is applied to the result of applying 'b'to 'c' (since '$(b\ c)$' must be evaluated first due to the parentheses surrounding them).

Armed with these three simple combinators, Turner showed that variables could be removed from expressions as follows.

We define an operation called abstraction to remove variables. The abstraction of x from expression E is written $[x]\ E$. Abstraction is inverse to application, that is

$$([x]E)x = E < x \text{ is abstracted from } E\ x \text{ where '}E\text{' is any expression}>$$

In the simplest abstraction algorithm, there are just three rules, depending on the structure of expression E

If E is atomic (a terminal) then either $E = x$ or $E = y \neq x$

$$[x]x \rightarrow I$$

$$[x]y \rightarrow Ky$$

If E is not atomic, but consists of an application ($E_1\ E_2$), then we introduce an S combinator and recurse.

$$[x]E_1E_2 \rightarrow S([x]E_1)([x]E_2)$$

It is simple to verify that abstraction is indeed the inverse of application, and that the process of abstraction does indeed remove all references to the bound variable from

the expression, at the cost of introducing a quantity of combinators. For functions of several arguments, recall the currying process and note that each argument can be removed in turn. This can result in an explosion in the number of combinators, so in practice, additional combinators are defined and more rules are added to those above to alleviate this problem. Two simple optimizations deal with the case where one or the other of the two sub expressions do not depend on x. By introducing new combinators B and C, with reduction rules

$$B\ f\ g\ x\ \rightarrow\ f\ (g\ x)$$

$$C\ f\ g\ x\ \rightarrow\ f\ x\ g$$

and abstraction rules

$[x]\ E_1\ E_2 \rightarrow K\ (E_1\ E_2)$ If neither E_1 nor E_2 contain x (note that the application of K selects $(E_1\ E_2)$ and abstracts out x)

$$[x]\ E_1\ E_2 \rightarrow B E_1([x]E_2)\ \text{If just } E_2 \text{ contains } x$$

$$[x]\ E_1\ E_2 \rightarrow C\ ([x]E_1)E_2\ \text{If just } E_1 \text{ contains } x$$

we can optimize the abstraction of variables.
Note however, that not even I is essential since

$$S\ K\ K\ x\ \rightarrow\ K\ x\ (K\ x)\ \rightarrow\ x$$

so

$I\ =\ S\ K\ K$

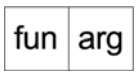
So only S and K are necessary ("form a basis"). In fact, a single element basis can be defined, but the combinator required is so complex that it is not practical for implementation.

As an example, with these rules, and a couple of other obvious optimizations, factorial becomes:

$$fac\ =S\ (C\ (B\ ?\ (=0))\ 1)\ (S\ *\ (B\ fac\ (C\ -\ 1)))$$

4.3 Implementation of SKGP

Diagrammatically, an application (fun arg) may be represented as:



Consider again the function "+" from above (Figs. 4.1, 4.2).

$$+\ 1$$

can be considered to have as value the function "+1" which adds one to things. In turn, this function is applied to 2 and has as value 3. Hence application associates to

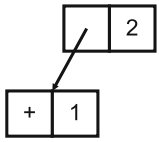


Fig. 4.1 Graph representation of + 1 2 in memory

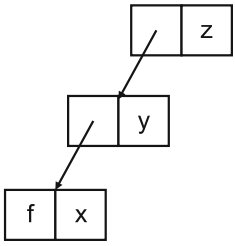


Fig. 4.2 Graph representation of $(f\ x\ y\ z)$ in memory

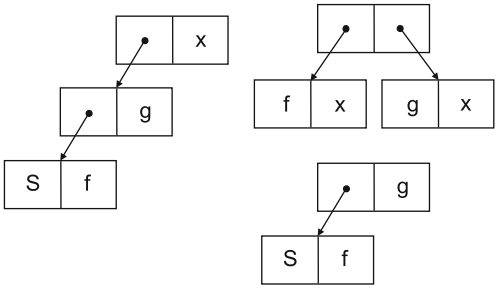


Fig. 4.3 Graph representation of application of *S* combinator

the left, so the original expression would be parenthesized as

$$(+\ 1)\ 2.$$

And in memory, it would look like Fig. 4.1
In general $(f\ x\ y\ z)$ is interpreted as $(((f\ x)\ y)\ z)$ and in memory it would look like Fig. 4.2.

Here is what *S* reduction looks like in graphical terms. The top application node on the right in Fig. 4.3 has had its fun and arg pointers overwritten with pointers to two newly created application nodes. Note that the two nodes below the new nodes are left dangling. They are not deleted, because there may be other pointers to them elsewhere. If this was the only pointer to them, then they will eventually be reclaimed by garbage collection.

Using this approach, the factorial program described above: $fac\ =\ S\ (C\ (B\ ?\ (=\ 0))\ 1)(S\ *\ (B\ fac\ (C\ -\ 1)))$ would be implemented as shown in Fig. 4.4.

However, this implementation still leaves a reference to fac itself in place as the recursion needed to express the factorial calculation in a functional way. We can even get rid of the "fac" by introducing another combinator Y, also called the fixed-point combinator or paradoxical combinator. It has the rule

$$Y\ f\ \rightarrow\ f\ (Y\ f)$$

although this is actually implemented as shown in Fig. 4.5.

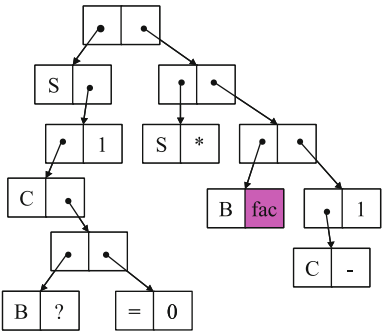


Fig. 4.4 Graph representation of factorial function

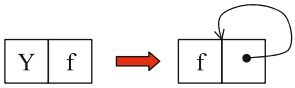


Fig. 4.5 Graph representation of Y combinator in memory

It is called the fixed-point combinator because $Y\ f$ is a solution of $x\ =\ f\ x$

Then we observe that if

$$fun\ =\ E$$

then $fun\ =\ ([fun]\ E)\ fun$ since abstraction is the inverse of application, so

$$fun\ =\ Y\ ([fun]\ E)$$

where the right hand side no longer contains any references to *fun*.

With this, programs move beyond trees to become graphs with cycles. Therefore, the process of evaluating programs by applying reduction rules is often called "graph reduction."

The lambda calculus can represent partial recursive functions, so the lambda calculus, and hence our combinator calculus, is "Turing complete." This, together with the very straightforward uniform implementation, makes it attractive as a GP representation.

It is this model of functional programming that we have chosen to implement in the SKGP genetic programming system. As in conventional GP, there are two kinds of object, terminals and non-terminals. However, there is only one non-terminal, and that is an application. The terminal consist of built in functions, the combinators described (and others) and potentially built in functions as well. There are no variables, which might be substituted by the actual value of arguments in a conventional GP system. Instead, the program is applied to the arguments and graph reduction performed until a result is computed.

There may be several reductions that can be performed at any one time. The Church-Rosser Theorem (Church and Rosser 1936) tells us that it does not matter in what order these are performed. A reduction order which has a particularly attractive property is called "normal order reduction." In this case the left most reduction that can be performed at any point is the one performed. It can be proven that if any reduction order terminates, then this one will. Also, normal order reduction has a

simple implementation. We simply trace down the function side of the program tree from the root pushing pointers onto a stack until we find a reduction we can perform and do that, using the stack for convenient access to the arguments.

There is a slight complication to this due to the fact that we have added some "strict" functions whose arguments must be evaluated before the function itself can be evaluated. These include the arithmetic functions such as "+". When we encounter a "+" to evaluate, we must first evaluate its arguments. To do this we evaluate each argument in turn as a sub-reduction, creating a new stack frame and repeating the process we use for the overall program.

4.4 Type System

There is a problem with the system if it is to be used in genetic programming. If you generate a random program tree, it is very likely to go wrong because an application tries to apply a function to the wrong kind of argument. The solution to this is to assign types to terminals and applications and only generate programs where the function and argument are "compatible" in the sense that the program will not go wrong when the application is evaluated. This kind of idea was introduced to GP in Montana (1975). However the approach used there was rather ad-hoc. Clack and Yu (1997) introduced a formal type system based on a Hindley/Milner polymorphic type system as used in functional programming systems such as ML or Haskell. Milner (1978) is a classic reference. Hindley (1997) is a thorough introduction, although developing a type system for the lambda calculus rather than combinatory logic. Practical treatments are given in Jones (1987) Chaps. 8 and 9 and Aho et al (1986) Chap. 6.

The development of SKGP is based on the work of Clack and Yu (1997), but using combinators rather than lambda abstractions, as they introduces in later papers, e.g. Yu and Clack (1998).

Interestingly, programs which can be assigned types under the basic type theory have the *Strong Normalization Property* i.e. they must always terminate with the same result. This seems to solve not only the problem of failure due to type incompatibility, but also failure due to non-termination. However, this means that this class of programs cannot be Turing complete, since otherwise we would have a solution (the type-checking algorithm) for the halting problem. This raises the question of whether this class of programs is a powerful enough subset to be useful and will be discussed a little further at the end of this section.

An informal discussion of the type system and type-checking algorithm will be presented here though the reader is encouraged to review the above references for more details.

The type system is *static*, rather than *dynamic*. This means that type checking is performed on the static program source, rather than at runtime, as in a dynamically typed language such as Smalltalk. This means that we do not need to carry type information at run time, because the possibility of type errors has been eliminated by the static analysis.

Note also that we do not implement any form of coercion or implicit type conversions such as allowing an integer to be used as an argument to a function that requires a float. Specific conversion functions would have to be provided to allow this.

We will write $E : \tau$ to denote that E has type τ.

We have some basic types, which include integers, double precision floating point, booleans e.g.,

1 : Integer
1.5 : Float
true : Boolean

A function taking an argument of type σ and returning a result of type τ has type $\sigma \rightarrow \tau$, e.g.

$$sin \ : \ \text{Float} \rightarrow \text{Float}$$

Recall that currying means that all functions are of this form (i.e., their application maps from a type to a result type), but the return type may itself be a function. Take for example the function + which returns the sum of two floating point numbers

$$+ \ : \ \text{Float} \rightarrow (\text{Float} \rightarrow \text{Float})$$

That is + takes a floating point number (x) and returns a function from floating point numbers to floating point numbers, namely the function that adds x. We omit the parentheses in this case, writing

$$+ \ : \ \text{Float} \rightarrow \text{Float} \rightarrow \text{Float}$$

What about the combinators? Consider I, which has the reduction rule

$$I \ x \ \rightarrow \ x$$

So I can take an argument of any type, and returns a result of the same type. To denote this, we use a type variable

$$I \ : \ \sigma \rightarrow \sigma$$

This is called a polymorphic type. σ can be replaced by any type, e.g. Integer, resulting in Integer → Integer.

What about the other combinators? $K \ x \ y \ \rightarrow \ x$. Assume $x \ : \ \sigma$ and $y \ : \ \tau$, so

$$(K \ x) \ : \ \tau \rightarrow \ \sigma$$

and hence

$$K \ : \ \sigma \rightarrow \ \tau \rightarrow \ \sigma$$

Remember that when fully parenthesized this would be written $K : \sigma \rightarrow (\tau \rightarrow \sigma)$. Now Let's tackle

$$S \ f \ g \ x \ \rightarrow \ f \ x \ (g \ x)$$

x is never applied to anything, so it can have the completely general type ρ.

$$x \ : \ \rho$$

We see g being applied to x. Let us say that it returns a value of type σ, so

$$g \ : \ \rho \rightarrow \ \sigma$$

Let us say that when f is applied to two arguments it returns a value of type τ. So we can deduce from the shape of $f\ x\ (g\ x)$ that when f is applied to an argument x of type ρ, it return a function which takes an argument of the type of $(g\ x)$, which is σ, and returns a value of the type τ so

$$f \ : \ \rho \rightarrow \ \sigma \rightarrow \ \tau$$

So now we know the types of all the arguments of S, and the result, so we have

$$S \ : \ (\rho \rightarrow \ \sigma \rightarrow \ \tau) \rightarrow (\rho \rightarrow \ \sigma) \rightarrow \rho \rightarrow \tau$$

The informal process used to find the type for the S combinator actually found the "Principal Type." That is, S may be applied to arguments f, g and x of any type for the type variables $\rho\ \sigma$ and τ and not have a type error. The Principal Type is the most general type that can be assigned to a term, in this sense.

Two types are "compatible" if there is a substitution for the type variables in the two types that can make them equal. The unification algorithm, first developed in Robinson (1965) for automated theorem proving, is used to find if such a substitution is possible. The unification algorithm lies at the heart of our type-checking algorithm.

A more accessible version of Robinson's original reference is Robinson (1965), which Robinson calls "the mechanization of deductive reasoning".

The algorithm takes two types, and returns either false, if they cannot be unified, or true, and a list of substitutions to unify them if they can be unified.

We perform type checking at three crucial stages

1. When creating programs in the first place. If we create only type correct programs, we should not get any run time errors. We create programs recursively from the root. We start with the target type of program we are trying to generate. We then choose at random to create either an application (non-terminal) or insert a terminal (subject to other limits on the size or depth of the program tree). If we choose a terminal, then we pick one at random (subject to the specified weights) and then try to unify its type (instantiated with new unique type variables) with the required type. If the types unify, then this terminal is type valid, the substitutions required for unification are added to the ongoing substitutions for the whole program and program construction continues. If the types do not unify, then another terminal is selected and tested in the same way. If no terminal has a type that unifies with the required type, then we can try generating an application instead, or failing. This can cause the generation of the whole program to fail, or backtracking to be applied, depending on runtime flags. If we choose to insert application, we will make two recursive calls to the creation function, one for the function part of the application, and one for the argument. If the type required for the application is Θ, say, then we instantiate a new type variable σ and require the function part of the application to have type $\sigma \rightarrow \Theta$ and the argument part to have type σ, so that when the function is applied to the argument, they are type compatible, and return a result of type Θ, as required.

2. When performing crossover, we only cross over type compatible program fragments. This ensures that if the parents are type correct, then so are the offspring.
3. When performing mutation. Again, when we replace a part of the program tree with a newly created branch, we make sure that the new branch is type correct and compatible with the node it is replacing.

The type system can be extended to cope with structure types such as lists and trees. In the initial implementation, we provided a list type, providing for homogeneous lists (all elements have the same type), written $[\sigma]$ for the type of a list of elements of type σ. An empty list is denoted by Nil, which has principal type $[\sigma]$. Functions hd and tl provide access to the head (first element) of the list and the tail (the rest of the list) and a pair function P to add elements to the head of a list

$$\text{Nil} : [\sigma]$$
$$\text{hd} : [\sigma] \rightarrow \sigma$$
$$\text{tl} : [\sigma] \rightarrow [\sigma]$$
$$\text{P} : \sigma \rightarrow [\sigma] \rightarrow [\sigma]$$

There is a problem with these functions, though. If you attempt to take the hd or tl of Nil, an error results. hd and tl are partial functions. Yu (2000) ran into this problem and devised a special fitness function to penalize programs that had errors of this kind. We prefer to use the type system to avoid errors of this type. For this reason we have introduced two new type operators, union and product, and also a method for creating user-defined types which will enable new types such as lists or binary trees to be added to the SKGP system without modifying the core.

The product type is the type of a pair (or "cons cell" for those coming from a LISP background). If the type of the left element (or "car") is σ and the type of the left element (or "cdr") is τ, then the type of the pair is $\sigma\ x\ \tau$, so the function types are now

$$\text{car} : \sigma\ x\ \tau \rightarrow \sigma$$
$$\text{cdr} : \sigma\ x\ \tau \rightarrow \tau$$
$$P : \sigma \rightarrow \tau \rightarrow \sigma\ x\ \tau$$

The union type means that an object can be one of two types. If the two member types are σ and τ, then the union type is written $\sigma\ +\ \tau$. A project function is provided which applies the right sort of function depending on which member of the union it is, and two injection functions give methods for creating union type objects from the member type objects

$$\text{inl} : \sigma \rightarrow \sigma\ +\ \tau$$
$$\text{inr} : \tau \rightarrow \sigma\ +\ \tau$$
$$\text{project} : (\sigma \rightarrow \rho) \rightarrow (\tau \rightarrow \rho) \rightarrow \sigma\ +\ \tau \rightarrow \rho$$

The next new feature is a special Nil type, for denoting empty lists. There is only one object of the nil type, Nil. At the risk of some notational ambiguity

$$\text{Nil} : \text{Nil}$$

Finally, in order to get back to a list type again, we introduce user-defined types. These create new parameterized types by recursion. Currently, we only support user-defined types with one parameter, but this could be expanded. We define the type List(σ) by the following equation

$$\text{List}(\sigma) = \text{Nil} + (\sigma \ x \ \text{List}(\sigma))$$

That is, a list of elements of type σ is either (union) Nil or a pair (product) consisting of an element of type σ (the head) and a list of elements of type σ (the tail).

We have extended the implementation of the unification algorithm to allow the expansion of a user-defined type by replacing any occurrence of the user-defined type by the right hand side of its definition in order to unify.

Now the project function is the only way to pick apart the union in the definition of a list, so this means that we must always handle the Nil case in order to have a type correct program. No more taking the hd or tl of an empty list!

That still leaves us with the problem of infinite recursion. So long as we don't introduce a recursion combinator like Y, then the strong normalization theorem tells us that any program that can be typed must terminate. That would seem to be an ideal solution. However, we have not managed to evolve programs that would normally be defined using recursion without introducing some sort of explicit recursion mechanism. We are not yet sure whether this is because the class of typeable programs is too small to include programs that can solve these problems, or because our existing combinator set is not well geared to evolving such programs.

There has been much work on proving termination for functional programs. An example is Elementary Strong Functional Programming, (Telford 2000). However this is a method for proving that a particular program will terminate, rather than constraining any program to terminate, such as we require.

The approach taken by Yu in Yu and Clack (1998) was to provide terminals such as map and foldr that have the desired termination properties and evolve functions using these as building blocks without providing any other recursion mechanisms. Unfortunately we have not yet seen immediately how to use this approach to user-defined types, for which we would have to hand-code the map functions. In languages with pattern-matching primitives, such as Charity (see home page at http://pll.cpsc.ucalgary.ca/charity1/www/home.html), you effectively get fold and map functions for free. Whether this can be achieved in a pure combinator language remains to be seen.

Note that Y cannot be assigned a valid type, since otherwise there would be a contradiction to the strong normalization theorem.

Another approach is to introduce a recursion combinator R. (This is similar to the approach taken in Yu (2000)). The R combinator is assigned the same type as the program being evolved, and represents a recursive call to that function. It is substituted with a pointer to the root of the function before evaluation. This is equivalent to abstracting the function name out, then adding the Y combinator and applying the Y combinator, with the restriction that the Y can only occur at the root of the program tree.

Using this method we have been able to evolve recursive functions successfully. This method is still not proof against non-termination due to infinite recursion (just

making a recursive call passing the original argument.) One possibility is to consider the recursive case of the user-defined type as a slightly different subtype i.e.

$$\text{List}(\sigma) = \text{Nil} + (\sigma \ x \ \text{List}_{sub}(\sigma))$$

where the matching of the type and its subtype is restricted, then wherever the type of the program references List(σ) then the R combinator would reference $\text{List}_{sub}(\sigma)$, thus forcing the recursive call to refer to just a part of the original argument. This takes the place of the pattern matching which might be provided in a more "user-friendly" language such as Charity.

4.5 Real World SKGP

The Briggs and O'Neill (2006) version of GP uses a Hindley-Milner strong type system with GP. They work through a number of common GP benchmarks including Linear Regression, Even Parity, and Stack and Queue evolution and show that combinator GP works well when compared to more standard representation forms. However such benchmarks, while demonstrating that and SKGP type approach is viable, does not in and of itself demonstrate usefulness in a real world setting.

Over the last 20 years, we have applied the SKGP to a number of different problems and in all cases where GP could be reasonably expected to return a result, we have reached satisfactory results. A brief summary of a few of these results are given along with some description of adaptations of the SKGP to solve specific problems.

4.5.1 Modeling Chemical Kinetics

In 1998, we used the SKGP to create a functional model of jet fuel combustion for NASA under a Small Business Innovation Research grant (SBIR). NASA had an existing first principles model of this combustion but it was very slow as it involved a set of complex PDEs to model the dynamics of the combustion of jet fuel. It was literally faster to test a specific jet fuel than it was to carry out the simulation. We were asked to "model-the-model" in order to produce a functional rendering of the model that was accurate within a range of inputs but was faster to compute. Such models are sometimes called surrogate models or emulators and have been used in many areas of study.

As a proof of concept, we were tasked with modeling three separate characteristics (temperature, pressure and velocity) at various points of a jet given a range of inputs and the resulting outputs of the first principles models. The resulting surrogate model was accurate to 0.1 % across the range of inputs it was trained on when tested on unseen combinations of inputs and had a speed of execution that was on average more than 2500 times faster than the first principles model. Using the SKGP we produced an empirical description of the reaction surface of the PDEs used in the model.

4.5.2 *Modeling Molecular Biology*

From 2001 through 2012 we applied the SKGP to various problems in molecular biology, particularly in molecular diagnostics. There were a series of publications relating to this work including Mitra et al. (2006); Lenehan et al. (2009); Almal et al. (2006); Yzerman et al. (2010) and Yu et al. (2007).

Most of these were classifier problems where we evolved functions from a training set of genomic data using the Area Under the Curve (AUC) for the fitness function, and assigned a slicepoint based on metrics dependent on medical preferences. For example, in predicting the recurrence of a disease (such as with Yzerman et al. (2010) it may be more desirable to have more false positive cases than false negative due to the consequences of missing a significant number of disease cases.

Mitra et al. (2006) is a particularly interesting case. We used GP to answer one question and found that we were asking the wrong question. In this case, Dr. Richard Cote asked whether we could find a molecular signature that corresponded to T-staging of bladder cancer tumors. Using the SKGP we analyzed the data but found that the results were, at best, inconclusive. However, in analyzing the samples that were misclassified, we noted that they had one unifying characteristic: they were all samples taken from tumors that had metastasized to local lymph nodes.

This suggested that there was a change in tumors after metastasis took place. Reanalyzing the samples in this manner found a clear signature associated with metastasis, though whether it was causative or simply after the fact of metastasis cannot be known without more experimentation, but the signature is at least suggestive as it seems to involve a molecular pathway known to be implicated in metastasis. This demonstrated the ability to derive a reliable identifier of metastasis using the SKGP as well as finding functions that are at least suggestive of the metastatic process. It also highlights the importance of "listening to data" in the sense that analyzing the results for more than success or failure is an important facet of GP.

4.6 Escaping the Bottle

In Daida et al. (2003), a fundamental limitation of tree-based GP was described based on structural aspects of tree-based GP. He discovered the existence of this problem and investigated the theoretical causes using a series of tunable problems he developed that were independent of anything other than the structure of the solution and showed that regardless of fitness landscapes, control parameters and other assumptions, these structural limitations severely limited GP from exploring more than a tiny fraction of the theoretical space. In essence, Daida described severe limitations on the space that could be searched due to the fractal nature of the growth of GP trees and its similarity to diffusion aggregation.

Figure 4.6 shows the constraint. This figure shows the number of nodes that can be found using one of the tunable problems. The fitness measure was how close to

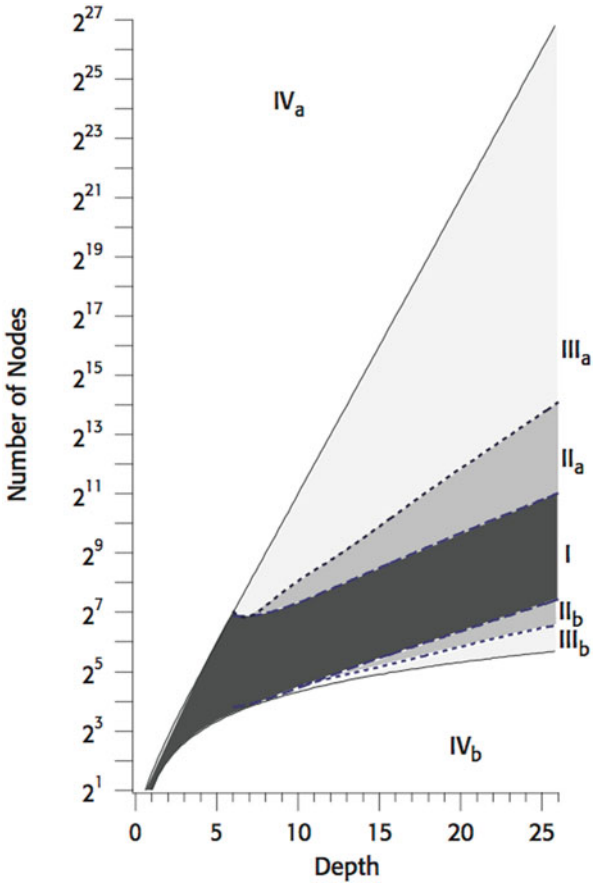


Fig. 4.6 GP constraints

a full tree can be created using GP where "full" means all possible nodes at a given depth are expressed in the tree.

In this figure first note that the plot is on a log scale so that the number of nodes increases by the power of two along the Y-axis. The two outermost lines show the theoretically accessible portion of the search space as the depth and number of nodes increases. Anything outside of those lines are impossible to represent and are labeled region IV. The innermost shaded area labeled I (which Daida called a "bottle") is easily accessible to standard GP. Most reachable solutions are contained within this region. The area labeled II is seldom reached and the region labelled III is almost never reached with standard tree-based GP. Note that region III is 15 orders of magnitude bigger than regions I and II. This means that, based on structure alone, standard GP cannot effectively search for solutions that are outside the bottle of region I.

Daida went on to observe that structure altering techniques or architectures make it possible to expand the search space. Such techniques include Automatically Defined Functions (ADFs), duplication and deletion operators. He also noted that other

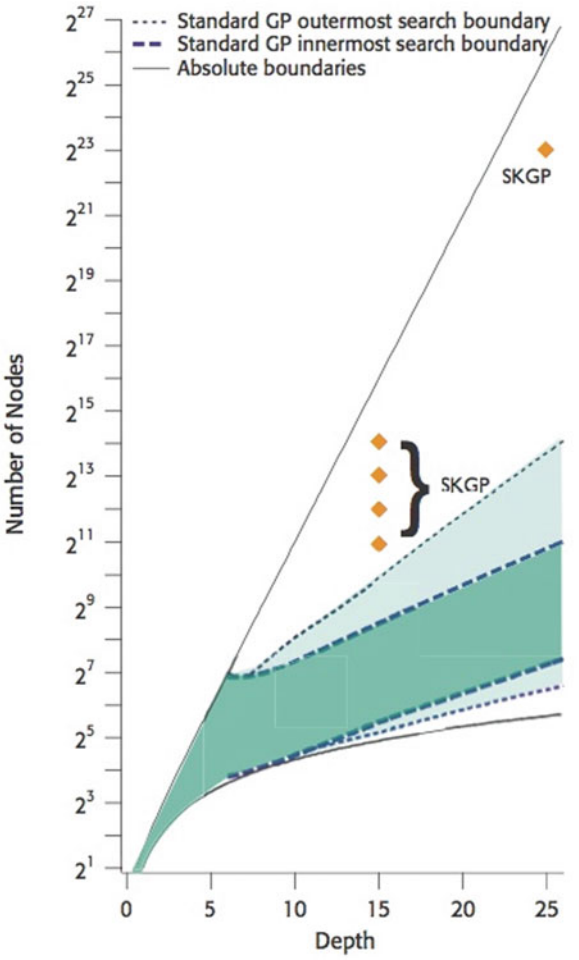


Fig. 4.7 SKGP results on LiD problem

representations such as the cyclic graph structures Koza used in Koza et al. (2004) to create analog circuits also had the property of escaping from this limitation through creating a meta structure to evolve designs.

With this in mind, given that SKGP uses structure altering operators in the form of graph reduction operators, we explored the SKGP's ability to "escape the bottle." As Fig. 4.7 shows, the SKGP was able to produce several solutions outside the bottle including those that were in region III, with one solution reaching approach 10 orders of magnitude greater number of nodes.

This result demonstrates that combinators are powerful structure altering operators and, that when they are used in GP, they have the potential to eliminate one of the structural limits inherent in standard GP. That said, in many cases, adequate solutions may reside within region I, in which case using non-strict combinators

expands the search space exponentially without any appreciable benefit. Daida et al. (2003) suggest this may be the case in their conclusion where they state that: *...this work does not necessarily support the premise that GP is deficient. Linkage between problem-solving ability and diversity has not been established.*

We have found that in many cases, limiting our analyses to 10 % of the possible solution space that Daida estimated as being accessible within the bottle will solve many useful problems. In this case, we simply remove structure altering operators such as S, K, B and C and instead focus on strict operators. However there have been cases where better solutions were found using structure altering operations as long as the depth of the allowable graphs are not excessive.

4.7 GP and SKGP in the World of Big Data

While the work described above, including successful "real world" applications, suggest that combinators and graphs are at least a satisfactory representation method for GP. The fact that combinators are structure altering operators may provide an escape from structural limitations inherent in tree-based techniques increases their value. However, there is more of value in using them in GP.

The first and most obvious value of combinator expressions is that because they are pure functional expressions, according to the Church-Rosser Theorem each expression may be evaluated independently of one another and a function composed of many expressions may be evaluated in parallel. Moreover, while this characteristic has been known for some time (see for example Turner (1979) and Jones (1987)), it was never as useful as it might have been because parallel evaluations may lead to wasted time due to unused expressions being evaluated. To take a trivial example, consider the expression:

$$(K(+\ 2\ 3)(*4\ 5))$$

Obviously the $(+2\ 3)$ and $(*4\ 5)$ expressions could be evaluated in parallel, apparently saving time. But recall that $K\ x\ y \rightarrow x$ and you see that the evaluation of $(*4\ 5)$ is totally useless. Variations on this conundrum reduce the value of the Church-Rosser Theorem as applied to the evaluation of computer expressions.

However, GP is not a case of a single run on a single set of data. Instead, we make multiple runs on training data over generations. This means that a component expression that is reused in future generations may have the graph-reduced value stored in order to replace the recurring sub-expression wherever it is used. By caching the reduced subexpression for each training case, it should be possible gain even more as it will reduce the time needed for each case for multiple training cases. Such a caching scheme would also apply to entire expressions since, depending on the selection method, some expressions will remain unchanged for generations.

Moreover Blickle and Thiele (1996); McPhee and Hopper (1999); Daida et al. (2003) all provide evidence that diversity of building blocks within a population decreases quickly (particularly under tournament selection) leaving a relatively small

subset of sub-expressions that are combined in different ways. This suggests that beyond the first few generations, the reuse of expressions should increase the speed of evaluation of individuals on a multi-processor system from the linear speedup inherent in parallel process to a super-linear speedup as the diversity of sub-expressions goes down and the available cache of sub-expressions applied to training samples goes up.

In the world of Big Data, where high-speed machine learning algorithms are of critical importance, the SKGP offers an inherently fast, parallel structure. Moreover, the fact that GP embodies a population of solutions means that as real-time data analysis needs change (eg, because of changes in shopping trends), a continually evolving solution is possible with subpopulations evolving using sliding time windows of data. In other words, it is not necessary to restart an analysis from scratch but rather we can continue an "endless" evolution of solutions.

In such an environment, niche solutions can be as important as mainstream solutions as different users may have different tastes. There is anecdotal evidence that big Internet companies are as interested in niche results as they are of the larger trends that are easy to find. Real-time analysis of GP populations can identify such sub-populations which can be presented for minority solution identification.

All told, GP is a useful analytic tool for Big Data and the SKGP is a particularly flexible and suitable to the existing parallel infrastructure used in this area.

References

Aho V, Sethi R, Ullman J (1986) Compilers principles, techniques, and tools. Addison-Wesley, Reading

Almal AA, Mitra AP, Datar RH, Lenehan PF, Fry DW, Cote RJ, Worzel WP (2006) Using genetic programming to classify node positive patients in bladder cancer. 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, vol 1. ACM Press, Seattle, pp 239–246. doi:10.1145/1143997.1144040. http://www.cs.bham.ac.uk/ wbl/biblio/gecco2006/docs/p239.pdf

Blickle T, Thiele L (1996) A comparison of selection schemes used in evolutionary algorithms. Evolut Comput 4(4):361–394. doi:10.1162/evco.1996.4.4.361. http://www.handshake.de/user/blickle/publications/ECfinal.ps

Briggs F, O'Neill M (2006) Functional genetic programming with combinators. In: Pham TL, Le HK, Nguyen XH (eds) Proceedings of the Third Asian-Pacific workshop on genetic programming. Military Technical Academy, Hanoi, pp 110–127. http://sc.snu.ac.kr/courses/2006/fall/pg/aai/GP/forrest/fsb-meo-combs.pdf

Briggs F, O'Neill M (2008) Functional genetic programming and exhaustive program search with combinator expressions. Int J Knowl-Based Intell Eng Syst 12(1):47–68. http://iospress.metapress.com/content/u6l4j13p67w66370/

Church A, Rosser JB (1936) Some properties of conversion. Trans Am Math Soc p 39(3): 472–482

Clack C, Yu T (1997) Performance enhanced genetic programming. In: Angeline PJ, Reynolds RG, McDonnell JR, Eberhart R (eds) Proceedings of the Sixth Conference on Evolutionary

Programming, Springer-Verlag, Indianapolis, Lecture Notes in Computer Science, vol 1213, pp 87–100. doi:10.1007/BFb0014803

Curry H (1929) An analysis of logical substitution. Am J Math 51:363–384

Daida JM, Li H, Tang R, Hilss AM (2003) What makes a problem GP-hard? validating a hypothesis of structural causes. In: Cantú-Paz E, Foster JA, Deb K, Davis D, Roy R, O'Reilly UM, Beyer HG, Standish R, Kendall G, Wilson S, Harman M, Wegener J, Dasgupta D, Potter MA, Schultz AC, Dowsland K, Jonoska N, Miller J (eds) Genetic and Evolutionary Computation – GECCO-2003. Springer-Verlag, Chicago, LNCS, vol 2724, pp 1665–1677. doi:10.1007/3-540-45110-2-60

Hindley J (1997) Basic simple type theory. Cambridge University Press, Cambridge

Jones SP (1987) The implementation of functional programming languages. Prentice-Hall International, UK

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge. http://mitpress.mit.edu/books/genetic-programming

Koza JR (1994) Genetic programming II: automatic discovery of reusable programs. MIT Press, Cambridge

Koza JR, Jones LW, Keane MA, Streeter MJ (2004) Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) Genetic programming theory and practice II, Springer, Ann Arbor, pp 121–142. doi:10.1007/0-387-23254-0-8. http://www.genetic-programming.com/gptp2004.pdf, pages missing?

Lenehan F, Fry D, Heyman E, Walters R, Worzel W (2009) Generation and validation of a primary tumor derived 4-gene prognostic signature for recurrence of stages i/ii colorectal cancer following potentially curative resection. J Clin Oncol (Meeting Abstracts), 27:15s

McPhee N, Hopper N (1999) Analysis of genetic diversity through population history. In: W B et al. (ed) Proceedings genetic evolutionary computation conferences. pp 1112–1120

Milner R (1978) A theory of type polymorphism in programming. J Comput Syst Sci 17(3):348–375

Mitra AP, Almal AA, George B, Fry DW, Lenehan PF, Pagliarulo V, Cote RJ, Datar RH, Worzel WP (2006) The use of genetic programming in the analysis of quantitative gene expression profiles for identification of nodal status in bladder cancer. BMC Cancer 6:159. http://www.biomedcentral.com/1471-2407/6/159

Montana DJ (1975) Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc.

Robinson JA (1965) A machine-oriented logic based on the resolution principle. J ACM 12 (1): 23–41

Ryan C, O'Neill M (1998) Grammatical evolution: A steady state approach. In: Koza JR (ed) Late breaking papers at the genetic programming 1998 conference, stanford University Bookstore, University of Wisconsin, Madison, Wisconsin, USA, pp 180–185. http://citeseer.ist.psu.edu/cache/papers/cs/12751/http:zSzzS zscare.csis.ul. iezSzmichaelzSzpaperszSzgp98.pdf/ryan98grammatical.pdf

Schönfinkel M (1967) From Frege to Gödel: a source book in mathematical logic, Harv, chap Über die Bausteine der mathematischen Logik, pp 1879–1931

Spector L (2001) Autoconstructive evolution: Push, pushGP, and pushpop. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt HM, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 137–146. http://hampshire.edu/lspector/pubs/ace.pdf

Spector L, Klein J, Keijzer M (2005) The push3 execution stack and the evolution of control. In: Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llora X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E (eds) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, vol 2. ACM Press, Washington DC, pp 1689–1696. doi:10.1145/1068009.1068292. http://www.cs.bham.ac.uk/ wbl/biblio/gecco2005/docs/p1689.pdf

Turner D (1979) A new implementation technique for applicative languages. Software Pract Exper 9: 31–49

Yu T (2000) Polymorphism and genetic programming. In: Whitley D (ed) Late breaking papers at the 2000 genetic and evolutionary computation conference. Las Vegas, pp 437–444

Yu T, Clack C (1998) Recursion, lambda abstractions and genetic programming. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic programming 1998: proceedings of the third annual conference. Morgan Kaufmann, University of Wisconsin, Madison, pp 422–431

Yu J, Yu J, Almal AA, Dhanasekaran SM, Ghosh D, Worzel WP, Chinnaiyan AM (2007) Feature selection and molecular classification of cancer using genetic programming. Neoplasia 9(4):292–303. doi:10.1593/neo.07121

Yzerman E, den Boer J, Caspers M, Almal A, Worzel B, van der Meer W, Montijn R, Schuren F (2010) Comparative genome analysis of a large dutch legionella pneumophila strain collection identifies five markers highly correlated with clinical strains. BMC Genomics 11:433. doi:10.1186/1471-2164-11-433. http://www.biomedcentral.com/1471-2164/11/433

William P. Worzel is one of the original organizers of the first GP Theory and Practice workshop along with Rick Riolo. He is an entrepreneur and a consultant, whose fundamental interests are in understanding the evolutionary mechanisms of GP (and nature) in order to create better GP systems and apply them to new problems

Duncan MacLean has worked with Evolution Enterprises for 20 years in developing and extending the SKGP. He has a Master's degree in mathematics from Cambridge University in England and is highly skilled in functional programming.

Chapter 5
Sequential Symbolic Regression with Genetic Programming

Luiz Otávio V.B. Oliveira, Fernando E.B. Otero, Gisele L. Pappa and Julio Albinati

5.1 Introduction

Many researchers have been interested in exploring the regularities and modularities of the search space in order to improve the performance of Genetic Programming (GP) when dealing with complex problems (Koza 1992a, 1994). A popular approach is to allow GP to define modules, by either evolving specific code to be used as a module or identifying potentially useful code in existing individuals, in the hope that a module will capture regularities in the search space and ultimately decompose the original problem into small (more tractable) subproblems. While previous approaches have shown some degree of success, they rely on the idea that useful modules will emerge during the GP search and they are very much focused on the structure (syntax) of the individuals. There are potential drawbacks associated with these assumptions: there is no guarantee that modules are solving different parts of the problem, the quality of modules is determined indirectly by evaluating the individuals that use the modules and there is still a pressure on the GP to find the complete solution to the problem at once—i.e., both modules and the code that uses the modules are evolved at the same time.

Traditionally, GP search operators perform modifications to individuals' representation (syntax), with the aim that these modification will lead to changes in their

L. O. V.B. Oliveira (✉) · G. L. Pappa · J. Albinati
DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
e-mail: luizvbo@dcc.ufmg.br

F. E.B. Otero
School of Computing, University of Kent, Chatham Maritime, UK
e-mail: F.E.B.Otero@kent.ac.uk

G. L. Pappa
e-mail: glpappa@dcc.ufmg.br

J. Albinati
e-mail: jalbinati@dcc.ufmg.br

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_5

behaviour (semantics). In other words, traditional GP search operators are *blind* operators regarding the semantics of an individual. In the same sense, syntactical approaches for modularisation are also *blind* regarding the definition of modules, since there is no guarantee that their behaviours are different—i.e., that they are solving different parts of the problem. Moraglio et al. (2012) recently proposed geometric semantic search operators in the context of the Geometric Semantic Genetic Programming (SGP), which can be used to directly search the semantic space of the problem. An interesting characteristic of SGP is that the fitness landscape seen by the search operators is unimodal for problems consisting in finding the correct mapping for input-output pairs—the fitness is the distance of the output vector of a solution to the optimum. Therefore, these operators present a new opportunity to explore the modularity of the GP search.

The problem-solving procedure employed by GP algorithms can be seen as a supervised learning procedure: given $\{(c_1, o(c_1)), \ldots, (c_n, o(c_n))\}$ input-output pairs representing the training cases C, where each pair $(c_i, o(c_i))$ denotes an input value and its correspondent output value, respectively; the problem can be defined as finding a function $f : C \rightarrow O$ that maps each case c_i in C to its correspondent output $o(c_i)$ in O. Many supervised learning algorithms employs a strategy to decompose the original into subproblems, find solutions to these subproblems and use them to generate the solution for the original problem. For example, top-down decision tree induction employ a divide-and-conquer strategy, where at each decision (internal) node the training cases are divided based on a test outcome. Each subset of the training cases, representing a reduced problem, is pushed down the tree and the procedure is repeated until a leaf node is generated. A similar strategy is used by many rule induction algorithms, where a sequential covering strategy is used to transform the problem of finding a list of classification rules into a sequence of smaller problems of finding a good rule. After a rule is created, the training cases classified by the rule are removed, reducing the number of training cases for the next iteration of the procedure.

Given that GP is essentially a supervised learning method and geometric semantic operators enable the direct manipulation of the output vectors, *could we apply a heuristic to decompose the problem into smaller subproblems and use GP to solve them?* Otero and Johnson (2013) presented a strategy based on the sequential covering to decompose a boolean problem into smaller subproblems. Each subproblem is then solved by a traditional GP and the individual solutions are combined using a geometric semantic crossover. It uses a property of the geometric semantic crossover for the boolean domain: individuals are combined using a boolean mask, which acts as a selector to inform when a particular individual solution should be used. While this strategy is successful for boolean domains, there is not a straightforward way to adapt it to the real domain, since the operation of the geometric semantic crossover is different.

In this chapter we present a method to sequentially solve symbolic regression problems using a combination of geometric semantic operators and a heuristic inspired by the traditional sequential covering strategy. The proposed method, Sequential

Symbolic Regression (SSR), works by sequentially transforming the original problem, according to the partial solutions generated, into potentially simpler ones. The rationale behind SSR is that, after generating a suboptimal function f via symbolic regression, the output errors can be approximated by another function, in a subsequent iteration. In order to transform the original output based on the output of function f, each iteration of SSR applies a transformation based on a geometric semantic crossover operator (Moraglio et al. 2012). This procedure allows the GP to focus, at each iteration, on different aspects (subproblems) of the original problem.

The remainder of the chapter is organised as follows. Section 5.2 reviews previous works exploring regularities and modularity in GP. Section 5.3 revises the properties of geometric semantic operators. The proposed strategy for sequential symbolic regression is presented in Sect. 5.4, followed by computational experiments in Sect. 5.5. Finally, Sect. 5.6 concludes the chapter and presents future research directions.

5.2 Modularisation in Genetic Programming

Since the introduction of genetic programming (Koza 1992a), researchers have been interested in exploring the regularities and modularity of problem spaces. One of the main motivations is to identify these regularities to decompose the problem at hand into more tractable sub-problems; finding solutions to sub-problems should be easier than finding a solution to the original problem, and these sub-solutions can be used to create the solution to the whole problem. This process is illustrated in Fig. 5.1. This is analogous to how human programmers usually tackle problems: instead of creating a single procedure to implement an entire program, they usually break down the implementation into several different procedures and the combination of these procedures compose the complete implementation.

The Automatically Defined Functions (ADFs) proposed by Koza (1992a, b, 1994) was one of the first ideas to address the automated problem decomposition. ADFs impose a syntactical structure to individuals: an individual genotype is divided into a *result-producing branch* and several *function-defining branches*. The motivation is that function definitions potentially exploit the regularities of the problem space and these definitions can be used from the result-producing branch. On the one hand, Koza argues that by allowing the definition and use of functions, the problem is decomposed into subproblems. On the other hand, the modular structure (syntax) of individuals is manually defined, therefore, the decomposition process is not autonomous—the number of ADFs and their parameters are controlled by user-defined values. Additionally, even if functions actually represent solutions to subproblems, they are being evolved at the same time as the complete solution. There is a pressure to solve all parts of the problem at once—the definition of the functions and the correct use of those functions.

A popular idea to explore problem space regularities focused on defining modules based on the genetic material of individuals. Several involved the random selection

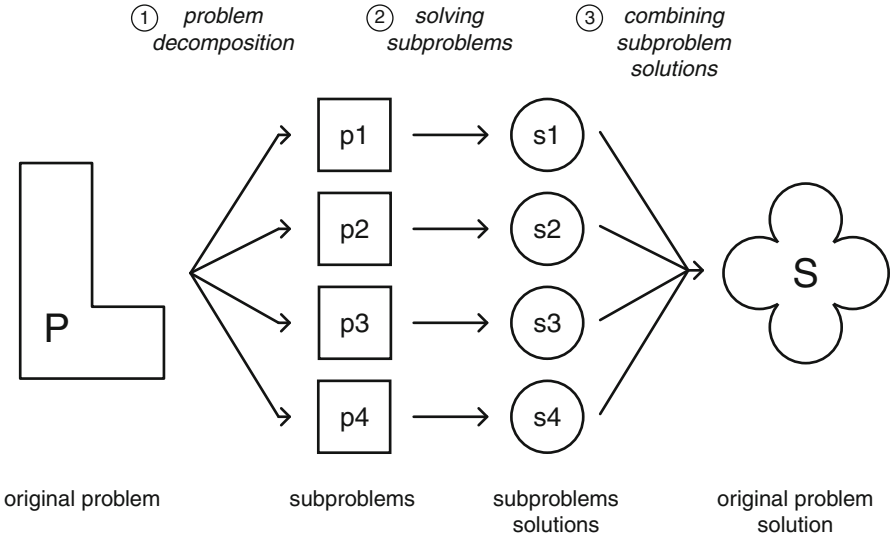


Fig. 5.1 The hierarchical problem-solving process: the original problem P is decomposed in a set of subproblems (step 1); the goal is then to solve each of the subproblems (step 2); finally, the solution S to the original problem P is created by using the solutions to the subproblems (step 3). Figure adapted from Koza (1992a)

of subtrees to create modules: Koza (1992a) proposed the use of a subtree *encapsulation* operator, which consists of randomly selecting a subtree from an individual to create a terminal primitive that encapsulated the subtree; Angeline and Pollack (1992, 1994) proposed the Genetic Library Builder (GLiB) system, which employs mutation operators that randomly select subtrees to create modules (the *compress* operator) that can be later expanded (the *expand* operator); similar *compress* and *expand* operators to create and expand modules were more recently proposed by Walker and Miller (2008) in the context of Embedded Cartesian Genetic Programming (ECGP), with the extension of the use of module-altering operators (*module point* mutation, *add-input*, *add-output*, *remove-input* and *remove-output* operators); Spector et al. (2011a, b, 2012) proposed the use of 'tags' to label fragments of code that can be later reused by referencing the same label—while this is similar to the use of a *compress* operator, it provides the flexibility of partial name matches (a label will match the closest matching tag).

Other authors followed the idea of identifying useful building blocks (subtrees) to define modules: Rosca and Ballard (1994) proposed the use of heuristics to create new modules, selecting fit blocks (blocks with high fitness value) and frequent blocks (blocks that appear frequently in the population). Similarly, Roberts et al. (2001) accumulate the frequency information of multiple runs of a GP to create a *subtree database* and subsequent runs can use the most frequent subtrees encapsulated as terminal primitives.

There are also works that explore the idea of a library of modules created prior to the run of a GP. Keijzer et al. (2004) introduced the use of Run Transferable Libraries (RTL). The RTL is created by running GP on lower-order problem instances, considered as a training phase, and then using it to solve more complex instances of the same problem. Similarly, Christensen and Oppacher (2007) generated small trees for the Santa Fe Trail problem to create a library of modules in a training phase, where the small trees are not necessarily complete solutions, and then using this library to find the complete solution to the problem. Another approach that uses the idea of training a GP on smaller problem instances in order to generate modules was presented by Jackson and Gibbons (2007), where the authors proposed the use of *layered learning*. The first layer is used to solve a lower-order version of the original problem and the final solution at this layer is converted to a parameterised module. The second layer uses this module to search the solution of a higher-order version of the same problem. While the creation of a library of modules in a training phase or in different layers can provide a decomposition of the problem, it represents a single decomposition step and it is not automated—the user has to manually choose to use either a training phase or to generate small trees prior to the search for the complete solution.

Considering the initial goal of problem decomposition, the aforementioned approaches rely on the assumption that the modules created could represent solutions to subproblems. The main drawback of this assumption is that modules are defined based on their syntax—i.e., the creation/selection of the modules does not involve any evidence that the modules are solving different parts of the problem.[1] A common characteristic of these approaches is that they provide a mechanism to create/identify modules during the run of the GP and expect that good modules will emerge as a result of the search, but at the same time, they do not employ any control over whether the use of modules decomposes the problem into subproblems. Perhaps the emphasis in syntactical approaches to modularity is a result of the tendency of using syntactical search operators in GP—both crossover and mutation operators are blind search operators regarding their effect on the individual behaviour, only focusing on syntactical changes. Additionally, the pressure of solving all parts of the problem at once might reduce diversity and, in some cases, also prevent the convergence to the optimal solution (McKay 2000).

Lee (1999) proposed an extension to GP to deal with forecasting of real world chaotic time series, which resembles the sequential strategy of the algorithm proposed in this chapter. Lee's assumption is that a time series is composed by deterministic and stochastic parts—subtracting the solution found by a run of the GP for the deterministic part from the original time series, the stochastic part is obtained as a residual time series. Applying this process recursively to the sequence of residual time series, a set of (sub-)solutions can be created. These are then combined using numerical coefficients calculated by the least square method with respect to a predetermined

[1] The selection of building blocks based on fitness proposed by Rosca and Ballard (1994) is an exception to the syntax-oriented selection, although there is no guarantee that different modules are solving different parts of the problem.

region of the time series—the explicit definition of regions of the time series (regions of the search space) can be seen as a *manual* decomposition of the problem. As we will discuss in the following sections, our proposed algorithm does not rely on the definition of regions of the search space and the (sub-)solutions evolved are combined using the principle of a geometric semantic crossover to produce the solution to the original problem.

5.3 Geometric Semantic Operators

Standard genetic programming operators were originally conceived to operate in the *syntatic-level* of the solutions being evolved. Consider, for example, a subtree crossover. It will randomly select subtrees from two previously generated solutions and swap them, regardless of what the outputs of the selected subtrees are. When tree outputs are neglected, we ignore the fact that, at the end of the evolutionary process, what matters is the quality of the best solution found, which is indirectly defined by the output generated.

The semantics of an individual can be informally defined as the meaning of syntactically correct programs or functions (Uy et al. 2011)—in a GP context, this is the set of outputs produced by a program or function given a set of inputs. Many approaches have been previously used to represent and extract semantics from genetic programming (Vanneschi et al. 2014). This section is interested in one of these approaches: geometric semantic operators.

In order to design operators that directly impact the semantics of a solution, Moraglio et al. (2012) defined the concept of *semantic distance* and *geometric semantic operators* for the real functions domain (e.g., symbolic regression), which are replicated in Definition 1 and Definitions 2 and 3, respectively.

Definition 1 Let S be the set of solutions and $s_1, s_2 \in S$. A function $SD : S \times S \rightarrow \mathbb{R}$ is said to be a semantic distance function if $SD(s_1, s_2) = D(O(s_1), O(s_2))$, where $O(s)$ returns the output vector of s and D is a distance function.

Definition 2 Let S be the set of solutions, $XO : S \times S \rightarrow S$ be a crossover operator and SD be a semantic distance function. XO is said to be geometric with relation to SD if, for all $s_1, s_2, s_3 \in S$ such that $s_3 = XO(s_1, s_2)$, $SD(s_1, s_2) = SD(s_1, s_3) + SD(s_3, s_2)$.

Definition 3 Let S be the set of solutions, $MT : S \rightarrow S$ be a mutation operator and SD be a semantic distance function. MT is said to be ε-geometric with relation to SD if, for all $s_1, s_2 \in S$ such that $s_2 = MT(s_1)$, $\mathbf{E}[SD(s_1, s_2)] \leq \varepsilon$, where $\mathbf{E}$ denotes the expected value.

Definitions 2 and 3 show that semantic geometric operators generate solutions in a much more controlled fashion. Particularly, the semantics of a solution generated through a geometric semantic crossover is guaranteed to be somewhere between the semantics of its parents. This fact implies in an interesting property: an offspring will never be worse than the worst of its parents. Similarly, an ε-geometric semantic

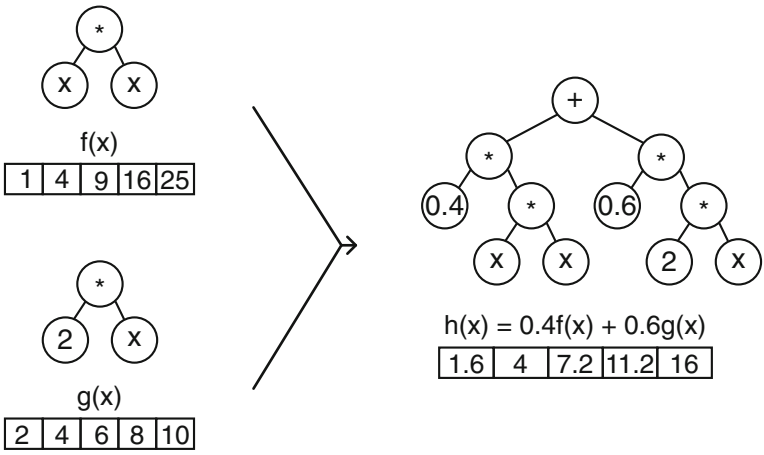


Fig. 5.2 Example of geometric semantic crossover operator between $f(x) = x^2$ and $g(x) = 2x$ using $r = 0.4$

mutation will generate solutions that are, on average, not worse than the original solution by more than ε.

Moraglio et al. (2012) also proposed specific semantic geometric operators for regression problems. The crossover operator proposed is essentially a convex combination of functions. Let S be the set of solutions, $s_1, s_2 \in S$, $XO(s_1, s_2) = r.s_1 + (1 - r).s_2$, where r is a random real number in the interval $[0, 1]$. The mutation operator was defined as $MT(s) = s + ms.(TR_1 - TR_2)$, where $s \in S$, ms is a real number and TR_1, TR_2 are randomly generated trees. The authors show that these operators are geometric with relation to the semantic distance function $SD(s_1, s_2) = \sum_{x_i \in T} [O(s_1)(x_i) - O(s_2)(x_i)]^2$, where T is a set of training examples.

Figures 5.2 and 5.3 show examples of geometric semantic operators for the real functions domain. Observe that in Fig. 5.2, each element of the output vector of the offspring is a convex combination of elements from the parents' output vectors using coefficients 0.4 and 0.6. In Fig. 5.3, we notice how the impact of the geometric semantic mutation operator can be controlled by setting appropriate values for ms.

5.4 Sequential Symbolic Regression

This section introduces Sequential Symbolic Regression (SSR), a method that sequentially executes a standard GP for symbolic regression and indirectly considers the semantic of the solutions being created. SSR is inspired by a sequential covering strategy, similar to the one employed by Otero and Johnson (2013), where at each iteration a solution to a transformed (and potentially simpler) problem is evolved.

The main difference between SSR and a traditional sequential covering method is in the transformation step that occurs at each iteration. In a traditional sequential

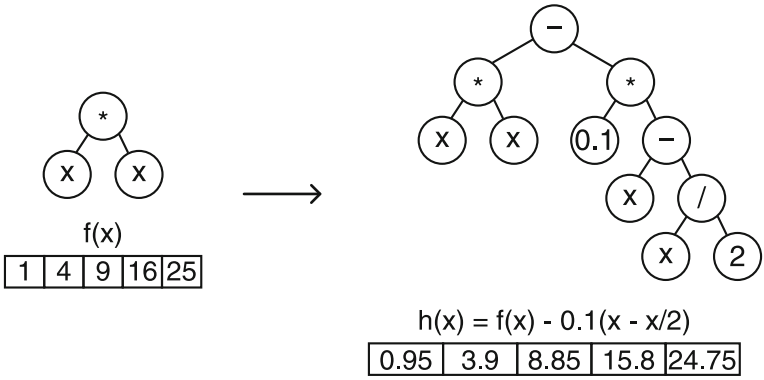


Fig. 5.3 Example of geometric semantic mutation operator of $f(x) = x^2$ using $ms = 0.1$, $TR_1(x) = x$ and $TR_2(x) = x/2$

covering strategy, the problem is reduced at each iteration—i.e., the training cases covered by the iteration solution are removed, effectively reducing the problem to the subsequent iterations. Since SSR deals with problems in the real-valued domain, the concept of *covered* training cases is not directly applicable.[2] Instead of removing training cases, at each iteration of SSR, the output values of the original problem are modified based on the use of a geometric semantic crossover and the iteration solution output—the transformation of the problem is based on the semantic of the solution created by the iteration. We hypothesise that the use of the iterative (sequential) solution construction procedure allows the GP to focus on different aspects (subproblems) of the original problem, creating individual solutions that are combined by a geometric semantic crossover.

A typical symbolic regression problem can be defined as follows. Given a set of input-output pairs $C = \{(c_1, o(c_1)), ..., (c_n, o(c_n))\}$ representing the training cases, where each pair $(c_i, o(c_i))$ denotes an input value and its correspondent output value, respectively; a symbolic regression problem can be defined as finding a function $f : C \rightarrow O$ that minimizes an error metric, such as the mean squared error (MSE), the mean absolute error (MAE) or the root mean squared error (RMSE).

The metrics described above use the summation of the squared or absolute residuals—the difference between the current output and the function output—to compute the error function. Hence, when the absolute value of residuals is minimized, so is the measured error. A residual $e(c_i)$ corresponds to the error in the fitting of the function to the i-th observation, and is defined as

$$e(c_i) = o(c_i) - \hat{o}(c_i) = o(c_i) - f(c_i). \tag{5.1}$$

[2] It is unlikely that a solution will reach (near) zero error only for a subset of the points (training cases), unless it is the optimal solution, which in this case it will reach a (near) zero error for all points.

The optimal solution to a regression problem is a function f^*, such that $e(c_i) = o(c_i) - f^*(c_i) = 0$ for $i = 1, 2, ..., n$, and often a function f found by a regression method is an approximation of f^*, not reaching a zero error or the minimum error defined according to the problem.

The rationale behind the sequential procedure of SSR is that, after generating a suboptimal function f, the residual can be approximated by another function in a subsequent iteration. In order to transform the original output based on the output of function f, each iteration of SSR applies a transformation based on a geometric semantic crossover operator (Moraglio et al. 2012). The geometric semantic crossover operator for the real-value domain combines the output of two known functions f and f' to generate a new function f^*, with an a priori unknown output. The principle used in SSR is that the output of function f and the output of the function f^* are known, and therefore, they can be used to define the transformation required to determine the desired output of function f' based on the residual of function f. The definition of the geometric semantic crossover is given by

$$f^*(c_i) = r \cdot f(c_i) + (1 - r) \cdot f'(c_i), \tag{5.2}$$

where r is a random real constant in the range $[0, 1)$. Substituting the definition of function f^* to the residual equation, we obtain

$$e(c_i) = o(c_i) - [r \cdot f(c_i) + (1 - r) \cdot f'(c_i)]. \tag{5.3}$$

Using Eq. (5.3) and given that f is the function created by an iteration of SSR, the output $o'(c_i)$ for function $f'(c_i)$ that reduces the residual error e to zero is computed as

$$o'(c_i) = \frac{o(c_i) - r \cdot f(c_i)}{1 - r}. \tag{5.4}$$

The transformed output vector o' defines a new regression problem, where the goal is to find a function f' that minimizes the new residuals $e'(c_i) = o'(c_i) - f'(c_i)$, which is the definition of problem for the next iteration.

Another way to see the strategy employed in SSR is to look at the use of the transformation step: a solution is built starting from the desired output, the output of the original problem; if the function (individual) f created at an iteration of SSR does not minimize the error e to zero, a geometric semantic crossover is used to transform the original problem. Given that we know the desired output—the output of the individual generated by the crossover operation—and one of the individuals of the crossover, we can determine the required output of a second individual that complements the crossover.

Therefore, instead of combining individuals at random as in the Semantic GP, SSR optimises the effect of the geometric semantic crossover operator by searching for the individual that represents the best match (minimises the error) given the desired output vector. At the same time, it indirectly mitigates the problem of exponential growth of individuals observed in SGP (Moraglio et al. 2012; Vanneschi et al. 2013), since the solution is created sequentially, without requiring that all individual solutions are kept in memory, and there is only one solution being created using the

geometric semantic operator, requiring a single simplification step at the end of SSR if the size of the complete solution needs to be reduced.

Algorithm 1 Sequential Symbolic Regression procedure

```
input: training points (C), stopping criteria, GP parameters
input ← (c_1, c_2, ..., c_n), for c_k ∈ C
output ← (o(c_1), o(c_2), ..., o(c_n)), for o(c_k) ∈ C
/* Solution iteratively constructed */
S ← {}
while stopping criteria not reached do
    f ← RunGP(input, output)
    if (MSE(f, output) ≤ 0.01) then
        S ← AddFunction(f)
    return S
    else
        r ←random()
        S ←AddFunction(f
        output ←AdjustOutputs(f,r,output)
    end if
end while
return S
```

5.4.1 SSR Procedure

Algorithm 1 presents the high-level pseudocode of the SSR procedure. It starts with an empty solution tree S, which is iteratively incremented by carrying out sequential regressions using a traditional GP algorithm. At the k-th iteration, a new function f_k is generated by the GP (*RunGP* procedure). If function f_k corresponds to the optimal solution—i.e., the output of f_k is such that $MSE(f_k, output) \leq 0.01$—$f_k$ is added to the solution tree S and the sequential procedure stops. Otherwise, f_k is added to the solution tree S using a geometric semantic crossover with a random constant r_k in the range [0, 1). Note that at this point the crossover operation is incomplete—i.e., only one of the parent individuals is known. Then, the constant r_k and the function f_k are used to modify the desired *output* using the transformation represented by Eq. (5.4). The iterative transformation step is given by

$$o_{k+1}(c_i) = \frac{o_k(c_i) - r_k \cdot f_k(c_i)}{1 - r_k}, \tag{5.5}$$

for $k = 1, 2, ..., n$, where n is the maximum number of iterations. The sequential SSR process continues until a minimum error or a maximum number of iterations is reached. Figure 5.4 illustrate the sequential solution construction, showing the solution tree S at different iterations of the procedure.

Next, we present an illustrative example of how SSR works. Let us consider that we want to find a function whose values match those in a set of training input cases $C = \{(1, 1), (3, 4), (5, 9)\}$, i.e., $input = (1, 3, 5)$ and $output_1 = (1, 4, 9)$. Let

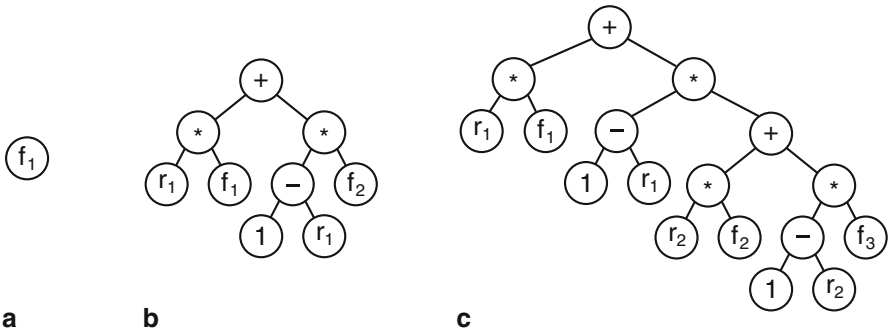


Fig. 5.4 Illustration of the solution tree S and the corresponding expression at different iterations: **a** $S = f_1$; **b** $S = r_1 \cdot f_1 + (1 - r_1) \cdot f_2$; **c** $S = r_1 \cdot f_1 + (1 - r_1) \cdot [r_2 \cdot f_2 + (1 - r_2) \cdot f_3]$

Table 5.1 Example of SSR execution. The First column presents the current iteration, followed by the values of r_k, the desired outputs $output_k$ (3 columns), the evolved outputs f_k (3 columns), the absolute residuals of f_k regarding $output_k$ (3 columns) and MSE (last column)

		$output_k$			$f_k(c_i)$			$\lvert e'_k(c_i)\rvert = \lvert o_k(c_i) - f_k(c_i)\rvert$			
k	r_k	$o_k(c_1)$	$o_k(c_2)$	$o_k(c_3)$	c_1	c_2	c_3	$c_1, o_k(c_1)$	$c_2, o_k(c_2)$	$c_3, o_k(c_3)$	MSE
1	0.4	1.00	4.00	9.00	1.00	3.50	8.00	0.00	0.50	1.00	0.417
2	0.5	1.00	4.33	9.67	1.00	4.00	9.00	0.00	0.33	0.67	0.067
3	0.3	1.00	4.67	10.33	2.00	4.50	11.00	1.00	0.17	0.67	0.044
4	0.2	0.57	4.74	10.05	0.50	5.00	10.50	0.07	0.26	0.45	0.004

us assume the first GP regression generates a function f_1 that produces the output vector $(1, 3.5, 8)$, and the absolute residual vector $(0, 0.5, 1)$. A constant $r_1 = 0.4$ is generated randomly and stored in f_1. From there, the new target output vector is calculated (Eq. 5.4), and is equal to $(1, 4.33, 9.67)$. The process continues until $MSE \leq 0.01$, as shown in Table 5.1. The column $output_k$ represents the target output points the regression needs to generate (when $k = 1$, they represent the original problem output), followed by the generated output ($f_k(c_i)$) and the residual generated by f_k ($|e'_k(c_i)|$) and the overall MSE.

5.5 Experiments

This section presents experimental results performed to test SSR. All tests are compared with the semantic GP (SGP) proposed in Moraglio et al. (2012) and a canonical GP (Koza 1994) in a set of polynomial regression problems. Given that one of the main characteristics of the method is to use the geometric semantic crossover to combine solutions sequentially discovered to solve the problem, we use the same testbed as Moraglio et al. (2012), composed by 8 univariate polynomials functions of degrees from 3 to 10, with real-valued coefficients uniformly drawn from $[-1, 1]$.

Table 5.2 Parameter values for the methods used in the experiments

Parameter	GP	SSR1	SSR2	SGP1	SGP2
Crossover Probability	0.9	0.9	0.9	1	1
Mutation Probability	0.1	0.1	0.1	1	1
Tournament Size	7	3	3	5	5
Population Size	1000	100	100	20	20
Number of Generations	100	50	100	5000	5000
Number of iterations	–	20	10	–	–
Initialization	–	–	–	YES	NO

In order to make the comparisons fair, all algorithms were given an execution budget of 100,000 evaluations, and the parameters used in each algorithm are detailed in Table 5.2. Note that, as SSR evolves a GP for k iterations, the sizes of populations vary across different algorithms, always respecting the evaluation budget. Because of that, different tournament sizes were used in order to balance selective pressure considering different population sizes. Notice that results of two versions of SSR and SGP are reported. In the case of SSR, the variation tests the trade-off between the number of generations of the canonical GP and the number of iterations of SSR.

For SGP, we used the same parameters reported in Moraglio et al. (2012), but varied the method used for population initialization. The first algorithm configuration (SGP1) initializes with polynomials of degree 10 (the same procedure used in Moraglio et al. (2012), while the initial population of SGP2 is randomly generated. One may argue that the assumption that we know the structure of the function we are looking for makes the use of symbolic regression unnecessary, which is true. However, the way geometric semantic crossover works depends heavily on the individuals in the initial population. If the genetic material we start with is not enough to produce the target function, mutation will probably not be able to insert enough modifications to the population to change this situation.

The experiments were performed in two phases. First, we ran the methods in a training set with 20 points. Then, we used the function discovered in the first phase in a second set of 20 points. The points were uniformly drawn from the $[0, 1]$ interval. All methods were executed 30 times. Table 5.3 shows the mean squared error (MSE) and standard deviation obtained by the three methods using different configurations.

Results are compared using a two-step approach. First, we apply Friedman's test with the null hypothesis $H_0 : \theta_1 = \theta_2 = ...\theta_5$, where θ_i represents the MSE of one of the algorithms tested. If H_0 is rejected we apply Nemenyi test (Demšar 2006) as a post-hoc procedure and make pairwise comparisons between the MSEs. Table 5.4 shows the results of the comparisons. The symbol ▲ indicates that the method in the column is statistically better than the method indicated in the row.

The results show that there is no evidence for statistical difference among the two versions of SSR. However there is statistical difference among the SGP versions, with SGP1 performing statistically better than SGP2. Concerning SSR, there is

Table 5.3 Average MSE (*average* [*standard error*]) for each algorithm in the training set, calculated over 30 runs

Problem	GP	SSR1	SSR2	SGP1	SGP2
polynomial3	0.000 [0.000]	0.000 [0.000]	0.000 [0.000]	0.000 [0.000]	0.009 [0.002]
polynomial4	0.000 [0.001]	0.000 [0.000]	0.000 [0.000]	0.000 [0.000]	0.009 [0.002]
polynomial5	0.001 [0.003]	0.000 [0.000]	0.000 [0.001]	0.000 [0.000]	0.013 [0.004]
polynomial6	0.001 [0.001]	0.000 [0.001]	0.000 [0.000]	0.000 [0.000]	0.010 [0.003]
polynomial7	0.002 [0.001]	0.001 [0.002]	0.000 [0.000]	0.000 [0.000]	0.008 [0.002]
polynomial8	0.002 [0.002]	0.000 [0.000]	0.000 [0.000]	0.000 [0.000]	0.009 [0.002]
polynomial9	0.005 [0.004]	0.001 [0.001]	0.001 [0.003]	0.000 [0.000]	0.010 [0.002]
polynomial10	0.002 [0.003]	0.001 [0.001]	0.001 [0.002]	0.000 [0.000]	0.010 [0.002]

Table 5.4 Pairwise Nemenyi test for MSE in the training set. The symbol ▲ indicates the method in the column is statistically better than the one in the row

	SSR1	SSR2	SGP1	SGP2
SSR2	–	–	–	–
SGP1	–	–	–	–
SGP2	▲	▲	▲	–
GP	–	–	–	–

no evidence of statistical difference regarding the GP or SGP1 and the results are statistically better than those obtained by SGP2. In summary, the results of the proposed approach are as good as the results of the GP and SGP1 and better than the results of SGP2.

Figure 5.5 shows the results of MSE for different iterations of SSR for the 8 functions tested using 50 and 100 generations over 20 iterations. The behaviour of the method is the expected one: as iterations go on, the error is reduced. As observed, in most cases the error converges as we approach 10 iterations. Hence, we can say that a different stopping criteria—such as convergence—could significantly reduce the number of evaluations required to obtain the reported results (note that we did not halt the algorithm and always allowed it to run for the maximum evaluation budget). A different parameter setting, where the GP ran for less generations at each iteration of the sequential procedure, combined with an effective stopping criteria might significantly reduce the fitness budget, making the use of SSR preferable over a single GP—these parameters can be tuned according to the problem at hand.

Table 5.5 presents the results of generalisation of the functions evolved in the training set and Table 5.6 the results for the Nemenyi test. The results show again that GP and SSR present no evidence of statistical difference. However, in this case, the results obtained by SSR are better than both versions of SGP. Looking at the values of MSE, we observe that SGP does not generalize well and has a tendency for overfitting. Therefore, these results show that SSR was successful in reducing the

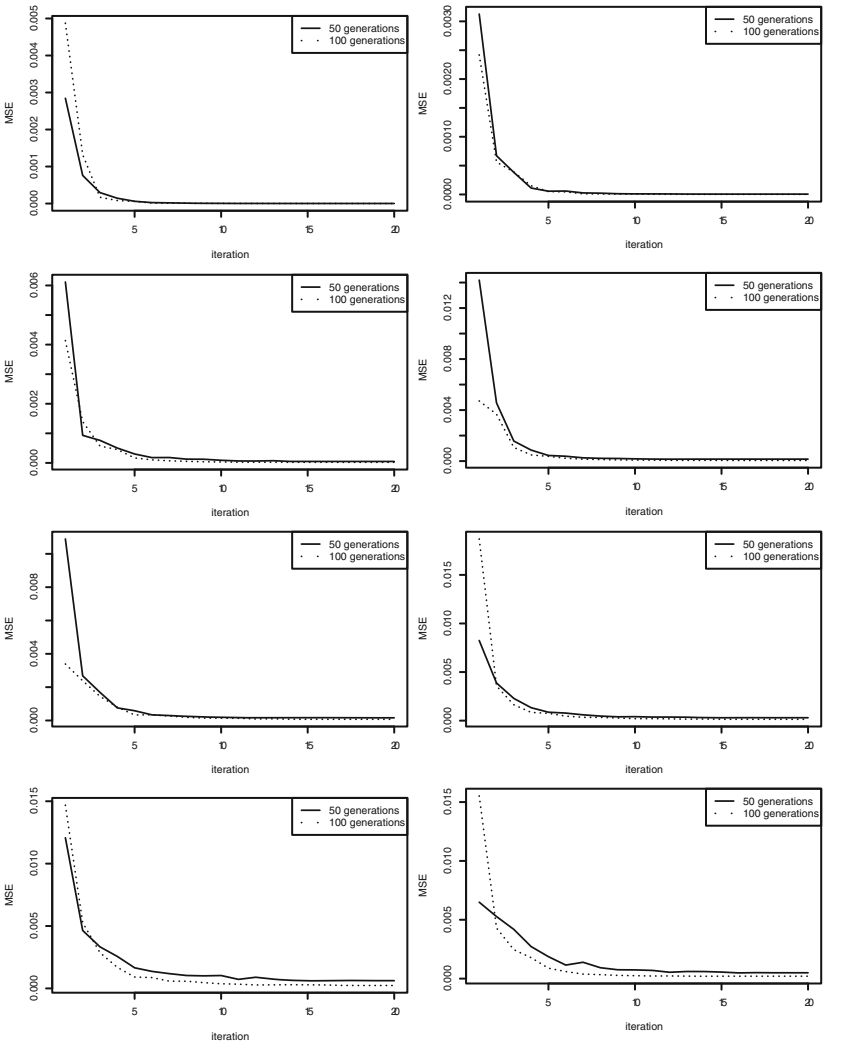


Fig. 5.5 Evolution of the error during iterations for both configurations of SSR for each problem, computed using the median of 30 runs

error of the symbolic regression problems and, at the same time, produced solutions with good generalisation power.

Regarding the comparisons with SGP, recall that the semantic operator has a completely different roles in the algorithms. For SGP, experiments showed that data overfitting (poor generalisation) can be a problem. Overfitting may be caused by the restrictions imposed by the geometric crossover, which combined with a semantic mutation designed to produce little semantic impact, makes SGP success heavily dependent on the initial population. This fact, combined with a small population

Table 5.5 Average MSE (*average* [*standard error*]) for each algorithm in the test set, calculated over 30 runs

Problem	GP	SSR1	SSR2	SGP1	SGP2
polynomial3	0.001 [0.001]	0.001 [0.004]	0.000 [0.001]	4.9e8 [2.6e9]	891.7 [2989.9]
polynomial4	0.001 [0.002]	0.000 [0.000]	0.000 [0.001]	84.33 [231.6]	5.360 [13.186]
polynomial5	0.007 [0.020]	0.001 [0.001]	0.001 [0.002]	8.158 [15.36]	7.158 [17.318]
polynomial6	0.008 [0.011]	0.003 [0.007]	0.002 [0.003]	1.2e5 [6.6e5]	9.350 [16.763]
polynomial7	0.009 [0.034]	0.001 [0.002]	0.001 [0.001]	41.27 [83.21]	6.005 [11.144]
polynomial8	0.004 [0.003]	0.001 [0.001]	0.001 [0.001]	117.0 [350.3]	13.497 [49.12]
polynomial9	0.014 [0.020]	0.006 [0.008]	0.003 [0.004]	43.66 [223.2]	2.811 [2.682]
polynomial10	0.032 [0.027]	0.013 [0.012]	0.011 [0.015]	58.64 [230.4]	3.574 [4.479]

Table 5.6 Pairwise Nemenyi test for MSE in the test set. The symbol ▲ indicates the method in the column is statistically better than the one in the row

	SSR1	SSR2	SGP1	SGP2
SSR2	–	–	–	–
SGP1	▲	▲	–	–
SGP2	▲	▲	–	–
GP	–	–	–	–

Table 5.7 Number of nodes (*average* [*standard error*]) of the resulting function for each algorithm, calculated over 30 runs

Problem	GP	SSR1	SSR2	SGP1	SGP2
polynomial3	50.5 [21.0]	1677.9 [242.5]	637.1 [165.3]	2.3e9 [1.2e9]	2.0e9 [1.3e9]
polynomial4	59.7 [28.0]	1720.7 [228.5]	635.2 [164.7]	2.0e9 [1.2e9]	2.3e9 [1.2e9]
polynomial5	68.1 [24.2]	1745.9 [195.9]	729.1 [133.8]	2.2e9 [1.4e9]	2.1e9 [1.2e9]
polynomial6	60.8 [26.9]	1664.9 [257.5]	691.6 [134.0]	1.9e9 [1.1e9]	2.0e9 [1.4e9]
polynomial7	63.2 [21.2]	1752.0 [170.8]	767.0 [140.4]	2.2e9 [1.1e9]	2.0e9 [1.2e9]
polynomial8	57.8 [28.8]	1644.1 [220.0]	712.1 [164.5]	2.1e9 [1.3e9]	2.2e9 [1.4e9]
polynomial9	49.4 [22.1]	1736.7 [197.5]	771.9 [154.9]	1.9e9 [1.3e9]	2.4e9 [1.1e9]
polynomial10	62.6 [24.2]	1786.6 [170.1]	784.3 [142.0]	2.0e9 [1.3e9]	2.0e9 [1.3e9]

size, can make it difficult for SGP to find a good solution. Even if such a solution is found, it will usually be much more complex than those produced by SSR, also potentially leading to overfitting, something that has been observed when analysing the size of the evolved solutions.

Table 5.7 presents the average number of nodes and standard deviation of the final solutions found by each algorithm. The size of SSR1 and SSR2 solutions reflect approximately the number of GP executions within the algorithm, i.e. it is 20 and 10 times the number of nodes of the solutions generated by the canonical GP,

respectively. The size of the functions generated by both SGP versions, on the other hand, are at least 10^6 times greater than the other methods, since the size of SGP individuals grows exponentially in the number of generations. Note that while SSR performs as many semantic crossovers as iterations, for SGP this number depends on the number of individuals, crossover probability and number of generations. The difference in size of the solutions found by SSR1 and SSR2 can be explained by the number of iterations of the sequential procedure: while SSR1 has a total of 20, SSR2 has a total of 10 (Table 5.2). This illustrates the impact of the number of crossover operations—iterations of the sequential procedure in the case of SSR—on the size of the solutions. At the same time, we don't see a big impact on the performance of the SSR algorithm, since the error is minimised after 10 iterations in most cases—as illustrated in Fig. 5.5.

5.6 Conclusions and Future Work

This chapter proposed Sequential the Symbolic Regression (SSR), a new strategy to perform symbolic regression by iteratively learning solutions from a transformed set of problems. The definition of the problem changes according to the semantic distance (or error rate) generated from the desired and obtained outputs, and different (sub-) problem solutions are put together using a geometric semantic crossover operator. The use of the semantic operator guarantees the solutions generated are never worse than the weakest of their parents.

Experiments were run on a set of eight polynomial functions and results compared with a canonical GP and a geometric semantic GP (SGP). When compared with SGP, which has a problem of exponential growth of its individuals, SSR has the advantage of generating smaller solutions that are less prone to overfitting. Regarding the canonical GP, the method has the potential of improving solutions even when the algorithm has already converged, by transforming the original problem into a new one.

Experimental results showed SSR presents MSE values that are statistically better than those generated by the solutions evolved by SGP, specially when a test set of points is used to evaluate the generalisation of the method. When compared with GP, there is no evidence of statistical difference among the results. However, we believe the results can still be improved to use a minimal computational budget (fitness evaluations).

For future work, a more complete study of the impact of the parameters in SSR needs to be performed, specially investigating what is the impact of running the GP for longer or SSR for more iterations. The method also needs to be validated on more complex symbolic regression problems, such as those suggested as GP benchmarks (White et al. 2013). Finally, other methods for combining different solutions are worth further investigation.

References

Angeline PJ, Pollack JB (1992) Evolutionary induction of subroutines. In: Proceedings of the 14th Annual Cognitive Science Conference, pp 236–241

Angeline PJ, Pollack JB (1994) Coevolving high-level representations. In: Langton C (ed) Artificial life III. Addison-Wesley, Reading, pp 55–71

Christensen S, Oppacher F (2007) Solving the artificial ant on the Santa Fe trail problem in 20,696 fitness evaluations. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, London, vol 2, pp 1574–1579

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Jackson D, Gibbons AP (2007) Layered learning in Boolean GP problems. In: Proceedings of the 10th European Conference on Genetic Programming, Lecture Notes in Computer Science, vol 4445, pp 148–159. Springer, Valencia

Keijzer M, Ryan C, Cattolico M (2004) Run transferable libraries—learning functional bias in problem domains. In: Genetic and evolutionary computation – GECCO-2004, Part II, Lecture Notes in Computer Science, vol 3103, pp 531–542. Springer, Seattle

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

Koza JR (1992b) Hierarchical automatic function definition in genetic programming. In: Whitley LD (ed) Foundations of genetic algorithms 2.Morgan Kaufmann, Vail, pp 297–318

Koza JR (1994) Genetic programming II: automatic discovery of reusable programs. MIT, Cambridge

Lee GY (1999) Genetic recursive regression for modeling and forecasting real-world chaotic time series. In: Advances in genetic programming III. MIT, Cambridge, pp 401–423

McKay B (2000) Partial functions in fitness-shared genetic programming. In: Proceedings of the 2000 Congress on Evolutionary Computation CEC00, IEEE Press, La Jolla Marriott Hotel La Jolla, California , USA, vol 1, pp 349–356

Moraglio A, Krawiec K, Johnson C (2012) Geometric semantic genetic programming. In: Parallel problem solving from nature - PPSN XII, Lecture Notes in Computer Science, vol 7491, pp 21–31. Springer, Berlin

Otero FEB, Johnson CG (2013) Automated problem decomposition for the boolean domain with genetic programming. In: Proceedings of the 16th European Conference on Genetic Programming, Euro GP 2013, Vienna, Austria, pp 169–180

Roberts SC, Howard D, Koza JR (2001) Evolving modules in genetic programming by subtree encapsulation. In: Genetic programming, Proceedings of EuroGP'2001, Springer-Verlag, Lake Como, Italy, LNCS, vol 2038, pp 160–175

Rosca JP, Ballard DH (1994) Learning by adapting representations in genetic programming. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence, IEEE Press, Orlando, Florida , USA, vol 1, pp 407–412

Spector L, Harrington K, Martin B, Helmuth T (2011a) What's in an evolved name? the evolution of modularity via tag-based reference. In: Genetic programming theory and practice IX, Genetic and Evolutionary Computation. Springer, Ann Arbor, pp 1–16

Spector L, Martin B, Harrington K, Helmuth T (2011b) Tag-based modules in genetic programming. In: GECCO '11: Proceedings of the 13th annual conference on genetic and evolutionary computation, ACM, Dublin, Ireland, pp 1419–1426

Spector L, Harrington K, Helmuth T (2012) Tag-based modularity in tree-based genetic programming. In: GECCO '12: Proceedings of the 14th international conference on Genetic and evolutionary computation conference, ACM, Philadelphia, Pennsylvania, USA, pp 815–822

Uy NQ, Hoai NX, ONeill M, McKay RI, Galván-López E (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet Programm Evol Mach 12(2):91–119

Vanneschi L, Castelli M, Manzoni L, Silva S (2013) A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: Proceedings of the 16th European Conference on Genetic Programming, Euro GP 2013, Vienna, Austria, vol 7831, pp 205–216

Vanneschi L, Castelli M, Silva S (2014) A survey of semantic methods in genetic programming. Genet Programm Evol Mach 15(2):1–20

Walker JA, Miller JF (2008) The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. IEEE Trans Evol Comput 12(4):397–417

White D, McDermott J, Castelli M, Manzoni L, Goldman B, Kronberger G, Jakowski W, OReilly UM, Luke S (2013) Better GP benchmarks: community survey results and proposals. Genet Programm Evol Mach 14(1):3–29

Luiz Otávio V.B. Oliveira is currently working towards the Ph.D. degree at the Universidade Federal de Minas Gerais, Brazil. He received the B.Sc. degree in computer science from the Universidade Federal de Itajubá, Brazil, in 2010; and the M.Sc. degree in computer science from the Universidade Federal do Rio Grande do Sul, Brazil, in 2012. His research interests include genetic programming, artificial immune systems and evolutionary computation applied to generative meta-learning.

Fernando E.B. Otero is a Lecturer in the School of Computing at the University of Kent, UK. He received a BSc in Computer Science from Pontifícia Universidade Católica do Paraná, Brazil, in 2002; and a PhD in Computer Science from the University of Kent, UK, in 2010. He is a member of the Computational Economics and Multi-Agent Systems (CEMAS) Laboratory at the University of Kent. His research interests include biologically inspired algorithms (mainly genetic programming and ant colony optimization), bioinformatics, and data mining.

Gisele L. Pappa received her PhD from the University of Kent in 2007 and is an Associate Professor at the Computer Science Department at Universidade Federal de Minas Gerais, Brazil. Her main research interests are on data mining algorithms, evolutionary computation, and applications of these areas in social networks and bioinformatics.

Julio Albinati is a graduate student at the Universidade Federal de Minas Gerais, Brazil, currently pursuing his Master's degree. He received a BSc in Computer Science from the Universidade Federal de Minas Gerais in 2014. His research interests include machine learning, biologically inspired algorithms and data mining.

Chapter 6
Sliding Window Symbolic Regression for Detecting Changes of System Dynamics

Stephan M. Winkler, Michael Affenzeller, Gabriel Kronberger, Michael Kommenda, Bogdan Burlacu and Stefan Wagner

6.1 Introduction

The idea of sliding window behavior in computer science is not novel; in machine learning, drifting concepts are often handled by moving the scope (of either fixed or adaptive size) over the training data (see for example (Widmer and Kubat 1996) or (Hulten et al. 2001)). The main idea is the following: Instead of considering all training data for training and evaluating models, the algorithm initially considers only the first part of the data. Then, after executing learning routines on the basis of this part of the data, the range of samples under consideration is shifted by a certain offset. Thus, the window of samples considered is moved; it slides over the training data.

Among modern data modeling techniques, symbolic regression using genetic programming (GP) distinguishes itself by its ability to identify nonlinear models

S. M. Winkler (✉) · M. Affenzeller · 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: stephan.winkler@heuristiclab.com

S. M. Winkler · M. Affenzeller · M. Kommenda · B. Burlacu
Institute for Formal Models and Verification, Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria

M. Affenzeller
e-mail: michael.affenzeller@heuristiclab.com

G. Kronberger
e-mail: gabriel.kronberger@heuristiclab.com

M. Kommenda
e-mail: michael.kommenda@heuristiclab.com

B. Burlacu
e-mail: bogdan.burlacu@heuristiclab.com

S. Wagner
e-mail: stefan.wagner@heuristiclab.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_6

which can be interpreted by domain experts. In comparison to other nonlinear modeling techniques such as artificial neural networks or support vector machines, the representation of solutions as mathematical terms allows to systematically analyze and interpret the influence of input variables with respect to the output variable (Vladislavleva et al. 2010; Affenzeller et al. 2013).

The combination of sliding window approaches and GP based structure identification (Zuo et al. 2004; Wagner et al. 2007) is not frequently discussed in the literature; in general, GP is seen as an explicitly offline, global optimization method (Koza 1992) working on all available training samples. Nevertheless, during research activities in the field of online systems identification (Winkler et al. 2007a, b), we discovered several surprising aspects. In general, online GP was able to identify symbolic regression models describing a Diesel engine's emissions remarkably fast; the even more astonishing fact was that these models were even less prone to overfitting than those created using standard methods.

Picking up this line of research, we here investigate the abilities of sliding window symbolic regression (SWSR) to detect changes in data characteristics occurring when the underlying system dynamics change. Here we focus on the analysis of specific algorithm variants, especially generations triggered and selection pressure triggered SWSR, regarding their ability to detect significant changes of the analyzed systems' dynamics. In real-world systems, several causes might change a systems characteristic; usually, the causes cannot be observed directly and must be considered as unknown external effects leading to non-stationary system dynamics. In this chapter, we focus on switching system dynamics, where influencing factors of a system's behavior might appear or disappear or change gradually, while other influencing factors remain unaffected. Other common causes for changes of system dynamics are gradually changing influences or noise levels over long time periods as well as changes of the volatility of factors.

We define new analysis methods for detecting such phenomena by observing the characteristics of SWSR when moving over the data. When entering a data partition with new characteristics, algorithm parameters such as model quality and selection pressure are expected to change significantly. Moreover, we present specialized analysis techniques for SWSR. In the experimental part of this contribution we present heat map visualizations of the performance of the collected models on all sliding window positions.

For testing we introduce special benchmark data sets simulating systems with changing dynamics including abrupt as well as gradual appearance or disappearance of major influence factors.

6.2 Sliding Window Symbolic Regression

Sliding window evaluation means that during the run of the algorithm, only a portion or slice of the training data is available to the algorithm. By changing the position of the sliding window according to the rules described below, we determine which

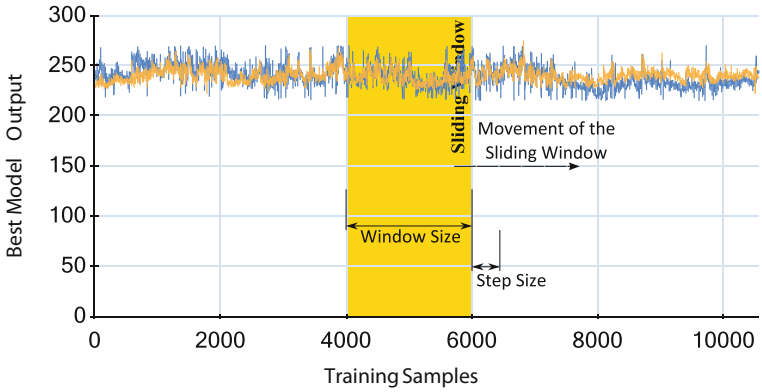


Fig. 6.1 Sliding window evaluation displaying the current sliding window, the output of the best model on the current sliding window (*yellow*) and the true values of the dependent variable (*blue*)

part of the training data is visible to the algorithm. This shall allow the evolutionary system to adapt gradually only to those patterns present in the whole of the training data, leading to more compact solutions with better generalization capabilities.

From an algorithmic perspective, the population of candidate solutions can be considered as some kind of implicit memory that stores the models which are able to adapt sufficiently well to the current sliding window. As the window slowly slides over the data during the algorithm execution, we assume that those models which are able to explain the data for a longer time should have a higher survival probability and are therefore expected to remain in the population, whereas those models which only describe the current window are more likely to disappear. If this assumption holds, after the sliding window has moved over the entire data-set, the resulting models should be those which were able to explain correlations that appear in the entire data set.

A sliding window approach, as illustrated in Fig. 6.1, is characterized by a trigger (criteria or condition for moving the window), a step-size and a window-size. The trigger can either be generational (slide the window every n generations) or based on the selection pressure value (when using offspring selection). Depending on the window update settings, the data windows used for training may overlap.

In the following subsections we propose those variants of sliding window symbolic regression that are used in the later sections for analyzing their respective abilities to detect and explain dynamic changes of a system.

6.2.1 Generations Triggered Sliding Windows GP

In the most simple variant, which is summarized in Alg. 2, the given data set is split into partitions. Initially, only the first partition of data is used for evaluating models. After a certain number of generations, the focus shifts and the next data partition is

used for evaluation; we here keep the beforehand trained models and continue the evolutionary process using the previously generated genetic material. Optionally, the window of the considered samples might grow over time. This procedure of training and shifting the focus in equidistant intervals is repeated until all partitions have been considered and the best model found so far is returned as the result of the whole GP process.

The concept of generations triggered sliding windows can be combined with different variants of GP, such as standard GP or offspring selection genetic programming (OSGP, detailed in Sect. 6.2.2). If OSGP with generations triggered sliding window is used, the selection pressure gives an indication for how difficult it is for the algorithm to produce better models; the lower the selection pressure, the less fit are the models on the given data. Thus, the selection pressure seen after shifting the focus gives an indication whether the behavior learned on previously considered samples is able to explain the new data—or if there might be a change of the behavior of the analyzed system.

Algorithm 2 Generations Triggered Sliding Windows GP

Input: Data, GenerationsTrigger, WindowStepSize, WindowSize
  Initialize population
  Generations$\leftarrow 0$
  $\text{Index}_1 \leftarrow 0$
  $\text{Index}_2 \leftarrow$ WindowSize
  while $\text{Index}_2 \leq$Data.Length **do**
    Generate offspring from selected parents
    Evaluate offspring on Data$[\text{Index}_1, \ldots, \text{Index}_2[$
    Generations $\leftarrow$ Generations $+1$
    if Generations *mod* GenerationsTrigger $= 0$ **then**
      $\text{Index}_2 \leftarrow \text{Index}_2 +$ WindowStepSize
      $\text{Index}_1 \leftarrow \max(0, \text{Index}_2 - \text{WindowSize})$
    end if
  end while
  return Best model

6.2.2 *Selection Pressure Triggered Sliding Window GP*

One of the most important problem independent concepts used in our implementation of GP-based structure identification is offspring selection (Affenzeller et al. 2009), an enhanced selection model that has enabled genetic algorithms and genetic programming implementations to produce superior results for various kinds of optimization problems. As in the case of conventional GAs or GP, offspring is generated by parent selection, crossover, and mutation. In a second (offspring) selection step (as detailed in Fig. 6.2), only those children become members of the next generation's population that outperform their own parents, all other ones are discarded. The algorithm therefore repeats the process of creating new children until the number of successful offspring is sufficient to create the next generation's population.

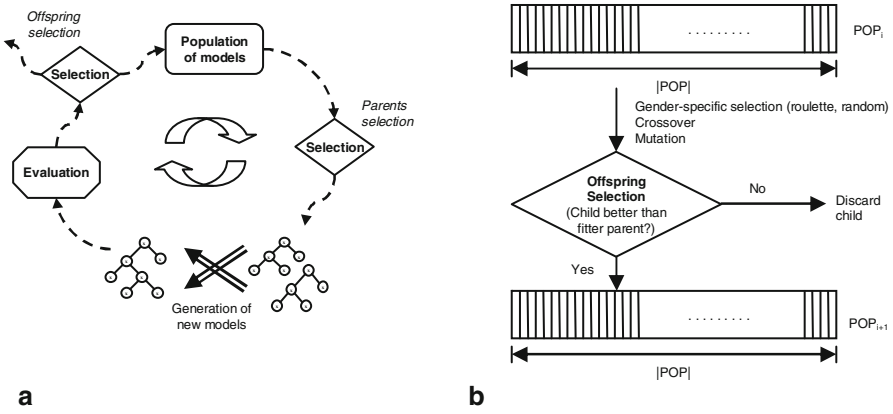


Fig. 6.2 **a** Genetic programming including offspring selection; **b** Embedding a simplified version of offspring selection into the GP process

Within this selection model the selection pressure is defined as the ratio of generated candidates to the population size:

$$\text{SelectionPressure} = \frac{|\text{Generated Offspring}|}{|\text{Population}|}$$

The higher these values becomes, the more models have to be created and evaluated in order to produce enough models that form the next generation's population. In other words, the selection pressure indicates how hard it is for the algorithm to produce a sufficient number of successful solution candidates.

The proposed idea is to reduce the amount of data that is available for the algorithm as identification data. As the identification process is executed, better and better models are created which leads to a rise of the selection pressure; as soon as the selection pressure reaches a predefined threshold value, the limits of the identification data are shifted and the algorithm goes on considering another part of the available identification data set. This procedure is then repeated until the sliding window has reached the end of the training data and hence, all training data have been considered. A benefit of evaluating the models on a much smaller data set, the runtime of the algorithm is significantly reduced.

In Algorithm 3 we give a sketch of the sliding window GP based structure identification process incorporating offspring selection. The standard GP parameters (as, for example, population size, mutation rate and crossover operator combinations) are hereby omitted; we only describe the sliding window specific process modifications. As soon as the current selection pressure reaches *SelectionPressureTrigger*, the window is moved by *WindowStepSize* samples. The *WindowSize* parameter specifies the maximum size of the current training data scope. This procedure is repeated until the end of the data set is reached.

Algorithm 3 Selection Pressure Triggered Sliding Window GP

Input: Data, SelectionPressureTrigger, WindowStepSize, WindowSize
  Initialize population
  Generations$\leftarrow 0$
  $\text{Index}_1 \leftarrow 0$
  $\text{Index}_2 \leftarrow$ WindowSize
  while $\text{Index}_2 \leq$Data.Length **do**
    Generate offspring from selected parents
    Evaluate offspring on Data$[\text{Index}_1, \ldots, \text{Index}_2[$
    Perform offspring selection
    SelectionPressure $\leftarrow$ reciprocal of the ratio of successful offspring
    if SelectionPressure $\geq$ SelectionPressureTrigger **then**
      $\text{Index}_2 \leftarrow \text{Index}_2 +$ WindowStepSize
      $\text{Index}_1 \leftarrow \max(0, \text{Index}_2 - \text{WindowSize})$
    end if
  end while
  return Best model

6.3 Analysis Measures

6.3.1 Sliding Window Qualities

In some situations, depending on sliding window size and movement speed, the population runs the risk of becoming too specialized on the current training partition, losing its ability to adapt to changing conditions due to the loss of genetic diversity. In order to identify and avoid such cases in which evolution is hindered, we introduce an analysis method that calculates the best solution's quality not only on the current sliding window, but also on the past training data. Ideally, the algorithm should produce solutions that perform well on the whole training data up to the current sliding window position. We can determine if the best solution becomes too specialized if its quality on past data is much worse than its quality on the current sliding window. Conversely, if the qualities are similar, we can say that the solution of the algorithm captures the intrinsic characteristics of the data.

In order to analyze the algorithm's behavior we also consider future data (the region of the training partition that the sliding window has not reached yet) in the best solution qualities analysis.

Figure 6.3 shows an exemplary evaluation using this analyzer as implemented in HeuristicLab 3.3.

We consider sliding window solutions *stable* if they maintain a good quality on past data, as well as current data. If the *before window* and *current window* quality curves converge, then the algorithm has successfully extracted the important patterns from the data. Ideally, for stable systems characteristics the *before window*, *after window* and *test* qualities will converge. If the quality curves do not converge, this could be an indication that the chosen parameter settings are not appropriate for the given data (sliding window too small, too slow to move, etc.). Additionally, variations in the convergence of the quality curves can also indicate changes in the data.

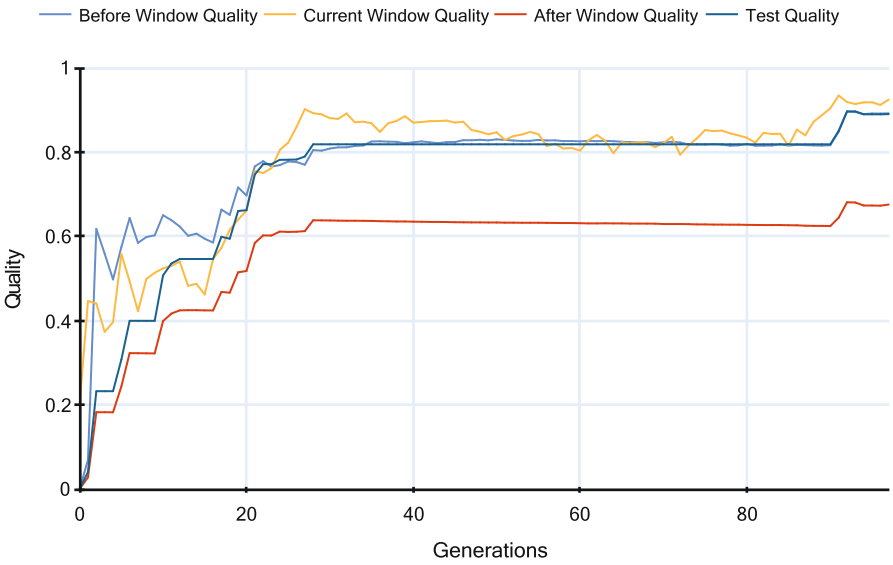


Fig. 6.3 Sliding window qualities, analyzed in HeuristicLab. In this case sliding window genetic programming with strict offspring selection was used to solve the Poly-10 problem (Poli 2003); the population size was set to 100, and the selection pressure trigger was applied with *SelectionPressureTrigger* = *15*

6.3.2 Sliding Window Best Solutions

To gain further insight about the data itself and the evolution process, we save the best models on the current sliding window each time before advancing its position. This is particularly useful for problems where the data is sequential and changing over time. The resulting collection of best models is displayed as a colored table (Fig. 6.4) in which warmer colors represent higher qualities. On the horizontal axis the sliding window positions are given and on the vertical axis from top to bottom the best solutions for the respective window are listed. The color of the cell represents the quality of each model when evaluated on all windows, before (left) and after (right) the current window (diagonal).

As an alternative view, we provide an alternate representation as a heat map which shows the quality of each model (Pearson's R^2) for each sliding window position. This view which shows the "big picture" about the data and the generated models, can make it easier to identify the regions where changes in the data or in the population of models occurred. Figure 6.5 shows a heat map view for the qualities of 100 models on 100 sliding windows.

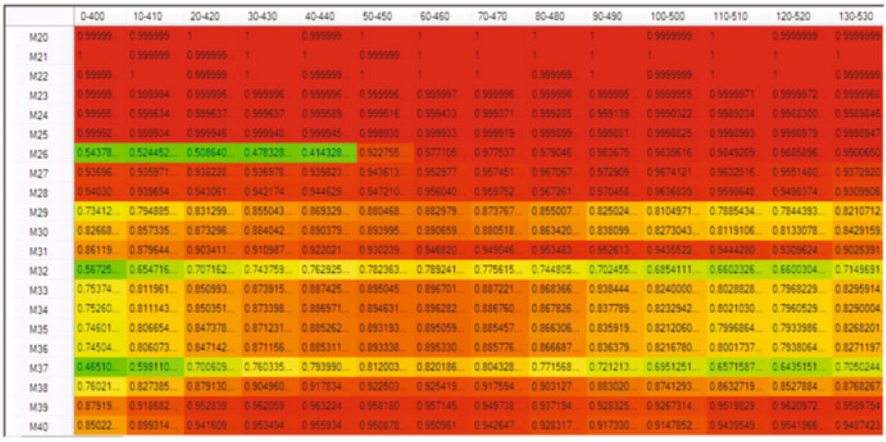


Fig. 6.4 Sliding window best solutions table. The x-axis shows the sliding window positions while the y-axis shows the best solutions. Each cell in the table is colored according to quality (Pearson's R^2), warmer colors represent better qualities

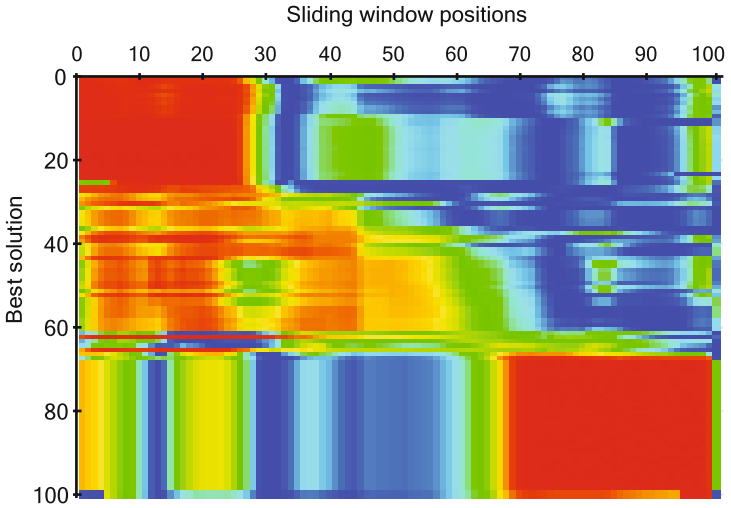


Fig. 6.5 Sliding window best solutions heat map. This representation is not as detailed as the one in Fig. 6.4, but it provides a more general picture: in this case, the sliding window has been moved 100 times and 100 best solutions were recorded. The heat map is also colored according to the Pearson R^2 quality measure

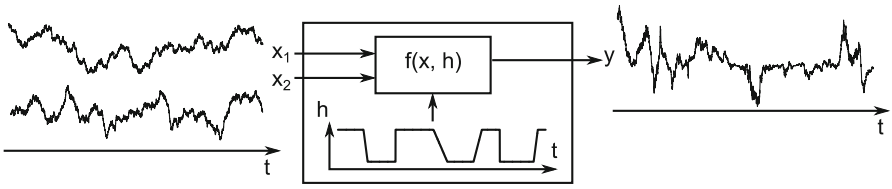


Fig. 6.6 The system output is described via a stationary transfer function using the external inputs and the hidden state as inputs. The system state changes are controlled via the hidden state variable

6.4 Experiments

6.4.1 Design of Experiments

For testing the described approach we created a set of synthetic data sets with controlled system dynamics. In the following we only discuss simulated systems with multiple input variables and one output variable without feedback loops. These systems can be easily described via a transfer function mapping input values to the output value. The system state changes are modeled using a hidden time-dependent state variable h_t. The hidden state variable is set to real values in the range [0,1] and is used to control which parts of the transfer function are active at each time point. Figure 6.6 shows an example system with two inputs and the hidden state.

To simulate a time-dependent process we generated the input variables x_i by sampling from a uni-variate zero mean Gaussian process using a squared exponential covariance function (Rasmussen and Williams 2006). Each input vector $x_i, i \in [1..10]$ has been sampled independently and has 5,000 elements. Each vector is scaled to unit variance.

$$\mathbf{t} := \left[\frac{i}{200}, i \in [0 \ldots 5000[\right]$$

$$K_{i,j} := \exp\left(-\frac{1}{2}(t_i - t_j)^2 \right)$$

$$\mathbf{x}_i \overset{i.i.d}{\sim} \mathrm{GP}(\mathbf{0}, K)$$

The covariance matrix K is Toeplitz, which means that each descending diagonal from left to right is constant. Thus, sampling from this Gaussian process can be implemented efficiently using the O(n) Cholesky transform for Toeplitz systems.

$$\alpha_i \overset{i.i.d}{\sim} N(0, 1)$$

$$L := \text{toeplitz-cholesky}(K)$$

$$\mathbf{x} := L\alpha$$

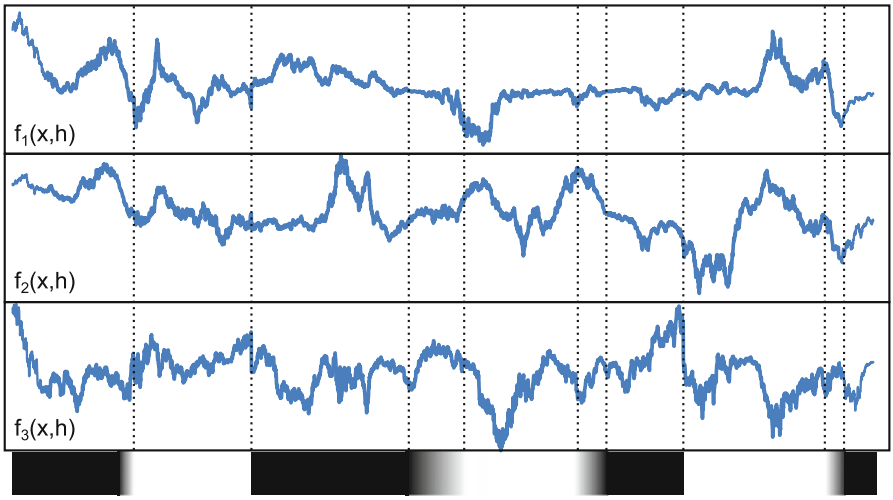


Fig. 6.7 The output targets and the time points of gradual and abrupt system change points

Examples for two sample inputs are shown on the left side of Fig. 6.6. The generated samples exhibit long term changes and short term fluctuations. Alternatively, input variable values could be generated by simply sampling from a simple auto-regressive (e.g., AR(1)) process, which is a special case of a Gaussian process.

The values for the hidden state variable h have been created manually to model several state switches, including slow gradual changes over long time intervals and abrupt jumps in very short time intervals. In all experiments and all problem instances we used the same hidden state vector to make it possible to compare the results of multiple runs and for different problem instances visually.

In the experiments several transfer functions have been used which are shown below. These functions include a very easy function ($f_1(x,h)$), a function of medium hardness ($f_2(x,h)$), and one harder function ($f_3(x,h)$). While all functions are static, the dynamic system behavior is induced through the time-dependent state variable h. It should be noted that all considered functions are relatively easy when compared with real-world data sets.

$$f_1(x,h) = x_1 * (h * x_2 + (1-h) * x_3) \tag{6.1}$$

$$f_2(x,h) = \alpha_2(x_1x_2 + x_3x_4) + \beta_2(h * x_5 * x_6 + (1-h) * x_7 * x_8) \tag{6.2}$$

$$f_3(x,h) = \alpha_3(x_1x_2x_3 + x_4x_5x_6) + \beta_2(hx_7x_8 + (h-1)x_9x_{10}) \tag{6.3}$$

To control the relative influences of the fixed part and the variable part of each function, scaling factors α and β are set so that both parts have equal variance. The state changes induced by the hidden variable are not visible in the resulting outputs as shown in Fig. 6.7.

To test our hypothesis if it is possible to detect changes in system dynamics using the sliding window approach we performed experiments using the three data sets

Table 6.1 Algorithm configurations

	Alg-1	Alg-2	Alg-3
Population size	500	500	500
Crossover rate	100 %	100 %	100 %
Mutation rate	25 %	25 %	25 %
Tree size (height/nodes)	12/50	12/50	12/50
Parent selection	Tournament (size = 4)	Gender-specific (prop./random)	Gender-specific (prop./random)
Offspring selection	– –	Strict (comp. factor) = 1.0	Strict (comp. factor) = 0.5
Sliding window size	400	400	400
Sliding window trigger	Generation	Generation	Sel. pressure over 4
Sliding window step size	10	10	10

described above and three different algorithm configurations. The first configuration uses tournament parent selection and simple generational replacement. The second configuration uses gender-specific parent selection and strict offspring selection. The third configuration also uses strict offspring selection but additionally uses the selection pressure to trigger sliding events. Table 6.1 shows the configurations of all algorithms.

Gradual and continuous adaptation of the population to the changing conditions in the data set can be facilitated when the sliding window is moved slowly and smoothly over the data set. Therefore, the window has been moved after each generation by 10 data points in our experiments. This has the effect that the algorithm produces a large number of different solutions for the whole run (one for each window). The speed of sliding is crucial and has to be adapted for each problem instance. On the one hand, if the window is moved too quickly, then the population might not able to adapt to the changing conditions quickly enough. On the other hand, if the window is moved too slowly the population might converge to a good solution for the current window, leading to a loss in genetic diversity which hampers the ability to adapt to the new data in the next window.

In our experiments we move the window by 10 samples either in each generation or when the selection pressure limit is reached. The sliding window size is 400 in all configurations. Therefore, the number of generations is 500 when using a generational trigger but can be larger for the selection pressure triggered algorithm.

6.4.2 *Discussion of Results*

The results of our experiments show, that for the simple problem (f_1), it is possible to detect the changes of system dynamics easily with all three algorithm configurations. The changes in system dynamics can be detected in the heat maps for all three

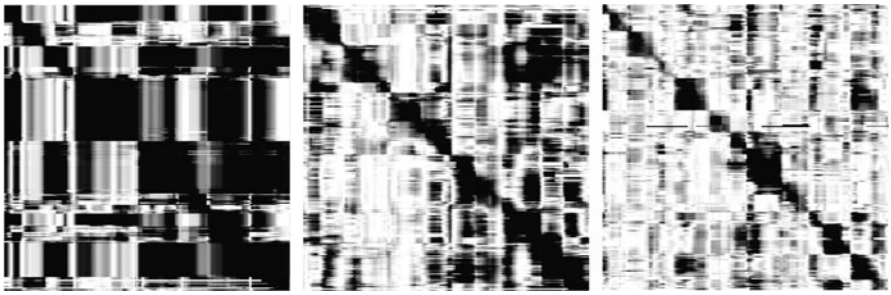


Fig. 6.8 Best solutions heat map for standard GP with generational sliding window applied on benchmark problems (*left*: f_1, *middle*: f_2 *right*: f_3). See Sect. 6.3.2 for the description of the visualization

test problems. For the harder problems (f_2 and f_3), the simple GP algorithm leads to worse results compared to the variants using offspring selection. When using offspring selection, more effort is spent in each generation to find individuals that perform well on the current data window compared to the standard GP algorithm.

For the first two algorithm variants the speed of the sliding window has to be adapted manually to prevent early convergence to a solution with high accuracy and simultaneous loss of genetic diversity. This is especially problematic if the system dynamics are changing slowly or seldomly. In our experiment runs, this happened several times and several retries have been necessary until we found good parameters for the sliding speed. In the third algorithm variant, the speed of the sliding window is dynamically adjusted based on selection pressure and need not be configured manually. However, in this case, the selection pressure threshold must be tuned.

In the following the results of all algorithm variants are discussed and compared.

6.4.2.1 Standard GP with Generational Sliding

Figure 6.8 shows the heat maps of the qualities of the best solutions for each window. In the heat map on the left side the state changes for f_1 are clearly visible. It is also possible to discern the areas in the data set in which the system dynamics change gradually over a longer time interval. Due to these windows, the system behavior cannot be modeled accurately with the available data. The dark squares along the diagonal represent the areas of the data set where the best solution from the current window also matches the windows before and after the current window. The heat map shows six squares along the diagonal representing the six phases of stable behavior of the system shown in Fig. 6.7. The last phase is not detected correctly because the window size (400) is larger than the number of samples in the last phase. In the left heat map, it is also visible that the solutions detected on the later part of the data describe earlier phases accurately (dark rectangles in the lower triangle).

For the harder problems the results are much worse. For these problems, the different phases can be hardly discerned in the heat maps. Instead, the algorithm is only able to produce accurate solutions for the current window only (visible as dark

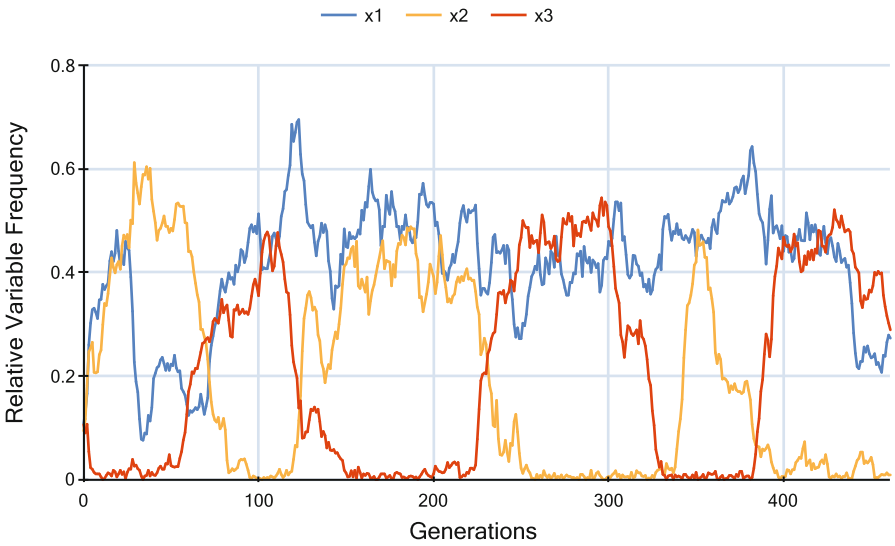


Fig. 6.9 Variable frequencies for problem 1 obtained by GA sliding windows

squared along the diagonal) and these solutions do not generalize well for the rest of the data.

In the variable frequency chart shown in Fig. 6.9, it is also clearly visible that the algorithm detects the relevant variables for the different phases while sliding the window over the data set.

6.4.2.2 GP with Offspring Selection and Generational Sliding

The benefit of this configuration is that the algorithm dynamically spends more effort to adapt the population to the data in the current window. Compared to the standard configuration the selection pressure is adapted dynamically in order to fill a population with better individuals.

Figure 6.10 again shows the heat maps for the three data sets. For the first data set shown on the left hand side the results are very similar. However, for the harder problems the results are much better with offspring selection and the phase changes can be discerned rather easily in the heat maps.

6.4.2.3 GP with Offspring Selection and Selection Pressure Triggered Sliding

In this configuration, the window is moved when an upper threshold for the selection pressure is reached. Using an upper limit prevents that too much effort is spent on only one window.

Figure 6.11 shows the heat maps produced by this algorithm configuration for the three data sets. Especially, for the third data set on the right hand side, this configuration produced very good results. The results for the second data set shown

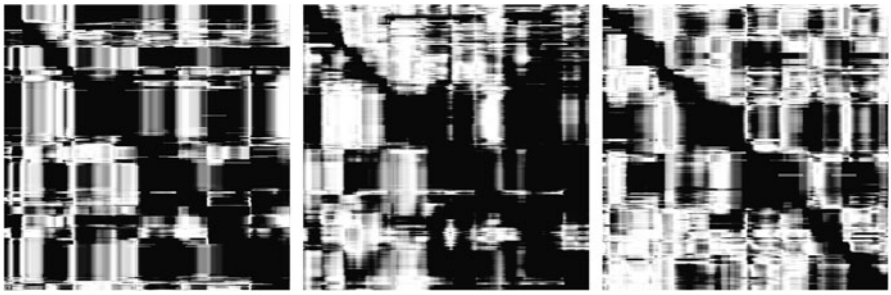


Fig. 6.10 Best solutions heat map for OSGP with generational sliding window applied on benchmark problems (*left*: f_1, *middle*: f_2, *right*: f_3). See Sect. 6.3.2 for the description of the visualization

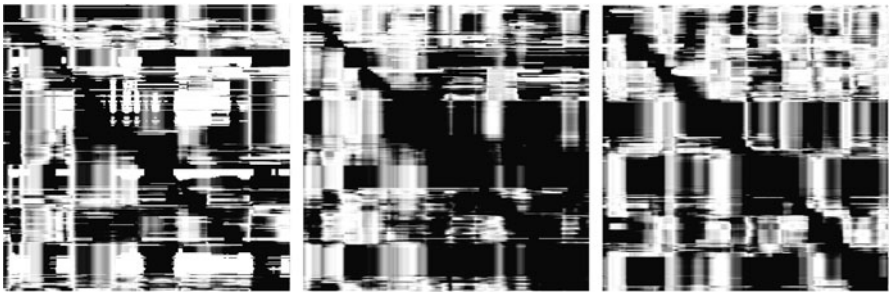


Fig. 6.11 Best solutions heat map for OSGP with selection pressure sliding window applied on benchmark problems (*left*: f_1, *middle*: f_2, *right*: f_3). See Sect. 6.3.2 for the description of the visualization

in the middle are similar to those visualized in Fig. 6.10. For this data set, the phase changes are not visible clearly.

The heat map produced for the first problem shows several white fragments which are caused by overly adapted models which produce extreme predictions. This problem instance is very easy, therefore over-adaptation occurred with the selected parameter settings.

In this configuration, the population is allowed to adapt to the data in the current window as long as this is possible; the effort is limited by the selection pressure threshold. This has the effect that the window moving speed is adapted according to the ability of the population to improve for the current window. When the system dynamics do not change over long time intervals, this has the effect that the window is moved quickly until a phase change is reached. At the phase change, the data in the window changes and the population must adapt. At this point, selection pressure drops and the window is moved more slowly.

The crucial parameter for this algorithm configuration is the selection pressure threshold which it has to be fine-tuned manually. On the one hand the window must not move too quickly to allow adaptation of the population to the current window, and

on the other hand the window must not move too slowly to prevent loss of diversity. Usually, it is necessary to hand-tune this parameter for each problem instance based on several tries.

6.5 Conclusion and Outlook

In this chapter we have described three variants of sliding window based symbolic regression with the aim to detect changing systems dynamics. Two of these variants move the sliding window continuously over the data stream, whereas the third variant uses an adaptive, selection pressure based trigger. In order to emulate changing environmental conditions we have used several benchmark data sets in which certain terms of the generating function appear and disappear dynamically (ad-hoc as well as continuously). All discussed algorithm variants have been analyzed using specific analysis measures introduced in this chapter, and we have shown that these analysis methods enable the detection and visualization of changes in systems characteristics.

In terms of practical applicability to real-world data, continuous sliders are computationally less expensive and can therefore be considered for online detection of changing environmental conditions. The selection pressure triggered sliding window approach, on the other hand, is better suited for a-posteriori analysis of more complex dynamic systems. Especially in the analysis of complex, mechatronical systems we expect this approach to produce more accurate and comprehensive modeling results as these systems often show system dynamic changes.

Future algorithmic developments in this area could be the introduction of sonar-like scanning of a short-term preview of data samples; we plan to implement a measure that compares the models' fitness on the whole current window to their fitness on the preview window. This information shall be used to self-adaptively adjust relevant parameters of the algorithm in order to support genetic diversity, which is essentially important for enhanced adaptation capabilities.

References

Affenzeller M, Winkler S, Wagner S, Beham A (2009) Genetic algorithms and genetic programming: modern concepts and practical applications. Numerical insights. CRC Press, Singapore

Affenzeller M, Winkler SM, Kronberger G, Kommenda M, Burlacu B, Wagner S (2013) Gaining deeper insights in symbolic regression. In: Riolo R, Moore JH, Kotanchek M (eds) Genetic programming theory and practice XI, genetic and evolutionary computation. Springer, Ann Arbor, chap 10, pp 175–190

Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 97–106

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

Poli R (2003) A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E (eds) Genetic programming, Proceedings of EuroGP'2003. Springer-Verlag, Essex, LNCS, vol 2610, pp 204–217

Rasmussen CE, Williams CK (2006) Gaussian processes for machine learning. The MIT Press, Cambridge, Massachusetts

Vladislavleva K, Veeramachaneni K, Burland M, Parcon J, O'Reilly UM (2010) Knowledge mining with genetic programming methods for variable selection in flavor design. In: Branke J, Pelikan M, Alba E, Arnold DV, Bongard J, Brabazon A, Branke J, Butz MV, Clune J, Cohen M, Deb K, Engelbrecht AP, Krasnogor N, Miller JF, O'Neill M, Sastry K, Thierens D, van Hemert J, Vanneschi L, Witt C (eds) GECCO '10: Proceedings of the 12th annual conference on genetic and evolutionary computation, ACM, Portland, Oregon, USA, pp 941–948. doi:10.1145/1830483.1830651

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

Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(2):69–101

Winkler S, Efendic H, Del Re L, Affenzeller M, Wagner S (2007a) Online modelling based on genetic programming. Int J Intell Syst Technol Appl 2(2/3):255–270

Winkler SM, Affenzeller M, Wagner S (2007b) Selection pressure driven sliding window genetic programming. Lecture Notes in Computer Science 4739: Computer Aided Systems Theory - EuroCAST 2007, pp 789–795

Zuo J, Tang Cj, Li C, Yuan Ca, Chen Al (2004) Time series prediction based on gene expression programming. In: Li Q, Wang G, Feng L (eds) Advances in Web-Age Information Management, Lecture Notes in Computer Science, vol 3129. Springer, Berlin, pp 55–64

Stephan M. Winkler received his PhD in engineering sciences in 2008 from Johannes Kepler University (JKU) Linz, Austria. His research interests include genetic programming, nonlinear model identification and machine learning. Since 2009, Dr. Winkler is professor at the Department for Medical and Bioinformatics at the University of Applied Sciences (UAS) Upper Austria at Hagenberg Campus; since 2010, Dr. Winkler is head of the Bioinformatics Research Group at UAS, Hagenberg.

Michael Affenzeller has published several papers, journal articles and books dealing with theoretical and practical aspects of evolutionary computation, genetic algorithms, and meta-heuristics in general. In 2001 he received his PhD in engineering sciences and in 2004 he received his habilitation in applied systems engineering, both from the Johannes Kepler University of Linz, Austria. Michael Affenzeller is professor for heuristic optimization and machine learning at Upper Austria University of applied science (UAS), Campus Hagenberg. He is head of the research group for heuristic and evolutionary algorithms (HEAL) which is responsible for the HeuristicLab framework; since May 2014. Michael Affenzeller is also acting as the overall head of the K-project for heuristic optimization in production and logistics (HOPL).

Gabriel Kronberger received his PhD in engineering sciences in 2010 from JKU Linz. His research interests include genetic programming, machine learning, and data mining and knowledge discovery. Since 2011, Gabriel Kronberger has been a professor for business intelligence and data mining at the Department for Information Engineering and Management at the University of Applied Sciences (UAS) Upper Austria at Hagenberg Campus.

Michael Kommenda finished his studies in bioinformatics at Upper Austria University of Applied Sciences in 2007. Currently he is a research associate at the UAS Research Center Hagenberg working on data-based modeling algorithms for complex systems within the research group HEAL.

Bogdan Burlacu finished his studies in Systems and Computer Engineering at the Technical University of Iasi, Romania in 2009. Currently he is a research associate at the UAS Research Center Hagenberg working on his PhD on analyzing genetic programming behavior within the research group HEAL.

Stefan Wagner received his MSc in computer science in 2004 and his PhD in technical sciences in 2009, both from the Johannes Kepler University Linz, Austria. From 2005 to 2009 he worked as an associate professor for software project engineering and since 2009 as a full professor for complex software systems at the University of Applied Sciences Upper Austria, Campus Hagenberg. Dr. Wagner is one of the founders of the research group Heuristic and Evolutionary Algorithms Laboratory (HEAL) and is the project manager and head developer of the HeuristicLab optimization environment.

Chapter 7
Extremely Accurate Symbolic Regression for Large Feature Problems

Michael F. Korns

Before we begin, let me warn the reader that this chapter is an extension of Korns (2013). *Due to space restrictions, the editors cannot allow both chapters to be amalgamated here. Therefore, one really must read Korns* (2013) *before reading this chapter in order to have a smooth understanding of the concepts herein.*

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 *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 deployed regression package which handles up to 50 to 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 (2013) we published 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. This algorithm is extremely accurate, on a single processor, for

M. F. Korns (✉)
Analytic Research Foundation, 2240 Village Walk Drive Suite 2305, Henderson, Nevada 89052
e-mail: mkorns@korns.com

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_7

up to 25 features (columns); and, a cloud configuration can be used to extend the extreme accuracy up to as many as 100 features.

While the previous algorithm's extreme accuracy for deep problems with a small number of features (25–100) is an impressive advance, there are many very important academic and industrial SR problems requiring from 100 to 1000 features.

In this chapter we extend the previous algorithm such that extreme accuracy is achieved on a wide range of problems, from 25 to 3000 features, using only a single processor. The class of problems, on which the enhanced algorithm is extremely accurate, is described in detail (*mainly up to 3000 features*). A definition of extreme accuracy is provided, and an *informal argument* of high SR accuracy is outlined in this chapter.

A set of representative problems on from 25 to 3000 features is learned by the new enhanced extreme accuracy algorithm. The enhanced algorithm is shown to be robust, performing well even in the face of testing data containing up to 3000 features.

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 $x = <x_0, x_1, x_2, \ldots, x_{M-1}>$: 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 (either a feature *v* or a real number constant *c*), 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 $\xi(a,b) = \xi(a) = a$, 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 = (+ − * / **abs inv cos sin tan tanh sqroot square cube quart exp ln** ξ) (where **inv**(x) = 1.0/x). The terminal set is the features $\mathbf{x}_0$ through $\mathbf{x}_{M-1}$ and the real constant **c**, which we shall consider to be 2^{18} in size.

During the writing of Korns (2010, 2011, 2012, 2013) 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 onscore* and *input* 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* clauses, and the **input 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.

The previous extreme accuracy algorithm (Korns 2013), in a laptop configuration, can manage all problems of *universal(2,1,t)* and *universal(1,3,t)* on training data sets containing up to 25 features. We shall name these $\mathbf{U}_2$**(1)[25]** and in $\mathbf{U}_1$**(3)[25]** respectively. In this chapter our problem is the large number of academic and industrial SR problems requiring far greater feature counts where training data sets containing 100–1000 features are often encountered.

In this chapter we will introduce extensions to the previous extreme accuracy algorithm (Korns 2013), which will extend the previous extreme accuracy performance on 25 features and also extend the extreme accuracy performance to training data sets containing from 150 to 3000 features. The algorithm extensions will be described in detail and an *informal argument* of extreme SR accuracy will be presented.

Obviously a cloud configuration will greatly speed up the enhanced extreme accuracy algorithm, and we will address cloud configurations and extreme accuracy in a later paper. For this chapter, we will develop an extremely accurate SR algorithm which any scientist can use on their personal laptop.

Our core assertion in this chapter is that the enhanced extreme accuracy algorithm will achieve, on a laptop computer, extremely accurate champions for all of the problems in $\mathbf{U}_2$**(1)[25]**, $\mathbf{U}_1$**(25)[25]**, $\mathbf{U}_1$**(5)[150]**, and in $\mathbf{F}_{(x)}$**(5)[3000]** (*note:* $\mathbf{F}_{(x)} = \xi$ *inv abs sqroot square cube quart exp ln cos sin tan tanh*) in reasonable computation times, of a maximum 60 h (*on an advanced laptop built in Jan 2013*) and a maximum 120 h (*on an advanced laptop built in Jan 2008*). Pushing things to the extreme, the enhanced algorithm will achieve extremely accurate champions for all of the problems from $\mathbf{U}_2$**(1)[50]** through $\mathbf{U}_1$**(5)[50]** in a maximum of 240 h (*on an advanced laptop built in Jan 2013*). Most problems finish far quicker than these maximum time horizons.

7.1 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)[25]**, $\mathbf{U}_1$**(25)[25]**, $\mathbf{U}_1$**(5)[150]**, or $\mathbf{F}_{(x)}$**(5)[3000]**, where the function set $\mathrm{F}_{(x)} =$ (**ξ inv abs sqroot square cube quart exp ln cos sin tan tanh**), and the terminal set is the features $\mathbf{x}_0$ thru $\mathbf{x}_{M-1}$ plus the real number constant **c** with **cbit** = 18. Our training data sets will contain 25 features, 150, and 3000 features as specified. Our core assertion is that the enhanced algorithm will find extremely accurate champions for all of these problems and for **all similar problems** in practical time on a laptop computer.

Similar problems are easily obtained by substituting all other possibilities within $\mathbf{U_2(1)[25]}$, $\mathbf{U_1(25)[25]}$, $\mathbf{U_1(5)[150]}$, or $\mathbf{F_{(x)}(5)[3000]}$. For instance one problem in $\mathbf{U_2(1)[25]}$ might be $\mathbf{y = 1.687 + (94.183*(x_3*x_2))}$. By substitution, $\mathbf{y = 1.687 + (94.183*(x_3/x_2))}$ and $\mathbf{y = 1.687 + (94.183*(x_{23}*x_{12}))}$ are also in $\mathbf{U_2(1)[25]}$. Another problem in $\mathbf{U_2(1)[25]}$ might be $\mathbf{y = -2.36 + (28.413*ln(x_2)/x_3)}$. By substitution, $\mathbf{y = -2.36 + (28.413*cos(x_{12})*x_6)}$ and $\mathbf{y = -2.36 + (28.413*sqroot(x_{21})-x_{10})}$ are also in $\mathbf{U_2(1)[25]}$. Our core assertion is that the extreme accuracy algorithm not only finds accurate solutions to the 45 test problems listed below, but also to *all other possible test problems* in $\mathbf{U_2(1)[25]}$, $\mathbf{U_1(25)[25]}$, $\mathbf{U_1(5)[150]}$, or $\mathbf{F_{(x)}(5)[3000]}$.

- **Deep problems in $U_2(1)[25]$**
- *..Note: these problems trained on 10,000 examples with 25 features each*
- (*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*): $y = .913 + (62.13*\ln(x_2)/\text{square}(x_4))$
- **Narrow problems in $U_1(2to3)[25]$**
- *..Note: these problems trained on 10,000 examples with 25 features each*
- (*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*): $y = 36.1 + (3.13*x_2) + (1.13*x_3) + (2.19*x_0)$
- **Wide problems in $U_1(5)[25]$**
- *..Note: these problems trained on 10,000 examples with 25 features each*

- (*T29*): $y = -9.16 + (-9.16*x_{24}*x_0) + (-19.56*x_{20}*x_{21}) + (21.87*x_{24}*x_2) + (-17.48* x_{22}*x_{23}) + (38.81*x_{23}*x_{24})$
- (*T30*): $y = -9.16 + (-9.16*x_{24}/x_0) + (-19.56*x_{20}/x_{21}) + (21.87*x_{24}/x_2) + (-17.48* x_{22}/x_{23}) + (38.81*x_{23}/x_{24})$
- **Broad problems in $F_{(x)}$(5)[3000]**
- *..Note: these problems trained on 5000 examples with 3000 features each*
- *..Note: $F_{(x)}$ = noop inv abs sqroot square cube quart exp ln cos sin tan tanh*
- (*T31*): $y = 50.63 + (63.6*\text{cube}(x_0)) + (66.54*\text{cube}(x_1)) + (32.95*\text{cube}(x_2)) + (4.87* \text{cube}(x_3)) + (46.49*\text{cube}(x_4))$
- (*T32*): $y = -9.16 + (-9.16*\text{square}(x_0)) + (-19.56*\ln(x_{123})) + (21.87*\exp(x_{254})) + (-17.48* x_3) + (38.81*x_{878})$
- (*T33*): $y = 0.0 + (1*\text{square}(x_0)) + (2*\text{square}(x_1)) + (3*\text{square}(x_2)) + (4*\text{square}(x_3)) + (5* \text{square}(x_4))$
- (*T34*): $y = 65.86 + (79.4*\sin(x_0)) + (45.88*\cos(x_1)) + (2.13*\tan(x_2)) + (4.6*\sin(x_3)) + (61.47* \cos(x_4))$
- (*T35*): $y = 1.57 + (1.57/x_{923}) + (-39.34*\sin(x_1)) + (2.13*x_2) + (46.59*\cos(x_{932})) + (11.54*x_4)$
- (*T36*): $y = 50.63 + (63.6*\text{sqroot}(x_0)) + (66.54*\text{sqroot}(x_1)) + (32.95*\text{sqroot}(x_2)) + (4.87*\text{sqroot}\ (x_3)) + (46.49*\text{sqroot}(x_4))$
- (*T37*): $y = 92.25 + (53.53*\text{square}(2.3*x_0)) + (88.26*\cos(x_1)) + (42.11/x_4) + (29.0*\text{cube}(x_3)) + (93.6*\tanh(x_4))$
- **Broad problems in U_1(5)[150]**
- *..Note: these problems trained on 10,000 examples with 150 features each*
- (*T38*): $y = -9.16 + (-9.16*x_{124}*x_0) + (-19.56*x_{120}*x_{21}) + (21.87*x_{24}*x_{26}) + (-17.48*x_{122}*x_{23}) + (38.81*x_{123}*x_{24})$
- (*T39*): $y = -9.16 + (-9.16*x_{124}/x_0) + (-19.56*x_{20}/x_{92}) + (21.87*x_{102}/x_2) + (-17.48*x_{22}/x_{143}) + (38.81*x_{23}/x_{149})$
- (*T40*): $y = -9.16 + (-9.16*\cos(_0)) + (-19.56*x_{20}/x_{21}) + (21.87*\text{square}(x_{125})) + (-17.48*x_{22}/x_{23}) + (38.81*\tanh(x_{24}))$
- **Dense problems in U_1(25)[25]**
- *..Note: these problems trained on 10,000 examples with 25 features each*
- (*T41*): $y = 50.63 + (63.6*\text{cube}(x_0)) + (66.54*\text{square}(x_1)) + (32.95*\text{quart}(x_2)) + (4.87* \text{cube}(x_3)) + (46.49*\text{square}(x_4)) + (62.85*\text{quart}(x_5)) + (90.45*\text{cube}(x_6)) + (63.28*\text{square}(x_7)) + (42.15* \text{quart}(x_8)) + (73.03*\text{cube}(x_9)) + (92.2*\text{square}(x_{10})) + (77.99*\text{quart}(x_{11})) + (56.67*\text{cube}(x_{12})) + (72.51*\text{square}(x_{13})) + (49.77*\text{quart}(x_{14})) + (56.94*\text{cube}(x_{15})) + (54.76*\text{square}(x_{16})) + (23.11* \text{quart}(x_{17})) + (56.03*\text{cube}(x_{18})) + (51.98*\text{square}(x_{19})) + (11.71*\text{quart}(x_{20})) + (33.82*\text{cube}\ (x_{21})) + (46.25*\text{square}(x_{22})) + (32.98*\text{quart}(x_{23})) + (36.06*\text{cube}(x_{24}))$
- (*T42*): $y = -9.16 + (-9.16*x_4*x_0) + (-19.56*x_0*x_1) + (21.87*x_1*x_2) + (-17.48*x_2*x_3) + (38.81* x_3*x_4) + (3.1*x_4*x_5) + (59.81*x_5*x_6) + (93.1*x_6*x_7) + (.81*x_7*x_8) + (9.21*x_8*x_9) + (-5.81*x_9*x_{10}) + (-.01*x_{10}*x_{11}) + (4.21*x_{11}*x_{12}) + (68.81*x_{12}*x_{13}) + (-8.81*x_{13}*x_{14}) + (2.11* x_{14}*x_{15}) + (-7.11*x_{15}*x_{16}) + (-.91*x_{16}*x_{17}) + (20.0*x_{17}*x_{18}) + (1.81*x_{18}*x_{19})$

$+ (9.71*x_{19}* x_{20}) + (8.1*x_{20}*x_{21}) + (6.1*x_{21}*x_{22}) + (18.51*x_{22}*x_{23}) + (7.1*x_{23}*x_{24})$

- (*T43*): $y = 0.0 + (1*\text{square}(x_0)) + (2*\text{square}(x_1)) + (3*\text{square}(x_2)) + (4*\text{square}(x_3)) + (5* \text{square}(x_4)) + (6*\text{square}(x_5)) + (7*\text{square}(x_6)) + (8*\text{square}(x_7)) + (9*\text{square}(x_8)) + (10*\text{square}(x_9)) + (11*\text{square}(x_{10})) + (12*\text{square}(x_{11})) + (13*\text{square}(x_{12})) + (14*\text{square}(x_{13})) + (15* \text{square}(x_{14})) + (16*\text{square}(x_{15})) + (17*\text{square}(x_{16})) + (18*\text{square}(x_{17})) + (19*\text{square}(x_{18})) + (20*\text{square}(x_{19})) + (21*\text{square}(x_{20})) + (22*\text{square}(x_{21})) + (23*\text{square}(x_{22})) + (24*\text{square}(x_{23})) + (25*\text{square}(x_{24}))$
- (*T44*): $y = 65.86 + (79.4*\sin(x_0)) + (45.88*\cos(x_1)) + (2.13*\tan(x_2)) + (4.6*\sin(x_3)) + (61.47* \cos(x_4)) + (30.64*\tan(x_5)) + (51.95*\sin(x_6)) + (47.83*\cos(x_7)) + (4.21*\tan(x_8)) + (37.84*\sin(x_9)) + (62.57*\cos(x_{10})) + (4.68*\tan(x_{11})) + (32.65*\sin(x_{12})) + (86.89*\cos(x_{13})) + (84.79* \tan(x_{14})) + (31.72*\sin(x_{15})) + (90.4*\cos(x_{16})) + (93.57*\tan(x_{17})) + (42.18*\sin(x_{18})) + (47.91* \cos(x_{19})) + (41.48*\tan(x_{20})) + (39.47*\sin(x_{21})) + (48.44*\cos(x_{22})) + (34.75*\tan(x_{23})) + (56.7* \sin(x_{24}))$
- (*T45*): $y = 1.57 + (1.57*x_0) + (-39.34*\sin(x_1)) + (2.13*x_2) + (46.59*(x_3/x_2)) + (11.54*x_4) + (30.64*\ln(x_5)) + (51.95*\text{abs}(x_6)) + (47.83*(x_7*x_3)) + (4.21*\text{quart}(x_8)) + (37.84*x_9) + (62.57* \text{square}(x_{10})) + (4.68*\text{sqroot}(x_{11})) + (32.65*(x_{12}/x_3)) + (86.89*x_{14}) + (84.79*\tan(x_{15})) + (31.72* \text{cube}(x_{16})) + (90.4*(x_{17}*x_{18})) + (93.57*(x_{17}/x_{16})) + (42.18*\sin(x_{18})) + (47.91* \cos(x_{19})) + (41.48*\ln(x_{20})) + (39.47*\text{square}(x_{21})) + (48.44*x_{22}) + (34.75*(x_{23}*x_{24})) + (56.7* x_{24})$

For the sample test problems, we use only statistical best practices out-of-sample testing methodology. A matrix of independent variables is filled with random numbers between -100 and $+100$. Then the model is 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 is filled with random numbers between -100 and $+100$. Then the model is 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 **.0001** or less on the **noiseless testing data**. In the table of results, at the conclusion of this chapter, the noisless test results are listed under the **Test-NLSE** column header.

All timings quoted in this chapter were performed on a Dell XPS L521X Intel i7 quad core laptop with 16 Gig of RAM, and 1 Tb of hard drive, manufactured in Dec 2012 (*our test machine*).

7.2 Large Feature Regression

The currently available techniques for attacking large feature regression problems include *stepwise regression* (Draper and Smith 1981), *ridge regression* (Hoerl 1962), *LASSO* (Tibshirani 1996), and *elastic nets* (McConaghy 2011) among others.

In statistics, *stepwise regression* refers to a class of regression algorithms in which the choice of predictive variables are incrementally selected via some automated process. Often F-Tests, t-Test, R-Square, Bayesian, and a host of other heuristic decision metrics are utilized. Ridge regression, LASSO, and elastic nets are similar iterative techniques in which the heuristic involves varying one or more scalar multipliers in the general regression equation itself.

For instance, suppose we have a large number of features $= |V|$, and we wish to choose a maximum of **B** basis functions (*where* $B <= |V|$) as the largest regression model that we are willing to accept. If $B_f(V)$ represents our choices for each basis function on V, then $|\mathrm{B}_f(\mathrm{V})|^B$ represents the total number of choices required for exhaustive search. As the set $\mathrm{B}_f(\mathrm{V})$ becomes more interesting, the number $|\mathrm{B}_f(\mathrm{V})|^B$ often becomes so large that exhaustive search is impractical.

Note that with modern sophisticated multiple regression, selecting a maximum of **B** *basis functions will automatically include all models of less than* **B** *basis functions which are the most accurate, by assigning zero coefficients to the basis functions which do NOT improve accuracy.*

Therefore, in a highly abstracted idealized stepwise regression algorithm, one would first search $\mathrm{B}_f(\mathrm{V})$ exhaustively with simple regressions, regress($\mathrm{B}_{fi}(\mathrm{V})$), gathering all necessary statistics for each individual simple regression. From those experiences, a single basis function would be selected for position 1, $\mathrm{B}_{f1}(\mathrm{V})$. Then one would next search $\mathrm{B}_f(\mathrm{V})$ exhaustively with multiple regressions, regress(B_{f1} (V),$\mathrm{B}_{fi}(\mathrm{V})$), gathering all necessary statistics for each individual multiple regression. From those experiences, a single basis function would be selected for position 2, $\mathrm{B}_{f2}(\mathrm{V})$. This idealized process would be repeated until B basis functions had been selected, or *until selecting additional basis functions failed to improve accuracy*.

All available current large feature regression techniques operate in a variation of this idealized iterative algorithm. There is great latitude in the statistics and heurisms used to select optimal basis functions, and there is some latitude in whether or not and how much backtracking can be utilized. None of the available large feature regression techniques performs an exhaustive search on all $|\mathrm{B}_f(\mathrm{V})|^B$ choices. As one would expect, each available large feature regression technique has its own advantages and disadvantages.

Ridge regression is a very fast multiple regression technique with a constant multiplier of the identity matrix which gives a non-zero regression coefficient to every basis function in the regression model. This will offer extreme accuracy, which is robust in the face of noisy training data, but not in the face of range shifting. Furthermore, ridge regression works properly with only a small number of features in the training data and has a very difficult time dealing with thousands of features in the training data.

Stepwise regression, based on F-Test statistics, is an incremental, greedy regression technique which works well with a large number of features in the training data, but which does not offer any pretense of extreme accuracy. Similarly, LASSO and elastic nets are also incremental greedy regression techniques which work well with a large number of features in the training data, often better than stepwise regression, but they also do not offer any pretense of extreme accuracy.

Unfortunately, none of the available statistical regression techniques promise extreme accuracy on large feature regression problems.

Since none of the currently available regression techniques offers extreme accuracy on large numbers of features, we will have to break new ground in this chapter. Our approach will be to extend the previous extreme accuracy algorithm (Korns 2013) with a new technique which merges the lessons learned in evolutionary programming with the lessons learned in stepwise regression, ridge regression, LASSO, and elastic nets.

The previous extreme accuracy algorithm (Korns 2013), consists of 25 separate search islands, *S0* through *S24*. Each island is an independent RQL search island capable of being run on a separate computing device. The best regression champion in any of the islands is automatically considered the answer to the extreme accuracy regression problem.

The search island *S0* is a general pareto front search island which conveys the power and capabilities of current state-of-the-art pareto front SR to the extreme accuracy algorithm. Search island *S1* provides extreme accuracy for problems in $U_1(3)[25]$ (*on a laptop—no cloud*). Search islands *S2* through *S24* jointly provide extreme accuracy for problems in $U_2(1)[25]$ (*on a laptop—no cloud*).

In this chapter we will leave search islands *S2* through *S24* unaltered (with the exception that we have added the unary functions **inv** and **abs** everywhere required in those search islands). We will enhance search islands *S0* and *S1* to provide extreme accuracy for problems in $U_2(5)[25]$, $U_1(25)[25]$, $U_1(5)[150]$, up to and including $F_{(x)}(5)[3000]$ (*on a laptop—no cloud*).

The previous extreme accuracy RQL (S0) search command is fairly straightforward and provides Pareto-front-like accuracy for all problems up through $U_D(B)[25]$ (*on a laptop—no cloud*).

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

The previous extreme accuracy RQL (S1) search command is fairly straightforward and provides extreme accuracy for all problems in $U_1(3)[25]$ (*on a laptop only—a cloud implementation would allow faster completion and a larger maximum number of features*).

- (*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)

If we wish to provide extreme accuracy beyond $U_2(1)[25]$, $U_1(3)[150]$ through $U_1(3)[25]$, we will have to enhance these two RQL search commands.

7.3 Search Island S0.0 to S0.B

General pareto front search is a powerful state-of-the-art approach to symbolic regression. Many of the current leading contender algorithms in SR use pareto front evolution in one form or another. If we wish to extend the power and reach of our extreme accuracy algorithm into $U_1(5)$[25] and beyond, we will have to enhance the algorithm with a more formidable pareto front treatment.

The enhancements to search island (S0) expand the number of pareto front islands based on the number of basis functions requested, yielding new pareto front search islands from (S0.1) through (S0.B) (*where B is the number of basis functions requested*). Assuming that B = 5, the new enhanced pareto search islands are as follows.

- (*S0.1*) **search** regress(universal(5,1,t)) **where** island(pareto,standard,256,25,00, 50,10) op(ξ,inv,abs,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,+,−,*,/)
- (*S0.2*) **search** regress(universal(5,2,t)) **where** island(pareto,standard,256,25,00, 50,10) op(ξ,inv,abs,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,+,−,*,/) reduce(true) onfinal('regress(*poly*)')
- (*S0.3*) **search** regress(universal(5,3,t)) **where** island(pareto,standard,256,25,00, 50,10) op(ξ,inv,abs,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,+,−,*,/) reduce(true) onfinal('regress(*poly*)')
- (*S0.4*) **search** regress(universal(5,4,t)) **where** island(pareto,standard,256,25,00, 50,10) op(ξ,inv,abs,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,+,−,*,/) reduce(true) onfinal('regress(*poly*)')
- (*S0.5*) **search** regress(universal(5,5,t)) **where** island(pareto,standard,256,25,00, 50,10) op(ξ,inv,abs,cos,sin,tan,tanh,sqroot,square,cube,quart,exp,ln,+,−,*,/) reduce(true) onfinal('regress(*poly*)')

The *reduce(true)* and *onfinal('regress(poly)')* clauses cause the search island, upon completion, to try each individual basis function and subgroup of basis functions in the reigning champion to see if some are not needed.

For instance if the correct answer was $y = 45.6 + (23.2*x_{102})$ and the reigning champion, upon completion, was $y = 49.1 + (.003*x_{23}) + (25.9*x_{102})$ then the *reduce(true)* and *onfinal('regress(poly)')* clauses will cause the search island to discover that $y = 45.6 + (23.2*x_{102})$ is a better answer.

Taken all together search islands (S0.1) through (S0.B) constitute a concerted pareto-front attack on all problems from $U_5(1)$[|V|], $U_5(2)$[|V|], ... $U_5(B)$[|V|] (*where |V| is the number of features in the problem and B is the number of basis functions*). The new pareto front islands consume about one third of the computational resources of the new enhanced extreme accuracy algorithm. The results are an amazingly powerful incursion into this vast problem space. Many of the test problems are solved with these pareto front search islands alone.

Unfortunately, although powerful, the accuracy is hit and miss. There are a surprisingly high number of deep problems solved by these pareto front islands; but, from time to time, a problem is missed. There is no informal accuracy argument and no way of predicting which problems will be missed. Changing the random number seed only causes the set of missed problems to move mysteriously. Furthermore, as |V| approaches 25, 150, and 3000, the percent of missed problems grows larger.

If we wish to have extreme accuracy from $U_2(1)$[25], $U_1(5)$[150] through $F_x(5)$ [3000], we will have to supplement the (S0.i) search islands with the other extreme accuracy enhancements.

7.4 Baseline Accuracy Measurements

Packaging together RQL search commands ONLY from search islands (S0.0) thru (S0.B), implemented with the baseline algorithm in Korns (2012), we now have a way of measuring the baseline accuracy performance of relatively straightforward pareto-GP symbolic regression on the 45 test problems.

As mentioned, each of the problems were trained and tested on from 25 to 3000 features as specified using out of sample testing. The allocated maximum generations were set such that the theoretical maximum time to complete a test problem on our laptop environment was 60 h, at which time training was automatically halted and the best champion was returned as the answer. However, most problems finished well ahead of that maximum time limit.

All timings quoted in this table were performed on a Dell XPS L521X Intel i7 quad core laptop with 16 Gig of RAM, and 1 Tb of hard drive, manufactured in Dec 2012 (*our test machine*)[1].

The results in Table 7.1 demonstrate only intermittent accuracy on the 45 test problems. Baseline accuracy is very good with 1, 2, or 5 features in the training data. Unfortunately, Baseline accuracy decreases rapidly as the number of features in the training data increases to 25, 150, and 3000. Furthermore, there is a great deal of overfitting as evidenced by the number of test cases with good training scores and very poor testing scores.

In such cases of overfitting, SR becomes deceptive. It produces tantalizing candidates which, from their training NLSE scores, look really exciting. Unfortunately, they fail miserably on the testing data.

Clearly the baseline testing results in Table 7.1 demonstrate an opportunity for improved accuracy.

Another serious issue with the baseline algorithm is that negative results have no explicit meaning. For example, Alice runs the baseline algorithm on a large block of

[1] Testing a single regression champion is not cheap. At a minimum testing a single regression champion requires as many evaluations as there are training examples as well as performing a simple regression. At a maximum testing a single regression champion may require performing a much more expensive multiple regression.

Table 7.1 Results demonstrating baseline accuracy

Test	WFFs[a]	Train-Hrs[b]	Train-NLSE[c]	Test-NLSE[d]
T01	1K	0.01	0.0000	0.0000
T02	9K	0.08	0.0000	0.0000
T03	87K	1.00	0.0000	0.0000
T04	11K	0.02	0.0000	0.0000
T05	812K	9.00	0.0000	0.0000
T06	1246K	13.86	0.5364	0.7727
T07	112K	1.29	0.0000	0.0000
T08	1221K	14.40	0.0034	0.1354
T09	1240K	25.86	0.0484	0.9999
T10	1242K	13.97	0.0185	0.9999
T11	817K	10.26	0.0317	0.9999
T12	914K	11.46	0.0244	0.9999
T13	5K	0.05	0.0000	0.0000
T14	9K	0.09	0.0000	0.0000
T15	724K	10.27	0.8540	0.9348
T16	884K	10.66	0.0077	0.9999
T17	10K	0.10	0.0000	0.0000
T18	360K	4.51	0.0000	0.0000
T19	73K	0.86	0.0000	0.0000
T20	356K	4.41	0.0000	0.0000
T21	908K	10.94	0.0560	0.0222
T22	908K	11.05	0.0568	0.0602
T23	621K	8.21	0.0000	0.9999
T24	5K	0.05	0.0000	0.0000
T25	77K	0.88	0.0000	0.0000
T26	17K	0.18	0.0000	0.0000
T27	79K	0.85	0.0000	0.0000
T28	10K	0.10	0.0000	0.0000
T29	870K	10.11	0.1324	0.1334
T30	900K	11.48	0.0290	0.0099
T31	900K	11.48	0.2104	0.2289
T32	179K	8.06	0.0000	0.0000
T33	280K	13.82	0.2435	0.2398
T34	283K	15.44	0.2028	0.2412
T35	251K	13.49	0.0511	0.0540

Table 7.1 (continued)

Test	WFFs[a]	Train-Hrs[b]	Train-NLSE[c]	Test-NLSE[d]
T36	333K	13.03	0.4524	0.4755
T37	255K	11.97	0.0000	0.0000
T38	275K	3.45	0.7453	0.8026
T39	282K	3.84	0.0403	0.9999
T40	249K	3.14	0.0022	0.9999
T41	854K	26.46	0.0455	0.0645
T42	978K	23.84	0.8415	0.9999
T43	507K	17.46	0.3838	0.8082
T44	517K	17.81	0.0062	0.9999
T45	517K	17.55	0.0024	0.9999

[a] The number of regression candidates tested before finding a solution is listed in the Well Formed Formulas (WFFs) column
[b] The elapsed hours spent training on the noiseless training data is listed in the (Train-Hrs) column
[c] The fitness score of the champion on the noiseless training data is listed in the (Train-NLSE) column
[d] The fitness score of the champion on the noiseless testing data is listed in the (Test-NLSE) column. Note also search commands (S1) thru (S24) are **not** *included* in the baseline algorithm

data for the maximum time specified. At the conclusion of the maximum specified generations, requiring a maximum of 60 h on our laptop, no candidate with a zero NLSE (*perfect score*) is returned. The meaning of this negative result is indeterminate, as one can argue that perhaps if Alice were to run the baseline algorithm for *a few more generations* an exact candidate would be discovered.

This chapter is devoted to enhancing the extreme accuracy algorithm in Korns (2013) by enhancing search commands (S0) and (S1), then combining the enhanced search commands (S0) and (S1) with the previous search commands (S2) thru (S24) together to form the new enhanced extreme accuracy algorithm. At the close of this chapter, the enhanced extreme accuracy algorithm will be measured against the 45 test problems, and we will see what, if any, accuracy improvements have been achieved over the baseline algorithm. *Note: The baseline algorithm is enhanced search command (S0) ONLY as implemented by the baseline algorithm in Korns* (2012).

7.5 Search Island S1.0

The enhancements to search island (S1) merge lessons learned from the statistical *stepwise* regression techniques together with the lesson learned in evolutionary symbolic regression. The goal is to achieve extreme accuracy in problems from $U_1(5)[150]$ through $F_x(5)[3000]$.

All enhanced search islands *(S1.0)* through *(S1.B)*, taken together, form a kind of *stepwise* regression algorithm, but one in which evolutionary pressures take the place of F-Tests, Bayesian, or other heuristic methods of selecting basis functions. The (S1.0) RQL search begins this stepwise process with a straightforward elitist attempt to find **B** basis functions which minimize the NLSE for the specified problem.

- (*S1.0*) **search** regress(universal(1,B,v)) **where** island(smart,standard,10,0, 400,200) op(ξ,inv,abs,sqroot,square,cube,quart,exp,ln,cos,sin,tan,tanh,*,/,+,−) reduce(true)
  onfinal('regress(*poly*)')
  name(Wide)

Search command (S1.0) performs multiple regressions with **B** basis functions, each of which is in U_1 and looks like f(t,t). Each of the variables v_i contain a single features such as x_j. Each of f_k are function variables containing single 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.

- (*E1*) regress(c_0+v_0) = a+b*(c_0+v_0) = a+(b*c_0)+b*v_0 = regress(v_0)
- (*E2*) regress(c_0/v_0) = a+b*(c_0/v_0) = a+(b*c_0)/v_0 = regress(inv(v_0))
- (*E3*) 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(universal(1,B,v))* as our search goal.

The *reduce(true)* and *onfinal('regress(poly)')* clauses cause the search island, upon completion, to try each individual basis function and subgroup of basis functions in the reigning champion to see if some are not needed.

For instance if the correct answer was $y = 45.6 + (23.2*x_{102})$ and the reigning champion, upon completion, was $y = 49.1 + (.003*x_{23}) + (25.9*x_{102})$ then the *reduce(true)* and *onfinal('regress(poly)')* clauses will cause the search island to discover that $y = 45.6 + (23.2*x_{102})$ is a better answer.

The *name(Wide)* clause gives the search island a group name, which will be used in searches (S1.1) through (S1.B) later in this chapter.

Search island (S1.0) starts the stepwise algorithm off with a rapid start due to the power of evolutionary pressure and the linear nature of the problem. Very quickly search island (S1.0) will arrive at an *approximately accurate* answer even for very difficult problems, and as evolution continues the reigning champion in search island (S1.0) will get approximately better and better.

Unfortunately, search island (S1.0) will often fail to arrive at an *extremely accurate* answer. When (S1.0) first starts out, it has many choices for basis functions, $B_{fi}(V)$, which will improve accuracy. However, if the problem is difficult, eventually (S1.0) will arrive at a champion $y = regress(...,B_{fi(x)}(V),...)$ where the ONLY mutation, which will improve accuracy, is to change $y = regress(...,B_{fi(x)}(V),...)$ to $y = regress(...,B_{fk(x)}(V),...)$. Now the probability of improvement is $1/(|B_{f(x)}(V)|^B)$ which is often too small to reach in practical time.

Therefore, if we wish to achieve *extreme accuracy* we will have to add additional iterative steps in addition to search island (S1.0). These additional iterative steps are the search islands *(S1.1)* through *(S1.B)* which are covered in the next section.

7.6 Search Islands S1.1 to S1.B

The purpose of search islands (S1.1) through (S1.B) is to provide stepwise *column focused* serial search on each of the B columns in all cases where the main (S1.0) evolutionary island has achieved a new best champion. This process starts with a main search island (S1.0) which searches all columns for all possible basis function in an elitist evolutionary fashion, and we have B stepwise search islands which search each of the B columns separately with elitist evolutionary and serial attacks. This iterative search is performed each time any of these islands discovers a new best champion.

The number of additional RQL search commands (S1.1) through (S1.B) are determined by the number of requested basis functions, **B**. For the purposes of illustration, let us assume that $\mathbf{B} = 3$. Then the additional search islands (S1.1) through (S1.3) are as follows.

- (*S1.1*) **search** regress(universal(1,1,v),x0,x0) **where** island(smart,standard,10,0, 400,200) op(ξ,inv,abs,sqroot,square,cube,quart,exp,ln,cos,sin,tan,tanh,*,/,+,−) input(Wide,'regress(universal(1,1,v),$B1$,$B2$)') delay(15) name(Wide) reduce(true) onfinal('regress(*poly*)')
- (*S1.2*) **search** regress(x0,universal(1,1,v),x0) **where** island(smart,standard,10,0, 400,200) op(ξ,inv,abs,sqroot,square,cube,quart,exp,ln,cos,sin,tan,tanh,*,/,+,−) input(Wide,'regress($B0$,universal(1,1,v),$B2$)') delay(15) name(Wide) reduce(true) onfinal('regress(*poly*)')
- (*S1.3*) **search** regress(x0,x0,universal(1,1,v)) **where** island(smart,standard,10,0, 400,200) op(ξ,inv,abs,sqroot,square,cube,quart,exp,ln,cos,sin,tan,tanh,*,/,+,−) input(Wide,'regress($B0$,$B1$,universal(1,1,v))') delay(15) name(Wide) reduce(true) onfinal('regress(*poly*)')

In search islands (S1.1) through (S1.B) the *delay(15)* clause forces these islands to start evolution only after 15 generations. This gives search island (S1.0) a chance to settle down before stepwise iterations are attempted. This parameter is NOT critical and can be set to any reasonable value without serious effect on the algorithm.

Also it is important to note that island (S1.0) searches on all columns for all possible basis functions. However, search island (S1.1) through (S1.B) each search on *only a single* but different column. For instance, island (S1.1) will perform a concentrated elitist evolutionary and serial search on the 1st basis function only, while island (S1.3) will perform a concentrated elitist evolutionary and serial search on the 3rd basis function only. In addition, islands (S1.0) through (S1.B) are all linked together in that they all share the same group name: **Wide**. This allows them all to perform *iterative stepwise* evolutionary searches like dancers whose steps are coordinated. Here's how the stepwise search coordination works.

It is the **input** clause for each island, (S1.1) through (S1.B), which determines the coordination of the subsequent iterative stepwise evolution. The input clause tells its island to reset and restart the evolutionary process whenever ANY island sharing the name *Wide* achieves a new local best fitness champion. Once triggered by *any* local fitness improvement, the specified search island resets its search goal by replacing its various basis functions, B_{fi}, with the appropriate basis functions from the local champion. These are substituted for the specified wild card expressions shown in each input clause, *regress($B0$,$B1$,universal(1,1,v))*. Once reset, each search island resumes the evolutionary process starting from its newly substituted search goal. Note that the new search goal has most of its basis functions fixed (*they do not evolve*) and only one of its basis functions available for evolutionary and serial search.

One other very important point of note is, in addition to the 200 evolutionary operations per generation, each of islands (S1.1) through (S1.B) also perform 400 serial iterations per generation. It is these serial iterations which are central to the informal argument for extreme accuracy.

Let's follow this iterative stepwise process with a small example as follows.

Assume that the exact solution to our example problem is:

- (*E4*) $y = \text{regress}(\text{square}(x_{21}),x_1/x_{11},x_9)$

In the beginning search island (S1.0) performs repeated evolutionary operations achieving increasingly fit local champions such as:

- (*E5*) $\text{regress}(\text{cube}(x_{21}),x_1/x_{11},x_{12})$

At that point, in our simplified example, the delay on search islands (S1.1) through (S1.3) expires, and these three search islands begin iteratively evolving using the following substituted goals.

- (*E6*) $\text{regress}(\text{universal}(1,1,v),x_1/x_{11},x_{12})$
- (*E7*) $\text{regress}(\text{cube}(x_{21}),\text{universal}(1,1,v),x_{12})$
- (*E8*) $\text{regress}(\text{cube}(x_{21}),x_1/x_{11},\text{universal}(1,1,v))$

After reset, search islands (S1.1) through (S1.3) plus search island (S1.0), *which has never been reset*, continue iteratively evolving until ANY one of these islands achieves a new local best fit champion such as:

- (*E9*) $\text{regress}(\text{cube}(x_{21}),x_1/x_{11},x_9)$

At that point, search islands (S1.1) through (S1.3) are triggered to reset again, and these three search islands begin iteratively evolving using the following newly substituted goals.

- (*E10*) regress(universal(1,1,v),x_1/x_{11},x_9)
- (*E11*) regress(cube(x_{21}),universal(1,1,v),x_9)
- (*E12*) regress(cube(x_{21}),x_1/x_{11},universal(1,1,v))

After reset, search islands (S1.1) through (S1.3) plus search island (S1.0), *which has never been reset*, continue iteratively evolving until ANY one of these islands achieves a new local best fit champion such as.

- (*E13*) regress(square(x_{21}),x_1/x_{11},x_9)

And, of course, (E13) is the answer to our simplified problem. Therefore, the informal argument for extreme accuracy is asserted as follows.

At the start, search island (S1.0) will perform an elitist evolutionary search discovering increasingly fit champions. Either search (S1.0) will find the global solution, or it will run into trouble where the probability of finding an improved champion is, $N/(|B_f(V)|^B)$, for some small $N > 0$. Let's assume the worst case situation where, $N = 1$, and the local champion is $y = regress(...,B_{fi(x)}(V),...)$ where the ONLY mutation, which will improve accuracy, is to change $y = regress(...,B_{fi(x)}(V),...)$ to $y = regress(...,B_{fk(x)}(V),...)$.

At that point, the search islands (S1.0) through (S1.B) will start evolving more focused solutions. As mentioned, now the probability of improvement is $1/(|B_{f(x)}(V)|^B)$ which is often too small to reach in practical time. However, one of the search islands (S1.1) through (S1.B) will be searching just that one column and it will need to search iteratively through only $|B_{f(x)}(V)|$ possible basis functions which is much easier. Eventually that particular search island will discover $y = regress(...,B_{fk(x)}(V),...)$ serially, and the answer will form a new global best fit champion.

Let's explore the basic math to see why search islands (S1.1) through (S1.B) are so helpful. In solving $U_1(5)[150]$, there are $(150^2 * 4) + (13 * 150) = 91950^5 = 6.57e10^{24}$ possibilities which, are far too many to be arrived at serially and often evolutionary methods are not able to find an exact answer in practical time.

However, each of search islands (S1.1) through (S1.B) perform a multiple regression wherein all of the columns save one are fixed and do not evolve. Only the ith basis function evolves in each of (S1.i) i.e. $y = regress(...,universal(1,1,v),...)$. In solving $U_1(5)[150]$, islands (S1.1) through (S1.B) only need to address there are $(150^2 * 4) + (13 * 150) = 91950$ possibilities which, at 400 iterations per generation, can be serially covered in just 230 generations. Evolutionary pressure, coupled with the linear nature of the problem, normally arrives at an extremely accurate answer in less than 10–20 stepwise cycles which is far less than the maximum time required.

Similarly in solving $F_{(x)}(5)[3000]$, there are $(13 * 3000) = 39000^5 = 9.0e10^{22}$ possibilities which, are far too many to be arrived at serially or with evolution in practical time. However, each of search islands (S1.1) through (S1.B) only need to address there are $(13 * 3000) = 39000$ possibilities which, at 400 iterations per generation, can be serially covered in just 98 generations. Evolutionary pressure,

coupled with the linear nature of the problem, normally arrives at an extremely accurate answer in less than 10–20 stepwise cycles which is far less than the maximum time required.

Most often the evolutionary search finds the correct answer in far less time.

7.7 Accuracy Measurements

Packaging together the previous extreme accuracy algorithm (Korns 2013) RQL search commands (S2) thru (S24), together with search island *S0* expanded to new enhanced search islands (S0.1 thru S0.B) and together with search island *S1* expanded to new enhanced search islands (S1.0 thru S1.B), we create the new enhanced extreme accuracy algorithm. We apply the new enhanced extreme accuracy algorithm to the 45 test problems using our test machine. As mentioned, each of the problems were trained and tested on from 25 to 3000 features as specified. The theoretical maximum time to complete a test problem on our laptop environment is 60 h, at which time training will automatically halt and the best champion will be returned as the answer. However, most problems finish well ahead of that maximum time limit.

Significantly, the results in Table 7.2 demonstrate extreme accuracy on the 45 test problems. This extreme accuracy is robust even in the face of problems with large number of features.

All timings quoted in this table were performed on a Dell XPS L521X Intel i7 quad core laptop with 16 Gig of RAM, and 1 Tb of hard drive, manufactured in Dec 2012 (*our test machine*).

Notice the extreme search efficiency which Table 7.2 demonstrates. Our assertion is that the extreme accuracy algorithm is getting the same accuracy on $U_2(1)[25]$, $U_1(25)[25]$, $U_1(5)[150]$, and $F_{(x)}(5)[3000]$ as if each and every single element of those sets were searched serially; and yet we are never testing more than a few million regression 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[2].

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 **.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 $y = 1.0 +$

[2] As a reminder, testing a single regression champion is not cheap. At a minimum testing a single regression champion requires as many evaluations as there are training examples as well as performing a simple regression. At a maximum testing a single regression champion may require performing a much more expensive multiple regression.

Table 7.2 Results demonstrating extreme accuracy

Test	*WFFs*[a]	*Train-Hrs*[b]	*Train-NLSE*[c]	*Test-NLSE*[d]
T01	1K	0.01	0.0000	0.0000
T02	1K	0.01	0.0000	0.0000
T03	1K	0.01	0.0000	0.0000
T04	3K	0.02	0.0000	0.0000
T05	176K	0.43	0.0000	0.0000
T06	424K	0.72	0.0000	0.0000
T07	175K	0.39	0.0000	0.0000
T08	427K	0.68	0.0000	0.0000
T09	5961K	4.84	0.0000	0.0000
T10	3864K	3.19	0.0000	0.0000
T11	529K	0.85	0.0000	0.0000
T12	228K	0.50	0.0000	0.0000
T13	1K	0.01	0.0000	0.0000
T14	688K	1.05	0.0000	0.0000
T15	1423K	1.78	0.0000	0.0000
T16	2049K	2.09	0.0000	0.0000
T17	5771K	4.30	0.0000	0.0000
T18	7249K	5.31	0.0000	0.0000
T19	675K	1.02	0.0000	0.0000
T20	571K	0.89	0.0000	0.0000
T21	14366K	13.03	0.0000	0.0000
T22	18856K	20.29	0.0000	0.0000
T23	1935K	2.46	0.0000	0.0000
T24	1K	0.01	0.0000	0.0000
T25	1K	0.01	0.0000	0.0000
T26	12K	0.11	0.0000	0.0000
T27	11K	0.10	0.0000	0.0000
T28	1K	0.01	0.0000	0.0000
T29	40K	0.22	0.0000	0.0000
T30	27K	0.22	0.0000	0.0000
T31	167K	3.53	0.0000	0.0000
T32	48K	1.00	0.0000	0.0000
T33	73K	1.61	0.0000	0.0000
T34	982K	11.64	0.0000	0.0000
T35	408K	5.69	0.0000	0.0000

Table 7.2 (continued)

Test	*WFFs*[a]	*Train-Hrs*[b]	*Train-NLSE*[c]	*Test-NLSE*[d]
T36	255K	4.41	0.0000	0.0000
T37	570K	7.56	0.0000	0.0000
T38	669K	5.26	0.0000	0.0000
T39	192K	1.78	0.0000	0.0000
T40	276K	2.31	0.0000	0.0000
T41	24K	0.38	0.0000	0.0000
T42	218K	3.84	0.0000	0.0000
T43	10K	0.16	0.0000	0.0000
T44	115K	1.81	0.0000	0.0000
T45	150K	2.62	0.0000	0.0000

[a] The number of regression candidates tested before finding a solution is listed in the Well Formed Formulas (WFFs) column
[b] The elapsed hours spent training on the noiseless training data is listed in the (Train-Hrs) column
[c] The fitness score of the champion on the noiseless training data is listed in the (Train-NLSE) column
[d] The fitness score of the champion on the noiseless testing data is listed in the (Test-NLSE) column

*(100.0*sin(x_0)) + (.001*square(x_0))* we notice that the final term *(.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 −1000 to 1000 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.

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—nor is it necessary or desirable that the end user have such involvement.

Another very important benefit of extreme accuracy will only be fully realized when all undiscovered errors are worked out of our *informal argument for extreme accuracy* and when our informal argument is crafted into a complete, peer reviewed, well accepted, formal mathematical proof of accuracy. Once this goal is achieved, we can begin to make **modus tollens** arguments from negative results!

For example, our future Alice runs the extreme accuracy algorithm on a large block of data for the maximum time specified. At the conclusion of the maximum specified generations, requiring a maximum of 60 h on our laptop, no candidate with a zero NLSE (*perfect score*) is returned. Referring to the published, well accepted formal mathematical proof of accuracy, Alice argues (*modus tollens*) that there exists no exact relationship between X and Y anywhere within $U_2(1)[25]$, $U_1(25)[25]$, and $U_1(5)[150]$ through $F_x(5)[3000]$.

7.8 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 Korns (2013), an extreme accuracy algorithm was introduced with an informal argument asserting extreme accuracy in a number of problems. 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.

In this paper we enhance that algorithm to achieve a level of extreme accuracy on problems with a large number of features.

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.

Future research must explore the extreme accuracy algorithm's robustness in the face of noisy training data and range shifted training data. In addition to understanding the response surface of our SR tools with noiseless data, we need to understand the behavior of our SR tools with noisy data and with range shifted data or both. Furthermore it would be useful to know what types of noise effect our SR tools in what ways.

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.

Truly knowing the strengths and weaknesses of our tools is an essential step in gaining trust in their use.

References

Draper NR, Smith H (1981) Applied regression analysis, 2nd edn. Wiley, New York

Hoerl A (1962) Application of ridge analysis to regression problems. Chem Eng Prog 58:54–59

Korns MF (2010) Abstract expression grammar symbolic regression. In: Riolo R, McConaghy T, Vladislavleva E (eds) Genetic programming theory and practice VIII, genetic and evolutionary computation, vol 8. Springer, Ann Arbor, 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, genetic and evolutionary computation. Springer, Ann Arbor, chap 8, pp 129–151. doi:10.1007/978-1-4614-1770-5-8

Korns MF (2012) A baseline symbolic regression algorithm. In: Riolo R, Vladislavleva E, Ritchie MD, Moore JH (eds) Genetic programming theory and practice X, genetic and evolutionary computation. Springer, Ann Arbor, chap 9, pp 117–137.doi:10.1007/978-1-4614-6846-2-9, URL http://dx.doi.org/10.1007/978-1-4614-6846-2-9

Korns MF (2013) Extreme accuracy in symbolic regression. In: Riolo R, Moore JH, Kotanchek M (eds) Genetic programming theory and practice XI, genetic and evolutionary computation. Springer, Ann Arbor, chap 1, pp 1–30. doi:10.1007/978-1-4939-0375-7-1

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, genetic and evolutionary computation. Springer, Ann Arbor, 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. http://mitpress.mit.edu/books/genetic-programming

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, http://trent.st/content/2011-GPTP-FFX-paper.pdf

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

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

Tibshirani R (1996) Regression shrinkage and selection via the lasso. J Royal Stat Soc Ser B (Methodological) 58:267–288

Michael F. Korns is Chief Technology Officer at Freeman Investment Management, Henderson, Nevada, USA.

Chapter 8
How to Exploit Alignment in the Error Space: Two Different GP Models

Mauro Castelli, Leonardo Vanneschi, Sara Silva and Stefano Ruberto

8.1 Introduction

The use of semantic awareness for improving Genetic Programming (GP) (Koza 1992; Poli et al. 2008) is nowadays a well-established reality. The term semantics has been used with several different meanings so far, but one of the most recent and well accepted definitions of semantics is *the vector of the output values of an individual/program on all training cases*. This definition makes sense because this vector encodes the input-output behavior of a program (i.e. its operational semantics) on that part of the problem that is known (the training set). A survey discussing most of the existing approaches that use this definition of semantics can be found in Vanneschi et al. (2014). In particular, in the last few years, semantic awareness has been used with success to improve GP for symbolic regression applications. Our team has contributed to the field with several publications. In particular, in Ruberto et al. (2014), we introduced an idea bound to semantics, that can be represented by the schema in Fig. 8.1 and sketched by the following points:

- With the term *genotypic space*, we indicate the space of the genotypes of all the individuals in the search space. We have represented genotypes as trees in Fig. 8.1, but our reasoning holds for any other alternative GP representation.
- As already said, the semantics of an individual is the vector of its output values on the training cases. It can be represented as a point in a space that we call

M. Castelli (✉) · L. Vanneschi
NOVA IMS, Universidade Nova de Lisboa, 1070-312 Lisboa, Portugal
e-mail: mcastelli@novaims.unl.pt

L. Vanneschi
e-mail: lvanneschi@novaims.unl.pt

S. Silva
BioISI, FCUL, University of Lisbon, 1749-016 Lisbon, Portugal
e-mail: sara@fc.ul.pt

S. Ruberto
GSSI, Gran Sasso Science Institute, INFN, 67100 L'Aquila, Italy

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_8

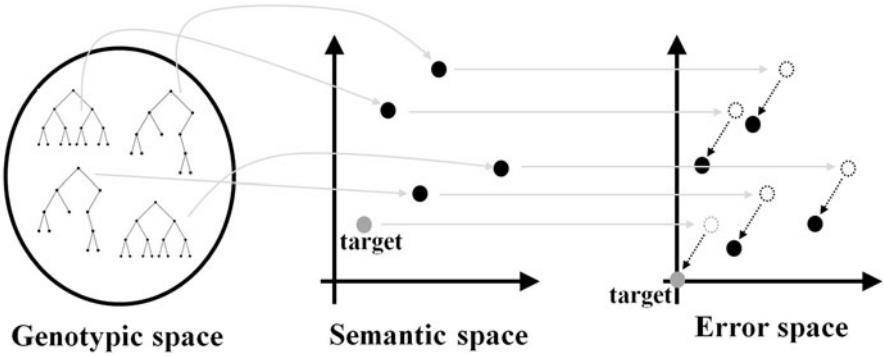


Fig. 8.1 Individuals are represented by trees (or any other structure like linear genomes, graphs, etc.) in the genotypic space. Each one of them maps into its semantic, identified by a point in the semantic space. The semantic is then translated by subtracting the target, obtaining a point in the error space. The target, which usually does not correspond to the origin of the Cartesian system in the semantic space, corresponds to the origin in the error space by construction

semantic space. Usually (except in the rare case where the target value is equal to zero for each training case), the target is also a point in this space which does *not* correspond to the origin of the Cartesian system.

- Each point in the semantic space can be translated by subtracting the target from it (in symbolic regression problems, the target is known for all the training cases, and this is the only kind of problem we are considering in this work). In this way, for each individual, we obtain a new point, that we call *error vector* (and we call *error space* the corresponding space). The target, by construction, corresponds to the origin of the Cartesian system in the error space.
- As we have shown in Ruberto et al. (2014), and as we will also present in Sect. 8.2, if we are able to find (for instance using GP) two individuals whose error vectors are aligned between each other and with the origin, then we are able to reconstruct a globally optimal solution analytically.

Exploiting this idea, the task of GP now becomes looking for two individuals such that the segment joining their error vectors also intersects the origin. In Ruberto et al. (2014) we have presented a possible GP model, called ESAGP (which stands for *Error Space Alignment GP*). The objective of this chapter is to deepen and better specify and discuss ESAGP, and also present an alternative model, which is presented for the first time here, that we call *Pair Optimization GP* (POGP). The main difference between ESAGP and POGP is that, while in ESAGP (as in standard GP), an individual is a single expression (for instance represented by a tree), in POGP an individual is a pair of expressions (for instance a pair of trees) and their fitnesses quantify their respective alignment with the origin in the Cartesian space.

8.2 Alignment in the Error Space

Let $\mathbf{X} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_n}\}$ be the set of input data, or fitness cases, of a symbolic regression problem, and $\overrightarrow{t} = [t_1, t_2, ..., t_n]$ the vector of the respective expected output or target values (in other words, for each $i = 1, 2, ..., n$, t_i is the expected output corresponding to input $\overrightarrow{x_i}$). A GP individual (or program) P can be seen as a function that, for each input vector $\overrightarrow{x_i}$ returns the scalar value $P(\overrightarrow{x_i})$. Consistently with what we have said in the previous section, we call *semantics* of P to the vector $\overrightarrow{s_P} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), ..., P(\overrightarrow{x_n})]$ and error vector of P the vector $\overrightarrow{e_P} = \overrightarrow{s_P} - \overrightarrow{t}$. As Fig. 8.1 shows, an error vector is the translation of a point in the semantic space (by subtraction of the target) and can be represented as a point in another n-dimensional space, the error space.

It is worth noticing that, once we have the error vector of an individual P, it is immediately possible to calculate the root mean square error (RMSE) of P on training data (RMSE $= \sqrt{\sum_{i=1}^{n} e_i^2}$, where e_i is the i^{th} coordinate of $\overrightarrow{e_P}$), a measure that is often used as fitness by standard GP in symbolic regression problems (see for instance Koza (1992)).

Let us now consider the following definition:

Definition 8.1 Optimally Aligned Individuals Two GP individuals A and B are *optimally aligned* if it exists a scalar constant k such that: $\overrightarrow{e_A} = k \cdot \overrightarrow{e_B}$

In other words, two individuals A and B are said to be optimally aligned if their respective error vectors are directly proportional, with a proportionality constant k. The reason why we use the term "aligned" for such individuals becomes clear by looking at Fig. 8.2a, where a simple bi-dimensional error space is represented. In this figure, A and B are two optimally aligned individuals: the points that represent their respective error vectors are aligned with each other and with the origin of the Cartesian system.

Now, let A and B be two optimally aligned individuals. Then, directly applying Definition 1, we have $\overrightarrow{e_A} = k \cdot \overrightarrow{e_B}$. Applying the definition of error vector, the previous equation can be rewritten as $\overrightarrow{s_A} - \overrightarrow{t} = k \cdot (\overrightarrow{s_B} - \overrightarrow{t})$, from which it follows that $\overrightarrow{t} = \frac{1}{1-k} \cdot \overrightarrow{s_A} - \frac{k}{1-k} \cdot \overrightarrow{s_B}$. This implies that, if we find two optimally aligned individuals, whose syntactic structure we represent with A and B, and if we know the proportionality factor k between their respective error vectors, then individual whose syntactic structure is:

$$P_{opt} = \frac{1}{1-k} \cdot A - \frac{k}{1-k} \cdot B \tag{8.1}$$

has a semantic vector that perfectly corresponds to target $\overrightarrow{t}$, and thus it is a globally optimal solution. Interestingly, this property holds independently from the quality (for instance measured by means of the RMSE) of A and B: even two extremely "bad" individuals (in terms of RMSE), if they are optimally aligned, can be used to produce a globally optimal solution. As a direct consequence, the new objective of GP can now be to find two optimally aligned individuals, instead of directly finding a globally optimal solution.

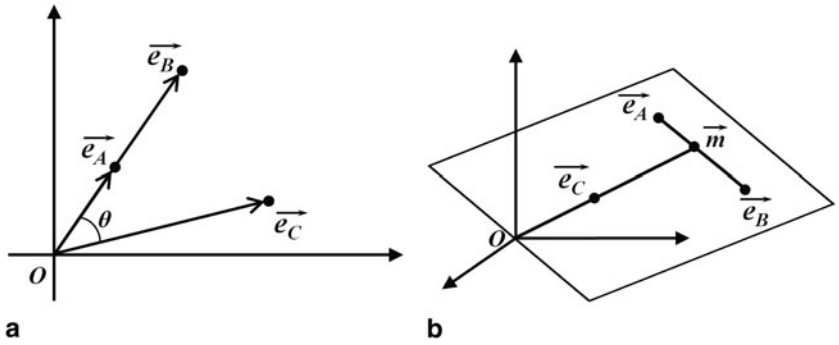


Fig. 8.2 **a** Representation of a simple bi-dimensional error space. Individuals A and B are optimally aligned, i.e. their respective error vectors are directly proportional. The angle between the error vector of A (as well as B) and the one of C is θ. **b** A simple tri-dimensional error space. We point out that it is possible to find a point m that is aligned with the error vectors of any pair of individuals A and B and optimally aligned with a third individual C

8.3 The First GP Model Based on Error Space Alignment: ESAGP

In this section, we present and discuss the ESAGP model. In particular, we define a system, called ESAGP-1, whose objective is to find a pair of optimally aligned individuals. Successively, we give hints on how to generalize the approach, presenting the idea to develop ESAGP-μ systems, with $\mu > 1$. In Ruberto et al. (2014) we have developed ESAGP-2, so the interested reader is referred to that paper in order to deepen the subject. The content of this section is inspired by Ruberto et al. (2014).

8.3.1 One Step Error Space Alignment GP: ESAGP-1

ESAGP-1 is based on the idea that GP should work with the objective of minimizing the *angle* between the error vectors of pairs of individuals (looking for a pair for which this angle is equal to zero). Fig. 8.2a graphically represents the angle θ between the error vectors of individuals A (as well as B) and C. Remembering that $\theta = \arccos((\overrightarrow{e_A} \times \overrightarrow{e_C})/(||\overrightarrow{e_A}|| \cdot ||\overrightarrow{e_C}||))$ (where $\times$ represents the scalar product between two vectors and $||\overrightarrow{v}||$ is the Euclidean norm of vector $\overrightarrow{v}$), the angle between the error vectors of two individuals is easy to calculate once we have their semantics. It is worth emphasizing that the objective of ESAGP-1 is to find optimally aligned individuals, regardless of their individual quality (for instance, as measured by the RMSE). To achieve this goal, we follow two ideas: (1) all the individuals found during the evolution, and not only the ones in the population at each generation, can be potential members of an optimally aligned pair; (2) the search cannot be driven by a measure of distance to the target in the semantic space (like the RMSE),

but instead by a different fitness function that promotes the discovery of optimally aligned individuals.

To implement idea (1), ESAGP-1 maintains an archive of all the "semantically new" individuals that have been found during the GP run. Every time a new individual P is generated, the algorithm checks whether it is optimally aligned with any of the individuals already in the archive. If it is not, P is added to the archive, unless the archive already contains an individual with the same semantics, and the algorithm continues. Otherwise, the algorithm terminates returning the newly found pair of optimally aligned individuals.

To implement idea (2), ESAGP-1 uses a fitness function that has no relationship with the distance to the target in the semantic space. To define this new fitness function, ESAGP-1 calculates a particular point in the error space, that we call *center of attraction*, or simply *attractor*. The fitness of an individual is the angle between its error vector and the attractor, and it has to be minimized (in other words, small angles are better than large ones). The attractor must be chosen in such a way to promote the evolution of optimally aligned individuals. Our idea is to choose a point that, informally, represents the majority of the individuals in a population, standing "in the middle of" an area where most of the error vectors of the individuals in the population are found. Therefore, the objective of the algorithm becomes driving the population towards this central point (as in Ruberto et al. (2014)). Also in this work we use as attractor the following vector: $\overrightarrow{a} = \sum_{P \in \texttt{Pop}} \overrightarrow{e_P} / ||\overrightarrow{e_P}||$ where `Pop` is the initial population of a GP run and $||\overrightarrow{v}||$ is the Euclidean norm of vector $\overrightarrow{v}$.

8.3.2 Possible Generalizations: ESAGP-μ with $\mu > 1$

Let us assume that we have two individuals, like A and B in Fig. 8.2b, i.e. two individuals such that the straight line joining their error vectors is not aligned with the origin. In this case, it is always possible to find a point $\overrightarrow{m}$ that lies on the straight line joining $\overrightarrow{e_A}$ and $\overrightarrow{e_B}$ that is aligned with the error vector of another individual C and the origin. This property holds for any three points, like A, B and C, that lie on a bi-dimensional plane intersecting the origin. In Ruberto et al. (2014) we have shown that given three individuals like A, B and C, we can obtain a globally optimal solution analytically. The proof is simply an "iteration" of the proof used in Sect. 8.3.1 for two optimally aligned individuals. Thus, in Ruberto et al. (2014), we have also introduced a system, that we have called ESAGP-2, that can be seen as a generalization of ESAGP-1 aimed at finding three individuals whose error vectors lie on a bi-dimensional plane intersecting the origin. Interestingly, this process can be iterated: for any μ between 1 and the number of fitness cases, it is possible to define a GP system whose objective is to find $\mu + 1$ individuals that belong to the same μ-dimensional hyperplane intersecting the origin (a property that can also be seen as the composition of μ alignments), which we hypothetically call ESAGP-μ.

The focus of this chapter is on the discussion of GP models aimed at finding pairs of optimally aligned individuals. So, here we concentrate on ESAGP-1, considering

generalized systems ESAGP-μ, with $\mu > 1$, out of the scope of this manuscript. Nevertheless, we believe that the definition of a general strategy allowing us to obtain an ESAGP-μ system for any possible number of dimensions μ is a very promising research track, and it is an important part of our current work.

8.4 Experimental Study of ESAGP-1

8.4.1 Test Problems

To test ESAGP-1, we have chosen two hard regression problems in the field of drug discovery. The objective of these problems is the prediction of two important pharmacokinetic parameters: human oral bioavailability (%F) and median lethal dose (LD50), also called toxicity, of medical drugs. Both problems have already been tackled by GP in published literature and for a discussion of them the reader is referred to Archetti et al. (2007). The %F (respectively LD50) dataset consists in a matrix of 260 (respectively 234) rows (instances) and 242 (respectively 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. Both these datasets are freely available from the GP benchmarks website, gpbenchmarks.org.

8.4.2 Experimental Settings

In this section, we compare ESAGP-1 with a standard version of GP (ST-GP from now on), which is the GP version originally defined by Koza in Koza (1992), and with geometric semantic GP as implemented in Vanneschi et al. (2013a) (GS-GP from now on), and deeply described in Vanneschi et al. (2013b).

For each of the three GP variants (ST-GP, GS-GP and ESAGP-1), 30 independent runs were performed, each using one of 30 different random partitions of the dataset in training (70%) and test (30%) sets. In each run, for each generation we record the RMSE of the best individual on the training set, and the RMSE of the same individual on the test set (and also the number of nodes of that individual). The results we report are the median values of the 30 runs. All the runs used populations of 100 individuals. Tree initialization was performed with the Ramped Half-and-Half method (Koza 1992) with a maximum initial depth of 6. The function set contained the four binary arithmetic operators $+$, $-$, $*$, and / protected as in Koza (1992). The terminal set contained as many variables as the number of features of each dataset. Tournaments of size 4 were used to select the parents of the new generation. To create new individuals, ST-GP and ESAGP-1 used standard (subtree swapping) crossover (Koza 1992) and (subtree) mutation (Koza 1992) with probabilities 0.9 and 0.1, respectively. GS-GP used geometric semantic crossover (Moraglio et al. 2012)

and geometric semantic mutation (Moraglio et al. 2012), each with probability 0.5 (the best setting according to Vanneschi et al. (2013a)). Survival was elitist, as it always copied the best individual into the next generation.

8.4.3 Experimental Results

The results we have obtained are reported in Fig. 8.3. For all the experiments, tests of statistical significance were performed. In particular, the Kolmogorov-Smirnov test has shown that, for all 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.

Plots (a) and (b) report the results obtained on the %F training and test sets and plots (c) and (d) report the results on the LD50 training and test sets respectively. In both cases, the small inset in the upper right corner portrays the results of the evolution during the first 50 generations. On these small plots we can see that in all cases ESAGP-1 outperforms ST-GP and GS-GP on both training and test sets. According to the Wilcoxon test, the differences at generation 50 are statistically significant. However, as shown in Vanneschi et al. (2013a); Vanneschi et al. (2013b), GS-GP is a powerful but slow method: geometric semantic operators induce a unimodal fitness landscape, but also explore it with small steps. Thus, for a fair comparison, the execution of GS-GP has to be continued for a number of generations larger than 50. The large plots (a) and (c) show that on the training set ST-GP and GS-GP at generation 350 find solutions of comparable quality to the ones found by ESAGP-1 at generation 50 (the differences are not statistically significant). The large plots (b) and (d) show that on the test set for %F ESAGP-1 in 50 generations finds results that are comparable with or better than the test set for LD50 found by GS-GP in 350 generations and always better (for both test problems) than the ones found by ST-GP in 350 generations. Interestingly, even if we allow GS-GP to run for 2000 generations (not shown), the results obtained on the test set are not statistically different from the ones obtained by ESAGP-1 in 50 generations. Plots (e) and (f) report the evolution of the number of nodes of the best individual and show that the solutions produced by ESAGP-1 are (in a statistically significant way) smaller than the ones produced by ST-GP. Looking at the definition of geometric semantic operators (Moraglio et al. 2012), it is not difficult to see that, if stored in memory (something that the implementation proposed in Vanneschi et al. (2013a) does not), the individuals generated by GS-GP at generation 50 would have a length of approximately 10^{16} nodes (and this is the reason why the curve of GS-GP is not shown in these plots). Summarizing, not only ESAGP-1 is able to find solutions of the same quality as, or better than, GS-GP, much faster, but these solutions are also smaller, which represents an additional and important advantage.

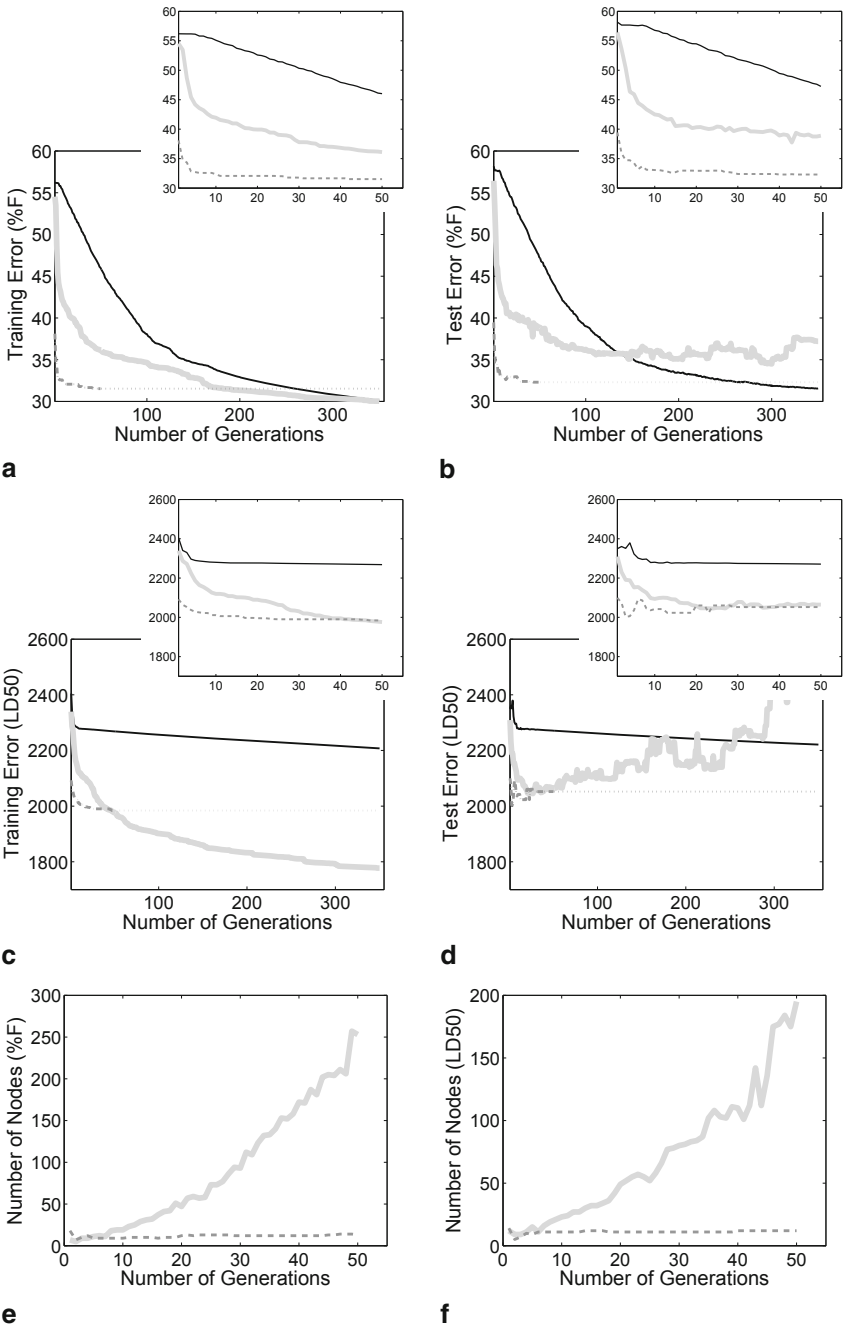


Fig. 8.3 Plot (a) shows the evolution of the (median) best fitness on the training and plot (b) shows the (median) best fitness on the test set for the %F problem. The insets show an enlargement of the plots for the first 50 generations. Plots (c) and (d) are analogous to plots (a) and (b), but for the LD50 problem. Plot (e) (respectively plot (f)) shows the evolution of the (median) number of nodes of the best individual for the %F (respectively LD50) problem

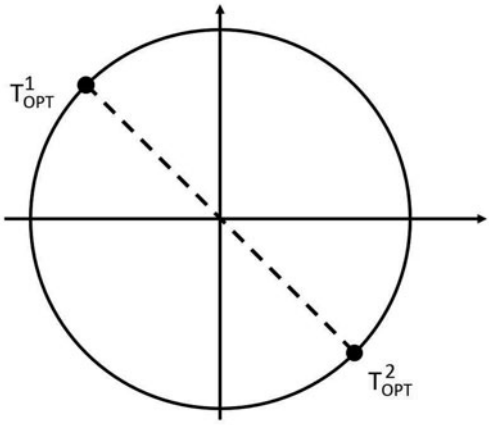


Fig. 8.4 Optimally aligned trees

8.5 Pair Optimization GP

In this section we describe a new design of the alignment-based framework described in the previous sections. In particular, with the model proposed here we want to address the main problem related to the use of ESAGP-1, namely the computational complexity and consequent slowness of the algorithm. This slowness is caused by the process employed by ESAGP-1 in order to find two error vectors that are aligned in the error space.

In order to introduce the new alignment-based model, called Pair Optimization GP (POGP), it is necessary to specify what an individual in the GP population is. From now on, a candidate solution is represented as a couple of trees (but the model is general and it is independent from the representation of the candidate solutions). Having a population in which each individual consists of a couple of trees, our objective is to find an individual such that the error vectors of the two expressions form an angle of 180° between each other (see Fig. 8.4). Thus, the evolutionary search process is guided by a fitness function that only considers the angle formed by the two error vectors. Hence, it is possible to avoid the process that was used in ESAGP-1 in order to find two aligned error vectors. With this individual's design it is necessary to explain how crossover and mutation are defined. Let $T1$ and $T2$ be the trees that form individual $I1$, and let $T3$ and $T4$ be the trees that form individual $I2$. The crossover simply swaps a subtree of $T1$ with a subtree of $T3$ and a subtree of $T2$ with a subtree of $T4$. The mutation operator acts probabilistically on one of the two trees that form an individual or on both the trees.

While this can have a beneficial effect on the speed of the search process, it is necessary to consider some possible problems. The first one is depicted in Fig. 8.5. While the error vectors of T_1 and T_2 form the same angle with respect to the error vector of T_*, error vectors of T_1 and T_2 present a different alignment with respect to the error vector of T_* and with the origin of the error space axes.

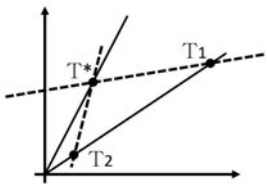


Fig. 8.5 Alignment and placement of the trees in the error space before considering the projective plane

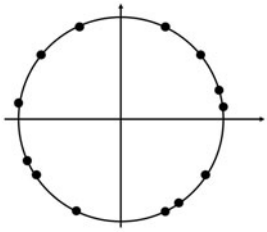


Fig. 8.6 Trees on the hyper-sphere in the $n-1$ dimensional space

In order to avoid this kind of situations, we consider a projective plane that, in this study, consists of an hyper-sphere in the $n-1$ dimensional space (where n is the size of the original search space). In particular, all the error vectors of all the expressions forming the individuals in the population are projected on this hyper-sphere before calculating the angles between each other. This operation allows us to have a search space in which all the individuals (or, better, the trees forming the individuals) lie on the circumference of the hyper-sphere of radius 1. The situation is depicted in Fig. 8.6. Is is important to underline that considering the hyper-sphere is a design choice and it is possible to project points on a different hyper-shape obtaining a different arrangement. Anyway, considering the fact that we are considering angles between vectors the choice of the hyper-sphere is the one that is more natural.

Once the trees have been placed on the hyper-sphere, the search process can start. At this point, it is important to consider another key factor for the success of the proposed POGP: as it is possible to note, the error vectors of the trees T_x^1 and T_x^2 that form the individual x are perfectly aligned if the angle between them corresponds to a flat angle or if the angle is a 0° angle. While the first angle corresponds to the optimal solution (considering a search process that uses as fitness function the maximization of the angle), in this work we penalize a 0° angle assigning to the corresponding individual a very poor fitness. This is exactly what we want: in fact, a 0° angle is formed on the hyper-sphere if and only if T_x^1 and T_x^2 have the same semantics (hence they correspond to the same point in the error space). As it is possible to note, having two trees with the same semantics causes a problem in Eq. 8.1. Hence, also in this case, we must penalize an individual whose trees T_x^1 and T_x^2 have the same semantics.

8.5.1 Technical Problems in the Fitness Calculation

While POGP seems to be able to overcome the main drawback related to the usage of ESAGP-1, there are other problems that must be considered. In particular, in this section we want to point out a numerical problem that we have found in the first experiments that we have performed.

In particular, let I_{BEST} be the individual found at the end of the search process and T^1_{BEST} and T^2_{BEST} be the trees that form T_{BEST}. For reconstructing the optimal solution and calculating its error with respect to the target (hence for applying Eq. 8.1), we have to calculate the proportionality factor k between the error vectors of T^1_{BEST} and T^2_{BEST}. In order to do that, we have to remap the points lying on the hyper-sphere on the original n dimensional space. After doing that, we have to consider, for each fitness case f_i, the ratio $\frac{ei^1}{ei^2}$, where ei^1 and ei^2 are respectively the error obtained from the difference between the output of $T^1_{BEST}$and T^2_{BEST} and the target t_i on the fitness case f_i. At this point, a numerical problem may arise: let us assume that ei^1 and ei^2 present different orders of magnitude (that is not uncommon in several applications), but both ei^1 and ei^2 differ widely from the target t_i. Under this hypothesis we can have two possible situations: (1) $ei^1 \gg ei^2$ (2) $ei^2 \gg ei^1$. In the first case the ratio $\frac{ei^1}{ei^2}$ (used to determine the value of the proportionality constant k) will be a large number, while in he second case the ratio is approximately 0. This will create an undesirable effect in Eq. 8.1. In particular, a large value of k suggests that the semantics of T^2_{BEST} is able to approximate the target value t_i and that the semantics of T^1_{BEST} can be basically ignored; on the other hand, a value of $k \approx 0$ suggests that the semantics of T^1_{BEST} is able to approximate the target value t_i and that the semantics of T^2_{BEST} does not have a significant contribution. In both cases, under the hypothesis in which both ei^1 and ei^2 differ widely from the target t_i, the semantics of T_{BEST} on f_i will differ widely from the target t_i.

This is an important problem that must be taken into account. In particular, this numerical problem will affect the quality of the predictive model if it affects the median of the k values obtained considering all the ratios $\frac{ei^1}{ei^2}$ for all the fitness cases.

We are currently working on this problem, in order to find solutions that do not create this problem. Anyway, it seems that there are two conflicting constraints:

- The search process looks for trees that must have a different semantics; if the semantics are "too similar" we will have a value of k that is not acceptable and an angle that is close to zero.
- On the other hand, to avoid the numerical problem, the semantics of T^1_{BEST} and T^2_{BEST} cannot be "too different".

As the reader can understand, it will be necessary to define a smart method in order to consider both constraints. While we are working on this problem, the experimental results reported in the next section have been obtained considering an experimental settings that allows us to avoid the occurrence of it on the studied test functions.

Table 8.1 Benchmark problems

Name	Definition	Number of variables
P_1	$3 + (2.13 * \log(x_4))$	9
P_2	$1.3 + (0.13 * \sqrt{(x_0)})$	9
P_3	$1/(1 + x_0^{-4}) + 1/(1 + x_1^{-4})$	2
P_4	$1.57 + (24.3 * x_3)$	9

Table 8.2 p-values returned by the statistical tests on the considered benchmark problems considering fitness values at the last generation

		TRAINING		TEST	
		ESAGP-1	POGP	ESAGP-1	POGP
P_1	GS-GP	3.00E-11	3.02E-11	3.00E-11	3.02E-11
	ESAGP-1		1.95E-10		3.48E-09
P_2	GS-GP	2.96E-11	3.02E-11	2.96E-11	3.02E-11
	ESAGP-1		1.15E-09		6.61E-10
P_3	GS-GP	1.70E-12	7.88E-12	1.70E-12	2.58E-12
	ESAGP-1		5.39E-12		1.66E-08
P_4	GS-GP	2.90E-11	3.01E-11	2.90E-11	3.01E-11
	ESAGP-1		3.67E-10		1.57E-06

8.6 Experimental Phase

In this section we report the experimental results achieved considering 4 benchmark problems. In more detail, we compare the results achieved considering the semantic-based methods already used in the previous sections: GS-GP, ESAGP-1 and POGP.

The benchmark problems are the ones reported in Table 8.1. Benchmark $P1$, $P2$ and $P4$ have 9 independent variables, but only one variable is used to determine the output of the function. $P3$ is a problem in spatial co-evolution and has 2 independent variables. These functions belong to the set of GP benchmarks discussed in White et al. (2013).

Regarding the experimental settings, we used the same configuration presented in Sect. 8.4.2 with two differences: each run consists of 500 generations and the set of the functional symbols only contains $+$ and $-$. We decided to consider this set of functional symbols because it allows us to limit the numerical problem that has been discussed in the previous section.

The results are reported in Fig. 8.7 in a similar fashion to those shown in Fig. 8.3 and display the curve of the RMSE on training and test instances. In all the plots, we reported the median RMSE obtained over 30 independent runs. As with the experiments discussed in sect. 8.4.3, the same statistical tests have been executed considering training and test fitness obtained at the end of the evolutionary process. The p-values are reported in Table 8.2.

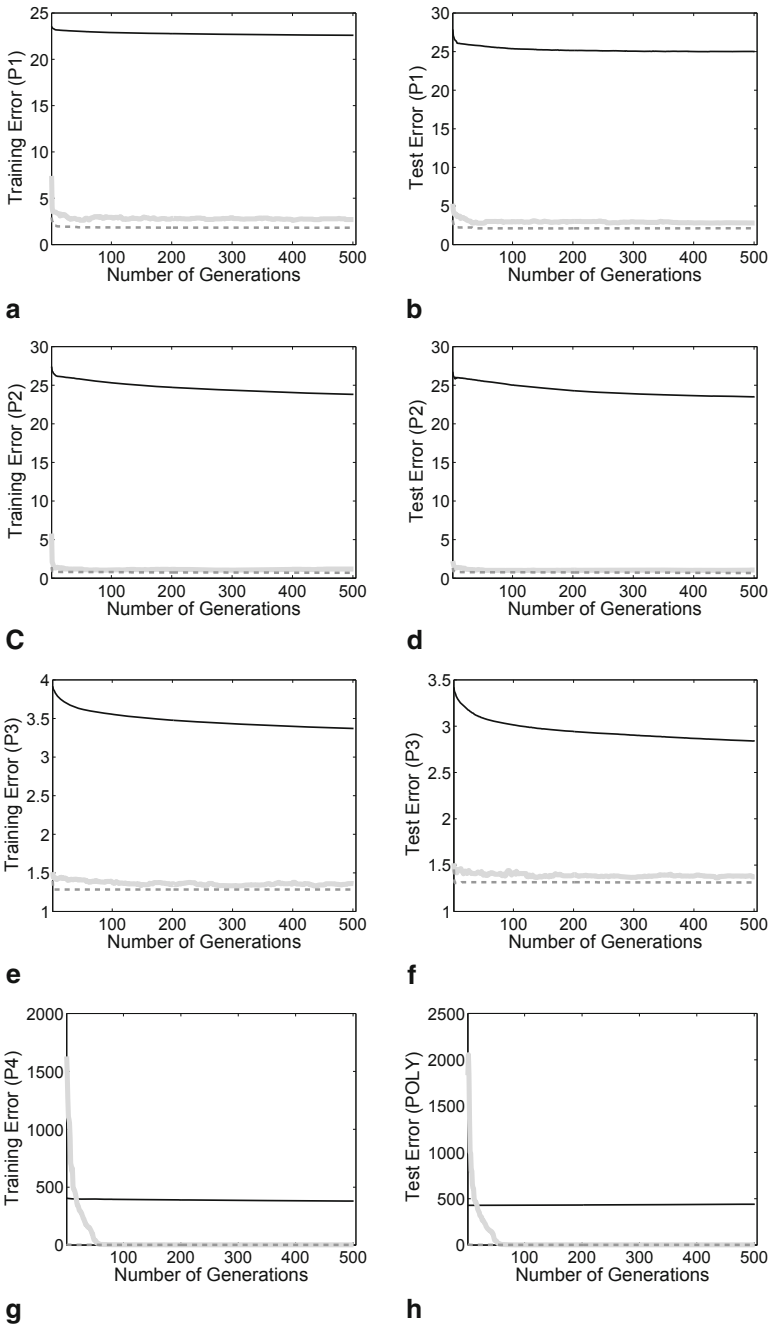


Fig. 8.7 Median of training and test fitness obtained over 30 independent runs for the considered benchmark problems. The legend for all the plots is

As it is possible to see, on all the problems considered, GS-GP performs worse than both ESAGP-1 and POGP. This behaviour can be explained by the particular experimental settings that we have considered. In particular, GS-GP usually needs more generations (with respect to ESAGP-1) in order to converge to optimal solutions. In the experimental phase that has been performed, this characteristic of GS-GP is amplified by the presence of only 2 functional symbols. Hence, more and more iterations are needed in order to have satisfactory fitness value with GS-GP.

Regarding ESAGP-1 and POGP, in all the benchmark problems it is possible to observe a common behaviour: ESAGP-1 is able to produce better results with respect to POGP. This result was expected as ESAGP-1 and POGP share a common idea about the alignment of vectors in the error space. The difference between them is that POGP tries to find two optimally aligned vectors using only a couple of trees. On the other hand, ESAGP-1 has a greater degree of freedom, because it tries to find two optimally aligned vectors using the error vectors of a large number of trees. For the sake of completeness, it is important to point out that POGP is faster than ESAGP-1, hence it can adress more complex problems, where a large amount of data is available, in an acceptable amount of time. Nevertheless, if we consider the execution time, the best technique is GS-GP, as it is at least 20 times faster than POGP.

To summarize, the three semantic-based techniques considered in this experimental phase present different trade-offs in terms of speed and performances. While GS-GP is faster than the other two techniques, it requires a larger number of generations in order to achieve satisfactory results. On the other hand, POGP and ESAGP-1 are slower than GS-GP but they are able to achieve satisfactory fitness values in a small number of generations.

8.7 Conclusions and Future Work

We have shown that, if we are able to find a pair of optimally aligned individuals (i.e. individuals such that the straight line joining their error vectors intersects the origin of the Cartesian system in the error space), then we can reconstruct a globally optimal solution in few simple analytical steps. Using this information, the objective of this chapter was to propose two different models of genetic programming (GP) aimed at finding, or approximating, pairs of optimally aligned individuals. The models we have presented are ESAGP-1 (One Step Error Space Alignment GP), that had already been introduced in Ruberto et al. (2014), and POGP (Pair Optimization GP).

The ESAGP-1 system represents individuals as simple expressions (like standard GP). It works by updating a repository of semantically different individuals visited by GP during the evolution and looks for individuals that are optimally aligned with the ones stored in the repository. In ESAGP-1 fitness is measured as the angle between the error vector of an individual and a particular point called attractor. In POGP individuals are pairs of expressions, and fitness is the angle between the error vectors of these two expressions.

At the time of the publication of this work, ESAGP-1 is much a more mature system than POGP. In fact, we have developed, tested and tuned ESAGP-1 for several months so far. Thus, we have been able to execute ESAGP-1 on complex real-life problems (including the two regression problems in the drug discovery field here). The results we have obtained indicate that ESAGP-1 is able to find solutions that are comparable of even better than standard GP and also than a recently defined GP system that exploits the geometry of the semantic space to induce unimodal fitness landscapes (called GS-GP, as defined in Moraglio et al. (2012) and efficiently implemented in Vanneschi et al. (2013a)). Interestingly, ESAGP-1 finds these results much faster and also returns significantly smaller models compared to standard GP and GS-GP.

Compared to ESAGP-1, POGP is a relatively new development for our research group. We have, in fact, just begun to develop and study it. For this reason, although the idea that inspires POGP seems very promising, we have been able only to present a preliminary study on a set of well known hand tailored symbolic regression benchmarks. The extremely promising results that we have obtained on those benchmarks encourage us to pursue the study of POGP in the future, extending it to real-life problems.

Acknowledgements The authors acknowledge projects MassGP (PTDC/EEI-CTP/2975/2012) and InteleGen (PTDC/DTP-FTO/1747/2012), FCT, Portugal.

References

Archetti F, Lanzeni S, Messina E, Vanneschi L (2007) Genetic programming for computational pharmacokinetics in drug discovery and development. Genet Program Evol Mach 8:413–432

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

Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. Parallel Problem Solving from Nature, PPSN XII (part 1, vol 7491). (Lecture notes in computer science). Springer, Berlin, pp 21–31

Poli R, Langdon WB, Mcphee NF (2008) A field guide to genetic programming. http://www.gp-field-guide.org.uk

Ruberto S, Vanneschi L, Castelli M, Silva S (2014) A new implementation of geometric semantic GP applied to predicting pharmacokinetic parameters. Proceedings of the 17th European conference on Genetic Programming, Springer

Vanneschi L, Castelli M, Manzoni L, Silva S (2013a) A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. Proceedings of EuroGP 2013, Springer, LNCS, pp 205–216

Vanneschi L, Silva S, Castelli M, Manzoni L (2013b) Geometric semantic genetic programming for real life applications. Genetic Programming Theory and Practice XI, Genetic and Evolutionary Computation, Springer US, Computer Science Collection, invited article. To appear

Vanneschi L, Castelli M, Silva S (2014) A survey of semantic methods in genetic programming. Genet Program Evol Mach 14(1):1–20

White DR, McDermott J, Castelli M, Manzoni L, Goldman BW, Kronberger G, Jaśkowski W, OReilly UM, Luke S (2013) Better gp benchmarks: community survey results and proposals. Genet Program Evol Mach 14(1):3–29

Mauro Castelli is an Assistant Professor at the Nova Information Management School (NOVA IMS) of the Nova University of Lisbon, Portugal.

Leonardo Vanneschi is an Associate Professor of at the Nova Information Management School (NOVA IMS) of the Nova University of Lisbon, Portugal. Also, he is the Director of the Master in Advanced Analytics in the NOVA IMS.

Sara Silva is a principal investigator of the Faculty of Sciences of the University of Lisbon (FCUL), Portugal, an invited researcher of the Center for Informatics and Systems of the University of Coimbra (CISUC), Portugal, and an invited assistant professor of the Nova Information Management School (NOVA IMS) of the Nova University of Lisbon, Portugal.

Stefano Ruberto is a PhD student at the Gran Sasso Science Institute (GSSI) of L'Aquila, Italy.

Chapter 9
Analyzing a Decade of Human-Competitive ("HUMIE") Winners: What Can We Learn?

Karthik Kannappan, Lee Spector, Moshe Sipper, Thomas Helmuth, William La Cava, Jake Wisdom and Omri Bernstein

9.1 Introduction

In the field of evolutionary computation (EC) ideas from evolutionary biology—random variation and selection—are harnessed in algorithms that are applied to complex computational problems. The origins of EC can be traced back to the 1950s and 1960s but the field has come into its own over the past two decades, proving successful in solving numerous problems from highly diverse domains (Sipper 2002). EC techniques are being increasingly applied to difficult real-world problems, often yielding results that are not merely academically interesting but also competitive with the work done by creative and inventive humans. Indeed, a recent emerging theme is that of *human-competitive machine intelligence*, produced by evolutionary means (Koza 2008, 2010).

K. Kannappan (✉) · T. Helmuth
School of Computer Science, University of Massachusetts, Amherst, MA, USA
email: kkarthik@cs.umass.edu

T. Helmuth
e-mail: thelmuth@cs.umass.edu

L. Spector · J. Wisdom · O. Bernstein
Cognitive Science, Hampshire College, Amherst, MA 01002-3359, USA
e-mail: lspector@hampshire.edu

J. Wisdom
e-mail: jake.r.wisdom@gmail.com

O. Bernstein
e-mail: hippiccolo@gmail.com

M. Sipper
Department of Computer Science, Ben-Gurion University, 84105 Beer-Sheva, Israel
e-mail: sipper@cs.bgu.ac.il

W. La Cava
Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA, USA
e-mail: wlacava@umass.edu

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_9

A recent survey cited 28 instances in which genetic programming (GP), a form of EC, "has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention" and cited "over a dozen additional known instances where genetic programming has produced a human-competitive result that is not patent related" (Koza 2008). These results come from an astonishing variety of fields, including image analysis, game playing, quantum computer programming, software repair, and the design of complex objects such as analog circuits, antennas, photonic crystals, and polymer optical fibers.

We believe this to be more than a mere novel line of research within a single research community. Surpassing humans in the ability to solve complex problems is a *grand challenge*, with potentially far-reaching, transformative implications.

In this chapter we take a close look at the 42 winners of the past decade (2004–2013) of Human-Competitive (HUMIE) competitions, seeking to draw conclusions about past and future directions of the field.

We note that two of the authors (Spector, Sipper) have extensive experience in human-competitive research, having won between them eight HUMIE awards (Koza 2010). In addition, Spector has served as a judge for the HUMIES awards for some years. Spector and his colleagues earned the competition's top prize twice, once for the use of EC to produce quantum computing results that were published in a top physics journal (Barnum et al. 2000; Spector 2004) and once for results in pure mathematics that exceeded human performance by several orders of magnitude (Spector et al. 2008). Sipper, who has six wins, tackled a string of hard games and puzzles, evolving game-playing strategies that held their own in competition against humans (Sipper 2006; Hauptman et al. 2009; Benbassat et al. 2012). In collaborative work with a partner from the semiconductors industry Sipper attained marked improvement over humans in developing automatic defect classifiers for patterned wafers (Glazer and Sipper 2008).

9.2 The HUMIES

As of 2004, one of the major annual events in the field of evolutionary computation — the Genetic and Evolutionary Computation Conference[1]—boasts a competition that awards prizes to human-competitive results: The HUMIES. As noted at the competition site (Koza 2010): "Techniques of genetic and evolutionary computation are being increasingly applied to difficult real-world problems—often yielding results that are not merely interesting, but competitive with the work of creative and inventive humans."

To set the stage for our current work we provide examples of HUMIE winners in two important areas: pure mathematics and games.

[1] see `sigevo.org`

Pure Mathematics

Spector has produced significant new results in the application of genetic programming to mathematics, in collaborative work with Distinguished Professor of Mathematics David M. Clark, at the State University of New York at New Paltz. Together with two Hampshire College undergraduates and one Hampshire alumnus, they applied genetic programming to a problem in pure mathematics, in the study of finite algebras.

Algebraists have been looking at finding "terms" that represent specific functions in specific algebras for several decades, with particular interest attaching to the discovery of terms for Mal'cev functions (the significance of which was first made clear in 1954), Pixley functions (1963), the discriminator function (1970), and majority functions (1975). The most effective methods previously developed for finding these terms are uniform search (including exhaustive search and random search) and construction via the primality theorem. In exhaustive search terms are enumerated systematically from smallest to largest, while in random search terms within a range of sizes are generated in random order. Exhaustive search will always produce the smallest term of the required type if such a term exists, but it requires astronomical amounts of time, except for the very smallest algebras or the very simplest terms. Random search has similarly problematic performance characteristics but without any guarantees concerning size or success. Construction via the primality theorem gives the most time efficient method known to describe these terms that applies to any primal algebra, but except for the very smallest algebras the terms it produces have astronomical length.

Spector and colleagues documented the application of genetic programming to these term-finding problems, producing human-competitive results in the discovery of particular algebraic terms (e.g., discriminator, Pixley, majority, and Mal'cev terms) and showing that genetic programming exceeded the performance of every prior method of finding these terms in either time or size by several orders of magnitude (Spector et al. 2008). This result earned the gold medal in the 5th Annual HUMIES Awards for Human-Competitive Results Produced by Genetic and Evolutionary Computation, held at the 2008 Genetic and Evolutionary Computation Conference. Subsequently, this work led to the development of new mathematical theory that has been published independently (Clark 2013).

Games

Ever since the dawn of artificial intelligence in the 1950s, games have been part and parcel of this lively field. In 1957, a year after the Dartmouth Conference that marked the official birth of AI, Alex Bernstein designed a program for the IBM 704 that played two amateur games of chess. In 1958, Allen Newell, J. C. Shaw, and Herbert Simon introduced a more sophisticated chess program (beaten in thirty-five moves by a 10-year-old beginner in its last official game played in 1960). Arthur L. Samuel of IBM spent much of the fifties working on game-playing AI programs, and by 1961 he had a checkers program that could play at the master's level. In 1961 and 1963 Donald Michie described a simple trial-and-error learning system for learning how to play Tic-Tac-Toe (or Noughts and Crosses) called MENACE (for Matchbox

Educable Noughts and Crosses Engine). These are but examples of highly popular games that have been treated by AI researchers since the field's inception.

Why study games? On this matter Susan L. Epstein wrote:

> There are two principal reasons to continue to do research on games... First, human fascination with game playing is long-standing and pervasive. Anthropologists have catalogued popular games in almost every culture... Games intrigue us because they address important cognitive functions... The second reason to continue game-playing research is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards (Epstein 1999).

Studying games may thus advance our knowledge in both cognition and artificial intelligence, and, last but not least, games possess a competitive angle which coincides with our human nature, thus motivating both researcher and student alike.

Over the past 7 years Sipper has done extensive research in the area of games (Sipper et al 2007; Hauptman and Sipper 2005b, a; Hauptman and Sipper 2007b; Azaria and Sipper 2005a, b; Benbassat and Sipper 2010; Hauptman and Sipper 2007a; Hauptman et al. 2009; Shichel et al. 2005), which culminated in his recent book, "Evolved to Win" (Sipper 2011) (see also www.moshesipper.com/games). Among the games successfully tackled are: chess, backgammon, checkers, Reversi, Robocode (tank-war simulation), Rush Hour, and FreeCell. These exhibit the full range from two-player, full-knowledge, deterministic board games, through stochastic, simulation-based games, to puzzles.

A recent line of research has attempted to build a more general form of evolution-based game intelligence by employing a structure known as a policy, which is an ordered set of search-guiding rules (Hauptman et al. 2009; Elyasaf et al. 2012). Policies are complex structures that allow one to define specific conditions under which certain actions are performed. They might specify, for example, that a certain stratagem for solving a puzzle becomes relevant when certain conditions hold. The combination of policies and evolution might just prove powerful enough to set up a general "strategizing machine" (Sipper et al. 2007), i.e., one able to automatically evolve successful game strategies given a description of the game in question. Sipper's work on games has garnered five HUMIE awards to date.

9.3 A Compendium of a Decade's Worth of HUMIE Winners

The main intention behind analyzing a decade's worth of HUMIE winners is to be able to determine whether there were any particular aspects that were similar across the various domains that the winners covered. In 2005, John R. Koza, Sameer H. Al-Sakran, and Lee W. Jones (Koza et al. 2005) did some preliminary analysis, looking for cross-domain features of programs evolved using Genetic Programming. We intend to do something similar, but do so from the specific point of view of trying to identify the aspects of the applications (of any evolutionary computation technique, and not just Genetic Programming) that make them human-competitive.

With the HUMIES over a decade old now, we also have a larger set of results to analyze relative to the number of results analyzed in the original Koza paper (Koza et al. 2005).

The actual HUMIES competition is designed to recognize work that has already been published that holds its own against humans in one of many ways. For example, a result produced by EC that reproduces a past patent, or qualifies as a new patentable invention is considered human-competitive. Similarly, a result that is publishable in its own right as a new scientific result (notwithstanding the fact that the result was mechanically created) is also considered human competitive (Koza 2010). Table 9.1 lists, in detail, the various criteria for a program to be considered human-competitive, and also supplies a count of the number of HUMIE winners that have matched each criterion in the past decade of the HUMIES.

The 42 HUMIE winners of the past decade are listed in Table 9.2. In addition to a very brief description of the winning entry and its author(s), the table includes the specific algorithm used by that entry, where GP is Genetic Programming, GA is Genetic Algorithms, ES refers to Evolutionary Strategies, DE refers to Differential Evolution and GBML refers to Genetics Based Machine Learning. A special category for "noise" is also included. An entry is marked as "noisy" if the data that was used to evolve the solution may inherently have some noise, such as data collected from say, a physical measurement, where the source of the noise is the measurement error. An example of an entry that does not have any noise would be trying to perform symbolic regression to fit a curve that is already known mathematically. Since the data that must be fit already has a known mathematical function, there's no real noise involved here as far as the data points that the program that does the regression sees — all data points are perfectly accurate. In our analysis of the HUMIES (Tables 9.1, 9.2, 9.3, 9.4, 9.5), we explicitly chose to ignore whether the entry won a gold, silver, or bronze award, since we believe that this is insignificant to the analysis because an entry that has won any award at all is necessarily human-competitive.

9.4 Lessons Learned

First and foremost, we note that techniques from evolutionary computation have been used to solve problems from a very wide variety of domains in a human competitive way ; the past 10 years of HUMIES awards alone have winners that have solved problems in a human-competitive way in 21 different domains (see Table 9.4). This clearly suggests that techniques based on EC are rather widely applicable, and are not confined to specific fields.

Second, Genetic Programming (GP) and Genetic Algorithms (GAs) certainly seem to be winning strategies at the HUMIES, with 22 papers based on GP and 15 papers based on GA's having won the HUMIES so far (See Table 9.3). In other words, a combined 37 papers out of the 42 overall HUMIE winners, or roughly 88 % of the winners, used either GP or GAs.

Table 9.1 Categories A-H, with a count of the number of HUMIE winners so far winning in this category, and a description of what each category means. Categories and Full Descriptions were obtained from the HUMIES website (Koza 2010)

Category	Brief description	Count	Full description
A	Patented invention	10	The result was patented as an invention in the past is an improvement over a patented invention or would qualify today as a patentable new invention
B	Equal to accepted scientific result	20	The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal
C	Could be put in archive of results	8	The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts
D	Publishable as new scientific result	29	The result is publishable in its own right as a new scientific result independent of the fact that the result was mechanically created
E	Best incremental solution	25	The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions
F	Achievement in field at time of discovery	25	The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered
G	Indisputable difficulty	26	The result solves a problem of indisputable difficulty in its field
H	Human competition contender	9	The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs)

Next, the problem "type" analysis from Table 9.5 seems to show an interesting trend, with a lot of the problems that evolutionary computation (EC) seems to solve human-competitively being design problems. We use the term design in a rather broad way, to include both designing concrete entities such as an antenna, as well as designing more subtle entities, such as say, designing a winning strategy for a game. This leads us to note that EC may be particularly well suited to designing new

Table 9.2 The 42 HUMIE winners of the past decade

Entry	First author	Alg	Type	Year	N	Setting	Application area	Technique	A	B	C	D	E	F	G	H
Antanae	Lohn	GP	Design	2004	X	Government	Electrical engineering, Antennas	Developmental GP				X	X		X	
Quantum	Spector	GP	Programming	2004		Academia	Quantum	Stack-based, Developmental		X		X				
SAT GP heuristics	Fukunaga	GP	Design	2004		Academia	Optimization	Strongly typed GP				X				
Kinematic machine straight line	Lipson	GP	Design	2004		Academia	Mechanical engineering	Developmental GP					X			
Organization Design Optimization	Khosraviani	GP	Optimization	2004		Academia	Operations research	Standard GP				X		X	X	
Circuit design	Stoica	GA	Design	2004	X	Government	Electronics	Mixtrinsic evolution (SW & HW)	X			X				
2D Photonic crystals	Preble	GP	Design	2005		Academia	Photonics	Tree and bitmap representations		X		X	X			
Quantum attosecond dynamics	Bartels	ES	Optimization	2005	X	Academia	Quantum	Standard ES				X	X	X		
Optical lens systems	Koza	GP	Design	2005		Industry	Optics	Developmental GP	X				X	X	X	

Alg Algorithm, *N* Noisy data, A-H are defined in Table 9.1.

Table 9.2 (continued)

Entry	First author	Alg	Type	Year	N	Setting	Application area	Technique	A	B	C	D	E	F	G	H
Quantum fourier transform algorithm	Massey	GP	Design	2005		Academia	Quantum	Developmental GP		X				X	X	
Assembly programs	Edgar	GP	Programming	2005		Academia	Software engineering	Micro GP			X					X
Space systems design	Terrile	GA	Optimization	2005		Government	Mechanical engineering						X	X	X	
Game playing	Sipper	GP	Design	2005		Academia	games	standard GP								X
Image compression	Grasemann	GA	Design	2005		Academia	Image processing	Coevolutionary		X			X	X	X	
Sinusoidal oscillators	Aggarwal	GA	Design	2006		Academia	Electronics	GA		X		X	X	X		
Photochemistry	Sastry	GA	Optimization	2006		Academia	Chemistry	Multi-objective GA		X	X	X	X			
Ellipse detection	Yao	GA	Classification	2006	X	Academia	Image processing	Multi-population	X	X				X		
Interest point detection	Olague	GP	Classification	2006	X	Government	Computer vision	Standard GP		X	X	X	X	X	X	
Polymer optical fibres	Manos	GA	Design	2007		Academia	Polymers	Developmental GA	X	X		X	X			

Table 9.2 (continued)

Entry	First author	Alg	Type	Year	N	Setting	Application area	Technique	A	B	C	D	E	F	G	H
Mate-In-N chess problem	Sipper	GP	Design	2007		Academia	Games	Koza-style GP		X		X		X	X	X
Diagnosing prostate cancer	Llor	GBML	Classification	2007	X	Academia	Medicine			X		X	X			
Automated alphabet reduction method	Bacardit	GA	Clustering	2007		Academia	Biology	Extended Compact Genetic Algorithm (EDA)		X		X	X		X	
Finite algebras	Spector	GP	Regression	2008		Academia	Mathematics	Stack-based, Develop-mental		X		X	X	X	X	
RTL benchmark circuits	Pecenka	GA	Design	2008		Industry, Academia	Electronics	Non-binary GA				X			X	
Evolving automatic defect classification	Glazer	GA	Classification	2008		Industry, Academia	Electronics	Standard GA	X			X	X	X	X	
Software patches	Forrest	GP	Programming	2009		Academia	Software engineering	AST with weighted program path							X	
User identification on smart phones	Shahzad	GA	Classification	2009		Academia	Security	GA with particle swarm optimization	X				X	X	X	

Table 9.2 (continued)

Entry	First author	Alg	Type	Year	N	Setting	Application area	Technique	A	B	C	D	E	F	G	H
GP-Rush - Rush hour puzzle	Hauptpman	GP	Design	2009		Academia	Games	Policy-based GP		X		X		X	X	X
Descriptor operators	Perez	GP	Design	2009		Government	Computer vision	Standard GP	X	X	X	X	X	X	X	
Protein structure prediction	Krasnogor	GP	Regression	2010	X	Academia	Biology	Standard					X	X		X
Domain-Independent satisficing planning	Pierre	MH	Planning	2010		Academia	Planning			X		X	X		X	
Solving iterated functions using GP	Schmidt	GP	Regression	2010		Academia	Mathematics	Symbolic regression algorithm					X	X	X	
Mixed-Integer evolution stategies-medical images	Thomas	ES	Optimization	2010	X	Academia	Image processing, Medicine	Mixed-Integer evolution strategies					X	X	X	
FreeCell	Elyasaf	GA	Design	2011		Academia	Games	Standard GA		X		X		X	X	X

Table 9.2 (continued)

Entry	First author	Alg	Type	Year	N	Setting	Application area	Technique	A	B	C	D	E	F	G	H
Verficiation algorithm for evolvable hardware	Lukas	GP	Optimization	2011		Academia	Electronics	Cartesian GP				X	X		X	
Optimal broad-band Stokes/Mueller polarimeter	Letnes	GA	Design	2011		Academia	Optics	Standard GA	X	X	X	X	X	X	X	
Game design	Browne	GP	Design	2012		Academia	Games	Evolving rule trees			X	X		X		
Automated Probe Microscopy	Woolley	GA	Optimization	2012		Academia	Mechanical engineering	Cellular GA	X			X	X	X	X	X
Automated program repair	Dewey-Vogt	GP	Programming	2012		Academia	Software engineering	GP over AST edit operations				X			X	
FreeCell	Sipper	GP	Design	2013		Academia	Games	Policy-based GP		X		X		X	X	X
Jupiter moon search	Izzo	DE	Design	2013		Government	Mechanical engineering	Self-adaptation differential evolution algorithm, Asynchronous island model			X	X		X		X
Solid state NMR pulse sequences	Bechmann	GA	Optimization	2013		Academia	Physics	Standard GA	X	X	X	X	X	X	X	

Table 9.3 A summary of the algorithms used by the HUMIE winners

Algorithm	Count
Genetic Programming (GP)	22
Genetic Algorithms (GA)	15
Evolutionary Strategies (ES)	2
Differential Evolution (DE)	1
Genetics Based Machine Learning (GBML)	1
Metaheuristic	1

Table 9.4 Categorization of the application domains of the HUMIE winners. Note that some winners may come under multiple application categories. The number in brackets after the application categories denotes the number of HUMIE winners in that particular application category

Application	Count	Application category
Antennas	1	Engineering (19)
Biology	2	Science (7)
Chemistry	1	Science (7)
Computer vision	2	Computer science (7)
Electrical engineering	1	Engineering (19)
Electronics	5	Engineering (19)
Games	6	Games (6)
Image processing	3	Computer science (7)
Mathematics	2	Mathematics (3)
Mechanical engineering	4	Engineering (19)
Medicine	2	Medicine (2)
Operations research	1	Engineering (19)
Optics	2	Engineering (19)
Optimization	1	Mathematics (3)
Photonics	1	Engineering (19)
Physics	1	Science (7)
Planning	1	Computer science (7)
Polymers	1	Engineering (19)
Quantum	3	Science (7)
Security	1	Computer science (7)
Software engineering	3	Engineering (19)

entities from scratch in a human-competitive way. The authors note that classifying problems based on a "type" is a slightly subjective process and that some problems may fit several types at times, but we believe that the above analysis is still sufficient to note how good EC is when it comes to design problems.

Table 9.5 A count of the broad types of problems that the HUMIE winners solved

Problem Type	Count
Classification	5
Clustering	1
Design	20
Optimization	8
Planning	1
Programming	4
Regression	3

Another interesting trend at the HUMIES seems to be the abundance of papers that have combined domain specific knowledge effectively with evolution in a way where evolution helps combine and adapt existing human knowledge in innovative new ways. For example, Stephanie Forrest's paper (Forrest et al. 2009) uses existing human knowledge embedded in non-faulty parts of the code to repair parts of the code that are faulty. Policy based GP (Hauptman et al. 2009; Elyasaf et al. 2012) is another such area where human knowledge is integrated into an EC system that then evolves a solution that is human competitive. This particular trend certainly suggests a rethink of the artificial (intelligence)-to-(human) intelligence (A/I) ratio, suggested by John Koza et al., which states that GP delivers a high amount of artificial intelligence relative to the relatively minimal amount of human intelligence that is put in to the system (Koza et al. 2003). In the context of human-competitive machine intelligence, our analysis suggests that looking for a high A to I ratio is not the best way to seek promising problems. Instead, we suggest that the focus should be on the additional knowledge gained by some automatic technique that makes the system human competitive. To quote Moshe Sipper from his book *Evolved to Win* (Sipper 2011),

> Rather than aiming to maximize A/I we believe the "correct" equation is:
>
> $$A - I \geq M_\varepsilon$$
>
> where M_ε stands for "meaningful epsilon". When wishing to attain machine competence in some real-life, hard-to-learn domain, then, by all means, imbue the machine with as much I(ntelligence) as possible! After all, if imbuing the I reduces the problem's complexity to triviality, then it was probably not hard to begin with. Conversely, if the problem is truly hard, then have man and machine work in concert to push the frontiers of A as far as possible. Thus, it is not max(A/I) that is of interest but the added value of the machine's output: Granting the designer "permission" to imbue the machine with as much I as he can, will it then produce a $\Delta A = A - I$, namely, added intelligence, that is sufficiently meaningful? Even if this meaningful epsilon M_ε is small in (some) absolute terms, its relative value can be huge (e.g., a chip that can pack 1–2 % more transistors, or a game player that is slightly better and thus world champion).

We believe that this approach of looking at the additional intelligence gained by an automated system is crucial not just for Genetic Programming, but for Artificial Intelligence on the whole. We strongly encourage people to build artificial intelligence systems that make as much use of existing human knowledge as possible.

9.5 Concluding Remarks

Overall, analyzing a decade's worth of HUMIES papers seems to suggest that when combined effectively with domain-specific knowledge, GP and GA are approaches to EC that are highly effective in producing human-competitive results, in a very wide array of fields. We strongly encourage further research in the human-competitive domain, particularly with EC approaches such as GA and GP, used in conjunction with human knowledge in the current field. One aspect that we note in particular is the significance of collaboration with experts outside the computer science domain, which leads to several interesting insights in multiple fields.

We would also like to mention the gradual shift from a high Artificial Intelligence to Human Intelligence (A/I) ratio, towards a focus on the additional intelligence gained by using an intelligent system, irrespective of how much human intelligence one supplies to it. One interesting aspect that must be brought up when moving away from high (A/I) is interpretability, particularly when a human is involved in a feedback loop used to improve the system. In her recent work, Cynthia Rudin has been suggesting interpretability as a key feature in several prediction systems, and notes that experts are more likely to use an interpretable system compared to a black box system that they do not understand (Letham et al. 2012; Rudin et al. 2012; Wang et al. 2013). While most of evolutionary computation has historically been using a black-box approach, interpretability might eventually become rather important too (both in EC-based approaches and in other machine learning approaches) to build human-competitive systems, particularly when human experts are involved in both building the system and improving its quality.

In a recent paper Kiri Wagstaff argues that much of current machine learning (ML) research has lost its connection to problems of import to the larger world of science and society (Wagstaff 2012). In reference to the much-used (and perhaps much-abused) UCI archive she eloquently writes,

> "Legions of researchers have chased after the best iris or mushroom classifier. Yet this flurry of effort does not seem to have had any impact on the fields of botany or mycology."

Wagstaff identifies several problems that underlie the "Machine Learning for Machine Learning's Sake" stance, including: overly focusing on benchmark data sets, with little to no relation to the real world; too much emphasis on abstract metrics that ignore or remove problem-specific details, usually so that numbers can be compared across domains; and lack of follow-through:

> "It is easy to sit in your office and run [some] algorithm on a data set you downloaded from the web. It is very hard to identify a problem for which machine learning may offer a solution, determine what data should be collected, select or extract relevant features, choose an appropriate learning method, select an evaluation method, interpret the results, involve domain experts, publicize the results to the relevant scientific community, persuade users to adopt the technique, and (only then) to truly have made a difference."

She argues for making machine learning matter: asking how one's work impacts the original problem domain; greater involvement of domain experts; and considering the potential impact on society of a problem one elects to work on. She proposes

a number of admirable Impact Challenges as examples of machine learning that matters, e.g., a law passed or a legal decision made that relies on the result of an ML analysis, and $ 100 M saved through improved decision making provided by an ML system.

We think that Wagstaff actually bolsters research into human-competitive results produced by EC. Despite her opining that, "human-level performance is not the gold standard. What matters is achieving performance sufficient to make an impact on the world", we think that the problems in Table 9.2 are very strongly coupled to the real world. Indeed, most, if not all, of them have involved expertise (and often actual experts) in a real-world problem domain, and the competition itself sets out to underscore the impact of such research on society at large. Thus, the HUMIE winners may all be unknowingly responding to Wagstaff's challenge, creating machine learning that matters.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grants No. 1017817, 1129139, and 1331283. 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

Azaria Y, Sipper M (2005a) GP-gammon: genetically programming backgammon players. Genet Program Evol Mach 6(3):283–300. doi:10.1007/s10710-005-2990-0 http://www.cs.bgu.ac.il/sipper/papabs/gpgammon.pdf. Accessed 12 Aug 2005

Azaria Y, Sipper M (2005b) Using GP-gammon: using genetic programming to evolve backgammon players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) Proceedings of the 8th European conference on genetic programming. Lecture notes in computer science, vol 3447. Springer, Lausanne, pp 132–142. doi:10.1007/b107383

Barnum H, Bernstein HJ, Spector L (2000) Quantum circuits for OR and AND of ORs. J Phys A: Math Gen 33(45):8047–8057. (http://hampshire.edu/lspector/pubs/jpa.pdf)

Benbassat A, Sipper M (2010) Evolving lose-checkers players using genetic programming. In: IEEE conference on computational intelligence and game, IT University of Copenhagen, Denmark, pp 30–37. doi:10.1109/ITW.2010.5593376 http://game.itu.dk/cig2010/proceedings/papers/cig10-005-011.pdf

Benbassat A, Elyasaf A, Sipper M (2012) More or less? two approaches to evolving game-playing strategies. In: Riolo R, Vladislavleva E, Ritchie MD, Moore JH (eds) Genetic programming theory and practice X, genetic and evolutionary computation. Springer, Ann Arbor, chap 12, pp 171–185 doi:10.1007/978-1-4614-6846-2-12 http://dx.doi.org/10.1007/978-1-4614-6846-2-12

Clark DM (2013) Evolution of algebraic terms 1: term to term operation continuity. Int J Algebra Comput 23(05):1175–1205 doi:10.1142/S0218196713500227 http://www.worldscientific.com/doi/abs/10.1142/S0218196713500 227, http://www.worldscientific.com/doi/pdf/10.1142/S0218196713500227

Elyasaf A, Hauptman A, Sipper M (2012) Evolutionary design of FreeCell solvers. IEEE Trans Comput Intell AI Games 4:270–281 doi:10.1109/TCIAIG.2012.2210423 http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber=6249736

Epstein SL (1999) Game playing: the next moves. In: AAAI/IAAI, pp 987–993

Forrest S, Nguyen T, Weimer W, Le Goues C (2009) A genetic programming approach to automated software repair. In: Raidl G, Rothlauf F, Squillero G, Drechsler R, Stuetzle T, Birattari M, Congdon CB, Middendorf M, Blum C, Cotta C, Bosman P, Grahl J, Knowles J, Corne D, Beyer HG, Stanley K, Miller JF, van Hemert J, Lenaerts T, Ebner M, Bacardit J, O'Neill M, Di Penta M, Doerr B, Jansen T, Poli R, Alba E (eds) GECCO '09: Proceedings of the 11th annual conference on genetic and evolutionary computation, ACM, Montreal, pp 947–954 doi:10.1145/1569901.1570031 http://citeseerx.ist.psu.edu/viewdoc/summary?doi:10.1.1.147.7 651, best paper

Glazer A, Sipper M (2008) Evolving an automatic defect classification tool. In: Giacobini M, Brabazon A, Cagnoni S, Caro GAD, Drechsler R (eds) Applications of evolutionary computing: proceedings of EvoWorkshops 2008. Lecture notes in computer science, vol 4974. Springer, Heidelberg, pp 194–203

Hauptman A, Sipper M (2005a) Analyzing the intelligence of a genetically programmed chess player. In: Rothlauf F (ed) Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005), Washington, DC. http://www.cs.bham.ac.uk/∼wbl/biblio/gecco2005lbp/papers/21-hauptmann.pdf

Hauptman A, Sipper M (2005b) GP-endchess: using genetic programming to evolve chess endgame players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) Proceedings of the 8th European conference on genetic programming. Lecture notes in computer science, vol 3447. Springer, Lausanne, pp 120–131. doi:10.1007/b107383

Hauptman A, Sipper M (2007a) Emergence of complex strategies in the evolution of chess endgame players. Adv Complex Syst 10(Suppl 1):35–59

Hauptman A, Sipper M (2007b) Evolution of an efficient search algorithm for the mate-in-N problem in chess. In: Ebner M, O'Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) Proceedings of the 10th European conference on genetic programming. Lecture notes in computer science, vol 4445. Springer, Valencia, pp 78–89. doi:10.1007/978-3-540-71605-1-8

Hauptman A, Elyasaf A, Sipper M, Karmon A (2009) GP-rush: using genetic programming to evolve solvers for the Rush Hour puzzle. In: Raidl G, Rothlauf F, Squillero G, Drechsler R, Stuetzle T, Birattari M, Congdon CB, Middendorf M, Blum C, Cotta C, Bosman P, Grahl J, Knowles J, Corne D, Beyer HG, Stanley K, Miller JF, van Hemert J, Lenaerts T, Ebner M, Bacardit J, O'Neill M, Di Penta M, Doerr B, Jansen T, Poli R, Alba E (eds) GECCO '09: Proceedings of the 11th Annual conference on genetic and evolutionary computation, ACM, Montreal, pp 955–962. doi:10.1145/1569901.1570032 http://dl.acm.org/citation.cfm?id=1570032

Koza JR (2008) Human-competitive machine invention by means of genetic programming. Artif Intell Eng Des, Anal Manuf 22(3):185–193. doi:10.1017/S0890060408000127

Koza JR (2010) Human-Competitive Awards: HUMIES. http://www.human-competitive.org. Accessed 6 May 2014

Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2003) Genetic programming IV: routine human-competitive machine intelligence. Kluwer Academic. http://www.springer.com/us/book/9780387250670

Koza JR, Al-Sakran SH, Jones LW (2005) Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits. In: Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on, IEEE, pp 205–212

Letham B, Rudin C, McCormick TH, Madigan D (2012) Building interpretable classifiers with rules using bayesian analysis. Department of Statistics Technical Report tr609, University of Washington

Rudin C, Waltz D, Anderson RN, Boulanger A, Salleb-Aouissi A, Chow M, Dutta H, Gross P, Huang B, Ierome S, Isaac D, Kressner A, Passonneau RJ, Radeva A, Wu L (2012) Machine learning for the New York City power grid. IEEE Trans Pattern Anal Mach Intell 34(2):328–345

Shichel Y, Ziserman E, Sipper M (2005) GP-robocode: using genetic programming to evolve robocode players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds)

Proceedings of the 8th European conference on genetic programming. Lecture notes in computer science, vol 3447. Springer, Lausanne, pp 143–154. doi:10.1007/b107383
Sipper M (2002) Machine nature: the coming age of bio-inspired computing. McGraw-Hill, New York
Sipper M (2006) Attaining human-competitive game playing with genetic programming. In: El Yacoubi S, Chopard B, Bandini S (eds) Proceedings of the 7th international conference on cellular automata, for research and industry, ACRI. Lecture Notes in computer science, vol 4173. Springer, Perpignan, p 13. doi:10.1007/11861201-4, invited Lectures
Sipper M (2011) Evolved to Win. Lulu http://www.lulu.com/. Accessed 6 May 2014
Sipper M, Azaria Y, Hauptman A, Shichel Y (2007) Designing an evolutionary strategizing machine for game playing and beyond. IEEE Trans Syste, Man Cybern, Part C: Appl Rev 37(4):583–593. doi:10.1109/TSMCC.2007.897326
Spector L (2004) Automatic quantum computer programming: a genetic programming approach, genetic programming, vol 7. Kluwer Academic, Boston. http://www.wkap.nl/prod/b/1-4020-7894-3
Spector L, Clark DM, Lindsay I, Barr B, Klein J (2008) Genetic programming for finite algebras. 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, ACM, Atlanta, GA, USA, pp 1291–1298. doi:10.1145/1389095.1389343 http://www.cs.bham.ac.uk/∼wbl/biblio/gecco2008/docs/p1291.pdf
Wagstaff K (2012) Machine learning that matters. arXiv preprint arXiv:12064656
Wang T, Rudin C, Wagner D, Sevieri R (2013) Detecting patterns of crime with series finder. In: Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases (ECMLPKDD 2013)

Karthik Kannappan is a Master's Student in Computer Science at the University of Massachusetts, Amherst, currently working with Prof. Lee Spector at the computational intelligence laboratory. His research interests include automatic software synthesis, affective computing, and software engineering.

Lee Spector is a Professor of Computer Science at Hampshire College and an Adjunct Professor of Computer Science at the University of Massachusetts, Amherst. He received a B.A. in Philosophy from Oberlin College in 1984 and a Ph.D. in Computer Science from the University of Maryland in 1992. He is the Editor-in-Chief of the journal Genetic Programming and Evolvable Machines and a member of the editorial board of Evolutionary Computation. He is also a member of the SIGEVO executive committee and he was named a Fellow of the International Society for Genetic and Evolutionary Computation.

Moshe Sipper is a Professor of Computer Science at Ben-Gurion University, Israel.

Thomas Helmuth is a Ph.D. Candidate in Computer Science at University of Massachusetts, Amherst. He received his B.A. from Hamilton College in 2009 and his M.S. from University of Massachusetts, Amherst in 2012.

William La Cava is a PhD student at University of Massachusetts, Amherst. He received his B.S. and M.Eng. from Cornell University in 2009 and 2010, and went on to work at the National Wind Technology Center in Boulder, Colorado. William is interested in automatic system identification and automatic control design for complex systems. As an NSF student fellow in the UMass IGERT Offshore Wind Energy Program, he is developing symbolic regression methods to create data-based models of offshore wind turbines and avian behavior.

Jake Wisdom was a 2014 graduate of Hampshire College in Amherst MA, where he studied 3D animation, visual effects and computer graphics. He is currently working towards his MFA in visual effects at Savannah College of Art and Design with his primary interests in procedural animation and simulation.

Omri Bernstein is a recent Hampshire College alumnus now working as an instructor at Fullstack Academy of Code in New York City. While at Hampshire, he studied the applications of genetic programming to biology, as well as the applications of biological principles to genetic programming. For his senior thesis he developed an educational quantum computer programming environment, and he is currently working on the evolution of quantum algorithms.

Chapter 10
Tackling the Boolean Multiplexer Function Using a Highly Distributed Genetic Programming System

Hormoz Shahrzad and Babak Hodjat

10.1 Introduction

Evolutionary systems are by nature well suited for parallel processing and distribution (González et al. 2009; Fernndez de Vega et al. 2013; Merelo et al. 2012). The EC-Star platform (Fig. 10.1) is designed to allow massive distribution of Genetic Programming by virtue of a hub and spoke architecture, where the evolutionary engines are distributed (the nodes on the spokes), with the processing being aggregated and coordinated by a server (the hub)(O'Reilly et al. 2013).

EC-Star maximizes efficiency by using an age-layered model that allows partial and incremental fitness evaluations in the evolutionary engines (Merelo et al. 2012; Hodjat and Shahrzad 2013; Jin 2005). Using the multiplexer problem as a benchmark, it has been shown that partial fitness evaluation speeds up evolution and allows generalization (Langdon 2011). The addition of age-layering allows for inclusion of the history of fitness evaluations to be preserved in the genes, and for validation to continue for surviving genes, reducing the need to validate solutions after convergence.

In EC-Star, age is defined as the number of fitness samples a gene has been validated upon. In this system, genes are accepted by the evolutionary coordinator depending on how they fare compared to their age-peers (i.e., other acceptable genes in the same age-layer). This allows for more efficient use of processing power, as genes that are deemed as relatively unfit at an earlier age, need not be evaluated any further. The genes that do survive this filtration are submitted back to evolutionary engines for further evaluation so they can move up the age-layers (see Fig. 10.2).

The evolutionary engines, which use an elitist strategy, make use of genes of various ages as parents to produce new genes for their respective pools. The coordinator

H. Shahrzad (✉) · B. Hodjat
Sentient Technologies Holdings Limited, 160 Spear St. #1600, San Francisco, CA, USA
e-mail: hormoz@sentient.ai

B. Hodjat
e-mail: babak@sentient.ai

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6_10

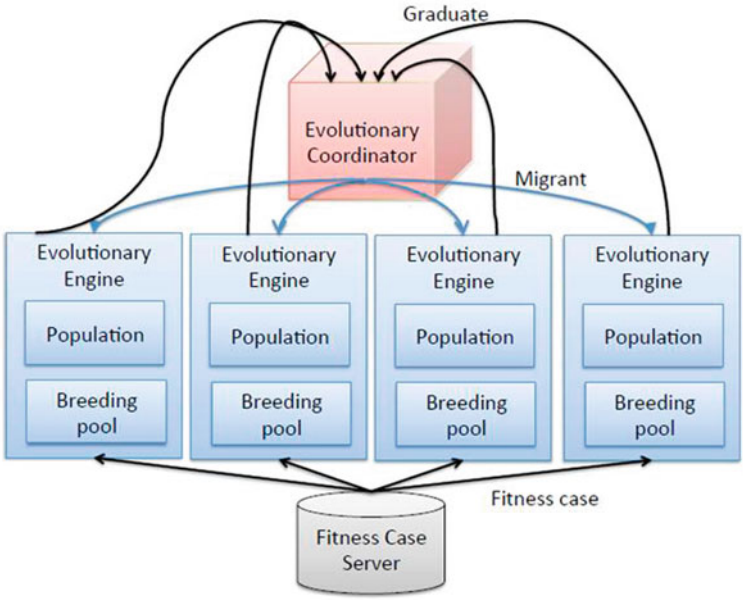


Fig. 10.1 The hub-and-spoke architecture of EC-Star

can send more than one copy of a gene down to different evolutionary engines in this manner. This means that the same gene can be evaluated on more fitness cases in parallel. The coordinator then has the responsibility of merging the results reported by the engines before considering them for insertion into their new age-layers. As a by-product of this give and take between the engines and the coordinator, fitter genetic material is spread through the system.

In this paper, we assess the EC-Star distributed evolutionary computation platform using the well-known multiplexer problem. We start by describing our representation of the problem, which uses EC-Star's default Pitts-style rule-based representation (Bacardit et al. 2008) rather than the LISP S-expression representation typically used in GP systems. Discussion of the 11-multiplexer test runs will follow, demonstrating the effects of partial evaluation, age-layering, rule-based representation, fitness function, and distribution on convergence and consumed processing power. We end the paper with a discussion of the results and proposals for future work.

10.2 Problem Representation

Multiplexer functions have long been identified by researchers as functions that often pose difficulties for machine learning, artificial intelligence, neural nets, and classifier systems (Koza 1990). In general, the input to the Boolean multiplexer function consists of k 'address' bitsA_i, and 2^k 'data' bits D_i, and it is a string of

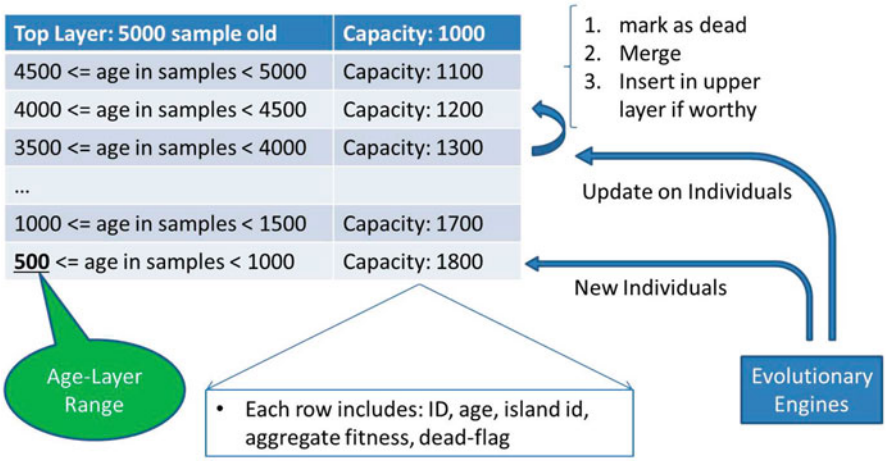


Fig. 10.2 Age-layering in EC-Star

length $k+2^k$ of the form $A_{k-1} \ldots A_i A_0 D_{2^k-1} \ldots D_1 D_0$. The value of the multiplexer function is the value (0 or 1) of the particular data bit that is singled out by the k address bits of the multiplexer. For example, for the 11-multiplexer, where k=3, if the three address bits $A_2 A_1 A_0$ are 110, then the multiplexer singles out data bit number 6 (i.e. D_6) to be its output.

A Boolean function with $k+2^k$ arguments has 2^{k+2^k} rows in its truth table. Thus, the sample space for the Boolean multiplexer is of size 2^{k+2^k}. When k=3, the search space is of size $2^{2^{11}}=2^{2048} \approxeq 10^{616}$. However, since a GP platform can also generate redundant expressions which will be logically equal with some others, the real size of the search space for different representations will vary.

EC-Star allows for pluggable representations. The representation chosen for the purpose of this paper is EC-Star's default representation, which is a Pitts-Style rule based representation, where the genotype consists of a header and body. The header includes such fields such as a unique ID, Age, Master Fitness, which represents the aggregate fitness over samples evaluated so far, etc. The gene body is a rule set with the following grammar:

$$< rules > ::= < rule > | < rule >< rules >$$

$$< rule > ::= < conditions > \rightarrow action$$

$$< conditions > ::= < condition > | < condition > \& < conditions >$$

$$< condition > ::= < predicate > | ! < condition >$$

$$< predicate > ::= comparative\ expression\ on\ a\ feature[lag]$$

$$< lag > ::= \text{positive numeral, index to prior feature values on time series}$$

For the multiplexer problem, we define a single feature we call Bit. The lag index is an index to the multiplexer input bit. N actions (i.e., prediction labels) are defined, each denoting the input value of one of the multiplexer data bits. The rules for the multiplexer problem, therefore, will take the following general form:

$$(<!Bit|Bit>[index] = <0|1>[\&<conditions>]) \rightarrow <data-bit-index>$$

Where index is between 0 and $k+2^k-1$, and data-bit-index is between 0 and 2^k-1. For example, the following rule is mapping the address bits 010 to the data bit 2:

$$Bit[0] = 0 \& Bit[1] = 1 \& !Bit[2] = 1 \rightarrow 2 \tag{10.1}$$

Note that with this definition, although logical OR is not explicitly represented in the grammar, it is conceivable that we can have several rules with the same action. This is equivalent to a logical OR and allows the representation to be functionally complete. In other words, the grammar above, which includes the AND, OR and NOT operators, can be used to express all possible Boolean functions.

This system can produce a range of genes, from only one rule, up to the maximum number of rules allowed per configuration. In the experiments presented in this paper, the max number of rules is set to 256, mainly due to memory constraints. Thus, going back to the 2048 row truth table, if we consider our rules to have all the necessary conditions to single out one row of the truth table, we can have all the possible combinations between 1 to 256 rows which will be:

$$f(2048, 256) = \binom{2048}{0} + \binom{2048}{1} + \ldots + \binom{2048}{255} + \binom{2048}{256}$$

We already know that if we continue the summation all the way to $\binom{2048}{2048}$, the sum will be 2^{2048}, therefore, intuitively, our search space is smaller than this. But even in the case of this representation, the search space is comparable in size. The upper bound estimate for such a function, from Lovsz et al. (2003), is:

$$f(n,k) \leq 2^n exp \frac{(n-2k-2)^2}{4(1+k-n)}$$

(where $k < \frac{n}{2}$)

$$f(2048, 256) \leq 2^{2047} exp \frac{2335156}{-7164} \approxeq 2^{2047} e^{-328} \approxeq 10^{484}$$

However, since we can also have a lower number of conditions per rule, and considering all the other redundancies the system can represent, the real estimate will be larger.

Per Koza's original fitness-function definition in Koza (1990), a gene's aggregate fitness (i.e., master fitness) is incremented every time the value of the data bit its rules suggest match the expected multiplexer's output for the given address bits. For instance, if a gene only had the single rule presented in Eq. (10.1) master fitness on all 2048 fitness samples would end up being 2^8=256. Now, considering the following rule:

$$Bit[0] = 0 \rightarrow 0 \tag{10.2}$$

This does not say anything for all the 1024 samples where the address bit zero has the value of one, and therefore gets a fitness score of zero for those samples, and its overall master fitness adds up to 640.

Unlike S-expressions, in this representation, the evaluation of a rule set on a fitness case can sometimes yield more than one action, and as seen above, sometimes it might not yield any action at all. For example, the rule in Eq. (10.2) does not trigger on all the odd samples, since their Bit[0] is equal to one. The manner by which such outcomes are distilled into the gene's final vote, or what we shall call the *election process*, has interesting implications. Here, for instance, is a short list of several available options when more than one rule fires:

- Ignore them all (i.e., only pick an action if one rule has fired)
- Go with the first one
- Go with the last one
- Pick one of them randomly (note that the result will not be repeatable)
- Go with the first action in the action set
- Go with the last action in the action set
- Go with the most popular action triggered by the majority of rules

10.3 Results

The process to increase confidence in the reported results here was to run each experimental setting ten times. In this paper, based on the context, we have provided either the average of the ten runs or a representative sample. To establish a baseline on the 11-Multiplexer problem, we ran the system with the full set of 2048 evaluation samples using a single client and a pool size, originally suggested by Koza (1990), of 4000 (i.e., no age-layering or distribution). The system converged to the perfect score of 2048 on all ten runs, averaging 28 generations to do so. Therefore, on average, we evaluated a total of 112,000 genes on all 2048 samples before converging.

10.3.1 Tendency to Produce Smaller Solutions

Figure 10.3 shows the maximum fitness over generations along with the total number of conditions for the smallest gene with the corresponding maximum fitness on a

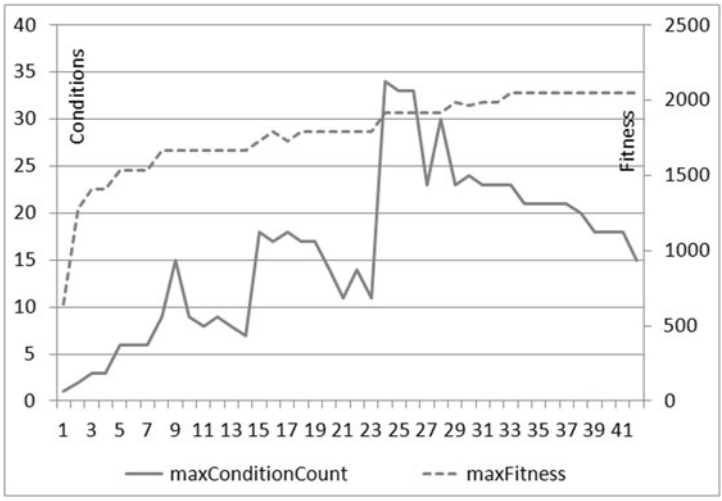


Fig. 10.3 A plot of the fitness of the best gene with least total number of conditions in the pool against that gene's number of conditions

sample single-client run. The system seems to engage in reducing the total number of conditions every time it hits a new fitness plateau. In other words, whenever max fitness stays the same for a few generations, there is a down trend on the total number of conditions.

This may be due to the manner by which EC-Star manages bloat using its Pitts-style representation. In order to avoid propagation of redundant or un-evaluable rules, and to control bloat, EC-Star prevents most cases of tautologies and falsehoods in rules. It also keeps track of the number of times a rule has fired over all the fitness cases and does not allow an inactive rule to be passed on to future generations.

For example, in this case, after 42 generations the gene with the least conditions and optimum fitness only has 15 conditions:

$$(!Bit[2] = 1) \rightarrow 3$$
$$(Bit[0] = 0) \rightarrow 6$$
$$(Bit[1] = 0) \rightarrow 5$$
$$(Bit[0] = 0 \& Bit[2] = 0) \rightarrow 2$$
$$(Bit[2] = 0 \& Bit[1] = 0) \rightarrow 1$$
$$(Bit[1] = 0 \& !Bit[1] = 1 \& !Bit[0] = 1 \& Bit[0] = 0) \rightarrow 4$$
$$(Bit[1] = 0 \& !Bit[2] = 1 \& !Bit[0] = 1) \rightarrow 0$$
$$(!Bit[2] = 0) \rightarrow 7$$

Note that the system is depending on the election logic in order to optimize the rule-set. One simple election logic is to take the action from the first firing rule that

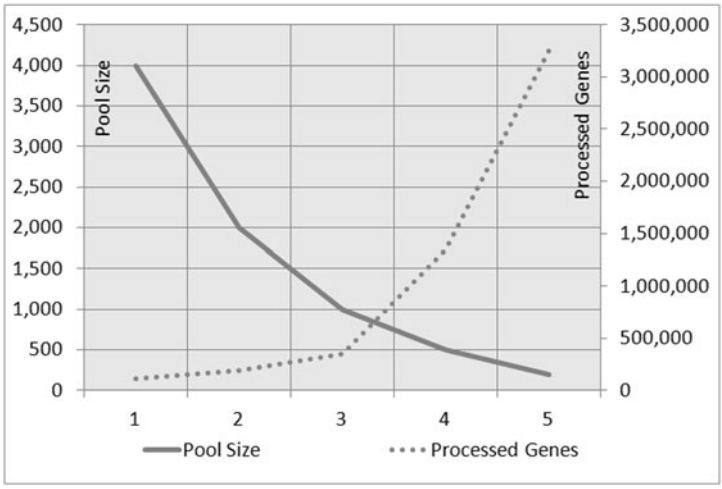


Fig. 10.4 Experiments with decreasing pool size require progressively larger number of genes to be processed before converging to the optimum

fires to be the gene's decision on a fitness sample, and to ignore the outcome of the other rules. Another possible election logic, which is used in these experiments, is to give the precedence to the most certain action. EC-Star allows for pluggable election logic but the default election logic implementation is based on fuzzy logic. Since the rules are not fuzzy here, when more than one action is triggered we need a tie breaker, which in this case is the action order. In the example above, the rules with action 0 are evaluated first, then action 1, then 2, . . . This rule processing order has allowed EC-Star to evolve the rule-set above, which is somewhat analogous to the Hamming code in its optimized brevity.

10.3.2 Pool Size Effect on Processing

We ran experiments with pool sizes of 4000, 2000, 1000, 500 and 200. Figure 10.4 shows that the size of the pool has an inverse relationship with the total number of genes required to be processed before convergence on to the optimum. This seems to hint at the importance of diversity.

10.3.3 Age-Layering Effect on Processing

The age-layer range denotes the age increments at which the genes are considered for inclusion in their respective age-layers. For instance, in a run with an age-layer range

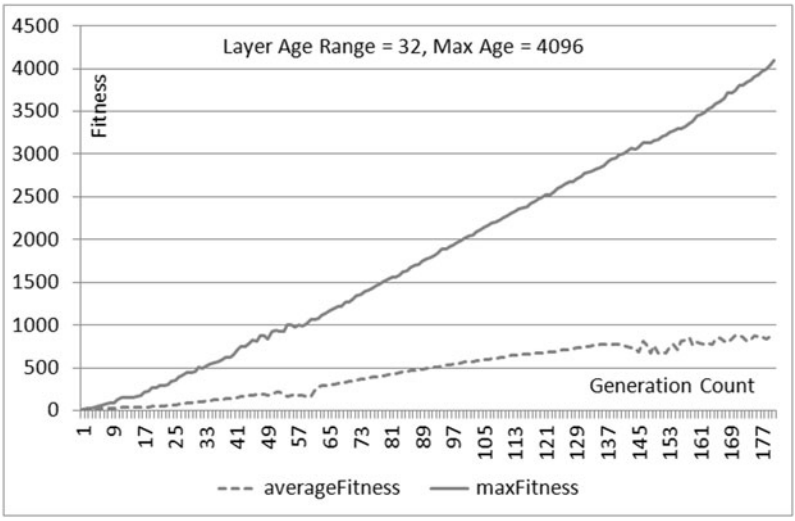


Fig. 10.5 Convergence of an age-layered run with pool size=4000

of 1024, all genes that have been evaluated on 1024 fitness samples are compared, and the best is retained to be aged further to the next age-layer (2048) and so on.

Fitness samples are visited by the genes in random order, and so there is no guarantee that a gene at age 1024 has actually visited 1024 unique fitness samples. In order to increase the chances of getting genes in the top age-layer visit all 2048 unique fitness cases, the top-layer age can be set to be higher than the actual number of fitness cases available. In the experiments here, for example, we set the top-layer age to be double the number of available fitness cases, at 4096.

To test the effect of age-layering, we ran single-client experiments with age-layer ranges of 1024, 512, 256, 128, 64, 32, and 16. All of the runs converged to the optimum 4096 score and we have verified that all top-layer genes with this master fitness score are indeed solutions to the multiplexer problem (i.e., their master fitness is 2048 when run on all unique fitness cases). This observation was in agreement with Langdon (2011), which has shown that even in lower layers a gene with a perfect score will be a complete solution to the multiplexer problem. Indeed, we observed this effect down to ages as low as 64.

As an example, the run in Fig. 10.5 converges in 177 generations over a pool of 4000 genes, so it computes over 700,000 genes only on 32 samples, which is equivalent to the processing of $\approxeq 11,000$ genes over the whole 2048 member sample space. This is roughly one tenth of the processing used to converge the baseline in Sect. 3.1.

Figure 10.6 shows a sample distribution of the genes in a converged pool. We can clearly distinguish two clusters, representing the stream of new random genes and the rest of the evolved population. This shows that the convergence was not due to random luck and that a healthy subset of the population harbors traits that result in higher fitnesses.

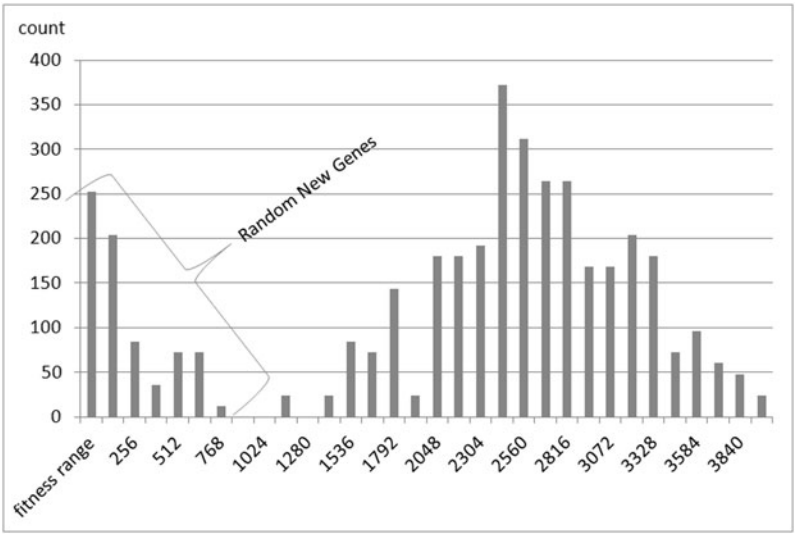


Fig. 10.6 A sample snapshot of fitness distribution in a converged pool

Cutting the age range to half consistently improved the efficiency of the run. This rate held up until we ran experiments with an age-layer range of 16, which took more processing power compared to our base-line to converge. On a single client, setting the age-layer range to 8 did not converge within a reasonable amount of time.

10.3.4 Distribution Effect on Processing

In order to test the distribution effect, we set the age-layer range to be 16, and to keep the diversity constant, we distributed it over 8 Evolutionary Engines with the pool size of each engine set to 500 to sum up to the same 4000 pool size as the single run discussed above. As illustrated in Fig. 10.7, the distributed run converged with less than half of the power we needed for the single age-layered run.

Running the same distribution experiment with age-layer range set to 8 converges to optimum, taking even less processing power than the distributed run with age-layer range of 16.

10.3.5 Effect of Fitness Function on Processing

As mentioned, unlike the common S-expression representation where there is always an output of zero or one, the EC-Star's Pitts-style representation means that for some genes, there are fitness cases on which no rules fire. So far, using Koza's fitness

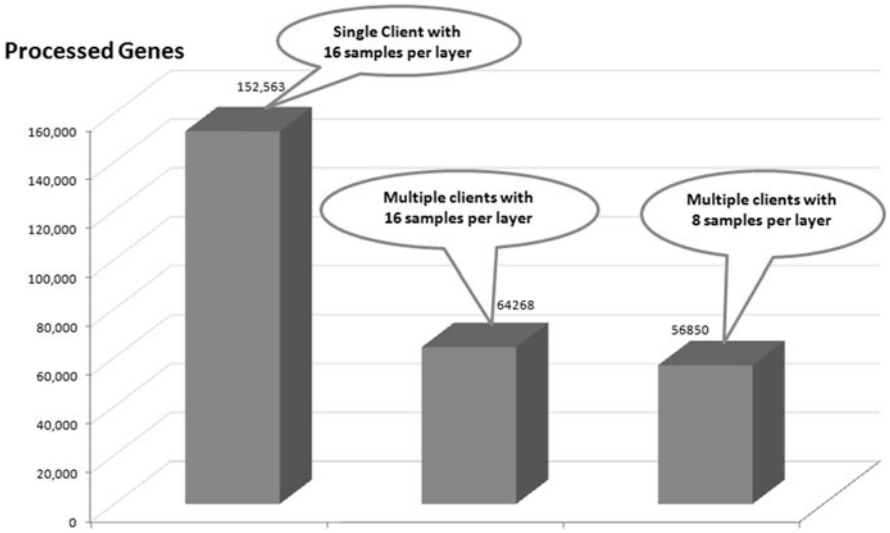


Fig. 10.7 Total processed genes before convergence for single-client runs compared to distributed runs with 8 clients

function (Koza 1990), we have not differentiated between a no output' and a wrong output' in calculating the fitness.

To emphasize that choosing a fitness function should not be taken for granted, in the next set of experiments, we tried a new tri-value fitness function: increment the fitness for a correct output, subtract two for an incorrect output, and zero if no rules fire for the gene over the fitness sample. For example, while the master fitness for rule in Eq. (10.1) will not change when using this new fitness function, the new scheme will result in a master fitness of -128 for the rule in Eq. (10.2). We conducted several experiments with different fitness schemes to come up with the above mentioned fitness function, however a full analysis of the broad range of fitness functions which could be used and why they work is beyond the scope of this work.

As seen in Fig. 10.4, the single client tests requiring the most processed genes was the run with a pool size of 200, which evaluated over 3,250,000 genes to converge. Changing the fitness function makes a significant difference, as seen in Fig. 10.8, only requiring $\approxeq$ 160,000 genes to be evaluated in total before converging to the optimum.

10.3.6 Scaling to Larger Problems

Higher order multiplexer problems have been tackled before using indirect methods or by using hierarchical boosting-like knowledge extraction from building blocks of

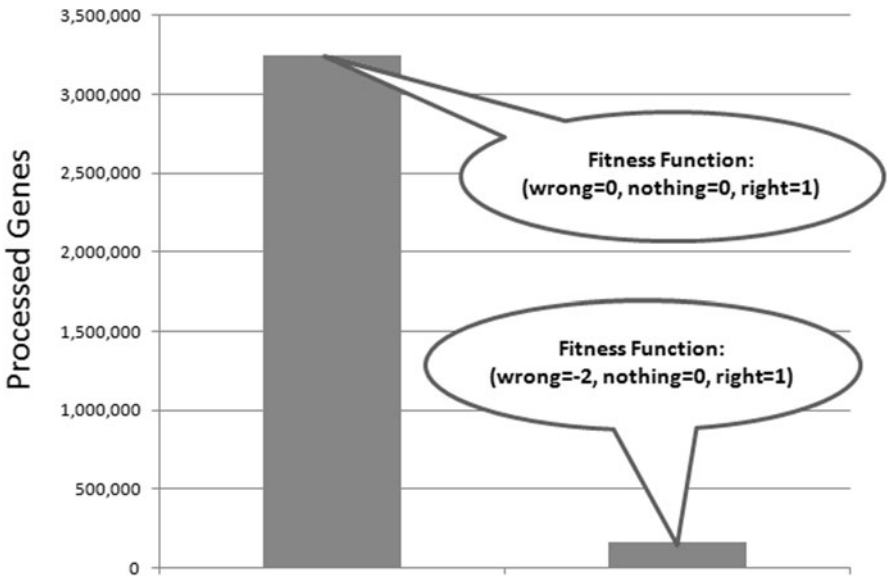


Fig. 10.8 Using a fitness function that is more appropriate for the Pitts-style representation yields significant improvement in total number of genes processed before converging to optimum

smaller order runs (Iqbal et al. 2013). Here we used the distributed age-layered EC-Star system from the experiments above to run higher order multiplexer problems in a direct manner, without making use of the above mentioned techniques.

Using an age-layer range of 4096 and a top-layer age set at $1048576{=}2^{20}$, the 20-Multiplexer problem was solved in less than 4 h by running on 56 clients (seven 8-core machines running an evolutionary engine per core). These runs, on average, only needed to process around 500,000 gene evaluations over 4096 samples, which is approximately equivalent to processing less than 2000 genes over the whole sample space of 2^{20}.

We used the same configuration to solve the 37-Multiplexer problem, again with an age-layer range of 4096 and the top-layer age set at 2^{20} (instead of 2^{37}). These runs converged in less than one week, processing an average of 3 million genes over 4096 samples, which is less than the equivalent processing of 400 genes over the whole 2^{37} samples. All genes with a perfect score at age 2^{20} were also verified and determined to be correct solutions.

An interesting future work would be to compare the overall time and processing for different distribution settings.

10.4 Conclusions

The experiments in this paper show that when solving the multiplexer problem:

- Less processing power is needed when using partial evaluations. Lucky' genes were not observed in any of the runs, in spite of partial evaluations and the possibility of genes not having seen every possible fitness sample.
- Runs scale with distribution
- The Pitts-style representation and its implementation in EC-Star seems to push towards smaller solutions

The system described in this paper can and is being applied to problems where the determination of the fitness of a gene is made incrementally by virtue of running the gene on fitness samples. In the multiplexer problem, the Evolutionary Engines are able to generate the fitness samples automatically, simply by using a random number generator. In classifier problems, on the other hand, real world data samples may need to be distributed to the evolutionary engines as fitness samples.

In one application of EC-Star, the fitness samples are time series data from ICU logs of patient blood pressure readings (Hemberg et al. 2013). The classifier genes run their rules on the data sample in order to predict the blood pressure level for the sample labelled based on the level observed after a thirty minute black-out period. This problem was run on an average of 3000 evolutionary engines over the course of a month.

EC-Star has also been used successfully to evolve financial trading strategies, using historical stock data time series as the fitness samples. Typical runs make use of hundreds of thousands of evolutionary nodes, over periods of months, continuously yielding diversified and improved solutions.

For future work, we plan to test the effect of using a co-evolutionary island model in solving the multiplexer problem. This is a feature of EC-Star that allows segregated evolution of islands' of evolutionary engines, with each island exclusively running on a subset of the global feature/action set. The convergence rate of the islands is monitored by the coordinator and, once at least two islands converge sufficiently, they are merged. Merging means allowing cross breeding of genes between the evolutionary engines, using the union of the parent islands' feature/action sets.

Acknowledgements The authors wish to thank Sentient Technologies Holdings Limited for sponsoring this research and providing the processing capacity required for some of the experiments presented in this paper.

References

Bacardit J, Bernadó-Mansilla E, Butz MV (2008) Learning classifier systems: looking back and glimpsing ahead. In Learning Classifier Systems (pp. 1–21). Springer Berlin Heidelberg

Fernndez de Vega F, Olague G, Trujillo L, Lombraa Gonzlez D (2013) Customizable execution environments for evolutionary computation using boinc + virtualization. Nat Comput 12(2):163–177. doi:10.1007/s11047-012-9343-8. http://dx.doi.org/10.1007/s11047-012-9343-8

González DL, de Vega FF, Trujillo L, Olague G, Araujo L, Castillo P, Merelo JJ, Sharman K (2009) Increasing gp computing power for free via desktop grid computing and virtualization. Parallel,

distributed and network-based processing, 2009 17th Euromicro International Conference on, IEEE, pp 419–423

Hemberg E, Veeramachaneni K, Dernoncourt F, Wagy M, O'Reilly UM (2013) Imprecise selection and fitness approximation in a large-scale evolutionary rule based system for blood pressure prediction. Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion, ACM, pp 153–154

Hodjat B, Shahrzad H (2013) Introducing an age-varying fitness estimation function. In Genetic Programming Theory and Practice X (pp. 59–71). Springer New York

Iqbal M, Browne WN, Zhang M (2013) Learning complex, overlapping and niche imbalance boolean problems using xcs-based classifier systems. Evol Intell 6(2):73–91

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft comput 9(1):3–12

Koza JR (1990) A hierarchical approach to learning the boolean multiplexer function. Foundations of genetic algorithms, 171–192

Langdon WB (2011) Generalisation in genetic programming. Proceedings of the 13th annual conference companion on genetic and evolutionary computation, ACM, pp 205–206

Lovsz L, Pelikn J, Vesztergombi K (2003) Discrete mathematics: elementary and beyond. Springer, Berlin

Merelo JJ, Mora AM, Fernandes CM, Esparcia-Alcazar AI, Laredo JLJ (2012) Pool vs. island based evolutionary algorithms: an initial exploration. P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on, IEEE, pp 19–24

O'Reilly UM, Wagy M, Hodjat B (2013) Ec-star: A massive-scale, hub and spoke, distributed genetic programming system. In Genetic Programming Theory and Practice X (pp. 73–85). Springer New York

Hormoz Shahrzad is Principal Scientist of Sentient Technologies Holdings Limited, responsible for the core technology of a massively distributed evolutionary system applied to various domains, including stock trading. Hormoz has been active in the Artificial Life and Artificial Intelligence field for more than twenty years.

Babak Hodjat is Chief Scientist and co-founder of Sentient Technologies Holdings Limited, responsible for the core technology behind the world's largest distributed evolutionary system. Babak is an entrepreneur having started a number of Silicon Valley companies as main inventor and technologist. He was also Senior Director of Engineering at Sybase Anywhere from 2004 to 2008, where he led Mobile Solutions Engineering including the AvantGo Platform, and the mBusiness Anywhere and Answers Anywhere product suites. Previously, Babak was the co-founder, CTO and Board member of Dejima Inc. acquired by Sybase in April 2004. Babak is the primary inventor of Dejima's patented agent-oriented technology applied to intelligent interfaces for mobile and enterprise computing—the technology behind Apple's Siri. Dejima was one of only four private firms enrolled in the DARPA (Defense Advanced Research Projects Agency) funded Cognitive Assistant that Learns and Observes (CALO) Project, managed by SRI International and one of the largest AI projects ever funded. Babak served as the acting CEO of Dejima for 9 months from October 2000. In his past experience, he led several large computer networking and machine learning projects at Neda, Inc. Babak received his PhD in Machine Intelligence from Kyushu University, in Fukuoka, Japan.

Index

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XII,*
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-16030-6