Numerical Optimization as a Means to Symbolic Regression Program Synthesis

BY

BRIAN MATTHEW CERNY
B.S., Elmhurst College, Elmhurst, Illinois, 2006

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2011

Chicago, Illinois

ACKNOWLEDGMENTS

Before continuing, I would like to express my sincerest and deepest thanks to my advisor, Dr. Peter C. Nelson. Without his unconditional support and almost infinite patience, the work presented in this thesis would never have culminated. Amongst his countless responsibilities and appointments, first as the Head of the Computer Science department and then later on as the Dean of the College of Engineering, Dr. Nelson always made time for me and my research. During my tenure with Dr. Nelson, his concern for me as an individual was readily apparent and genuine. The guidance and words of wisdom offered by him, regarding not only school but also life, are still greatly appreciated to this very day.

In addition, any thanks would be incomplete without mentioning Dr. Chi Zhou or Dr. Weimin Xiao. Both of these individuals continually inspired me with their intense passion for knowledge and their devotion to quality research. I have fond memories of our many very enjoyable discussions. The insights, ideas, and suggestions they shared therein were invaluable to this thesis. Through their example, they helped me grow as a researcher and a scientist. It was truly a pleasure working with these two individuals and I am very thankful for the generous funding they were able to secure from the Physical Digital Realization Research Center at Motorola Labs.

Last but not least, I would like to thank both Dr. Robert H. Sloan and Dr. Bing Liu for taking time from their busy schedules to serve on my thesis committee. While it was immensely challenging to me personally, Dr. Sloan's course on computability and complexity theory was intellectually stimulating and made my following algorithmically oriented courses a breeze. Dr. Sloan's excellent teaching skills and clear love of the material always made me look forward to his many enjoyable lectures. Dr. Liu is also another outstanding individual with whom I had the pleasure and opportunity to study under. His course in data and text mining was one of my favorite courses and it was quite beneficial to me to observe the evolution of his research firsthand.

Springer and the Association for Computing Machinery should also be thanked for granting permission to include previously published works in the third[1] and fourth[2] chapters of this thesis, respectively.

BMC

[1]With kind permission from Springer Science + Business Media: Brian Cerny, Chi Zhou, Weimin Xiao, and Peter Nelson. Probabilistically Guided Prefix Gene Expression Programming. In Natalio Krasnogor, Giuseppe Nicosia, Mario Pavone, and David Pelta, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*, pages 15–26. Springer Berlin / Heidelberg, 2008a. URL http://dx.doi.org/10.1007/978-3-540-78987-1_2 © Springer-Verlag Berlin Heidelberg 2008

[2]© ACM, 2008. This is a minor revision of the work published in Brian M. Cerny, Peter C. Nelson, and Chi Zhou. Using Differential Evolution for Symbolic Regression and Numerical Constant Creation. In Maarten Keijzer, editor, *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1195–1202, New York, NY, USA, July 2008b. ACM. ISBN 978-1-60558-130-9. URL http://doi.acm.org/10.1145/1389095.1389331

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

AA	Affine Arithmetic
AP	Analytic Programming
BN	Bayesian Network
BW	Baum-Welch
CAS	Computer Algebra System
CC	Constant Creation
CRN	Common Random Number
DE	Differential Evolution
DE-PGEP	Differentially Evolved Prefix Gene Expression Programming
DM	Differential Mutation
DOE	Design of Experiments
DP	Dynamic Programming
DTL	Decision Tree Learning
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EDA	Estimation of Distribution Algorithm
EDP	Estimation of Distribution Programming
EM	Expectation-Maximization
ENES	Expected Number of Evaluations per Success

LIST OF ABBREVIATIONS (continued)

ERC	Ephemeral Random Constant
ET	Expression Tree
FBA	Forward-Backward Algorithm
GA	Genetic Algorithm
GDE	Grammatical Differential Evolution
GDE*	Geometrical Differential Evolution
GE	Grammatical Evolution
GEP	Gene Expression Programming
GP	Genetic Programming
HMM	Hidden Markov Model
IA	Interval Arithmetic
LHS	Latin Hypercube Sampling
LS	Linear Scaling
ML	Machine Learning
MLS	Multiple Linear Scaling
MSE	Mean Squared Error
NRMSE	Normalized Root Mean Squared Error
NSM	No Same Mates
OCIP	Ocean Color Inverse Problem
PG-PGEP	Probabilistically Guided Prefix Gene Expression Programming
PGEP	Prefix Gene Expression Programming

LIST OF ABBREVIATIONS (continued)

PSO	Particle Swarm Optimization
RMSE	Root Mean Squared Error
RNG	Random Number Gene
RS	Ratio of Success
SFI	Symbolic Function Identification
SR	Symbolic Regression
TSP	Time Series Prediction
TSR	Trustable Symbolic Regression

SUMMARY

Over the years there has been an increasing interest in probabilistically oriented Evolutionary Algorithms (EAs), but it has not been until recently that these innovative methods have been collectively recognized and achieved an independent status. By eliminating the traditionally employed genetic operators, these probabilistic EAs have been forced to adopt an alternative approach, and in the case of Estimation of Distribution Algorithms (EDAs), probabilistic graphical models have become the favored substitute. In the first major contribution of thesis, the proposal is made to utilize a probabilistic model that has been previously overlooked in the EDA literature, namely Hidden Markov Models (HMMs). But preferring not to completely abandon the biologically inspired genetic operations, the classical learning algorithms used to train HMMs are largely ignored, and instead, Differential Evolution (DE) is used to evolve the underlying constrained numerical parameters of the chosen probabilistic model. The evolved HMMs are then used to generate likely Prefix Gene Expression Programming (PGEP) chromosomes which encode candidate solutions, and thus provide feedback to guide this proposed evolutionary search process. After the algorithm is described in detail, benchmarking results on a set of Symbolic Regression (SR) problems are reported and this novel approach is compared to the original PGEP method.

Besides investigating probabilistic model driven representations for SR, one other problem that has plagued Genetic Programming (GP) and its derivatives like PGEP, is also investigated herein. That being, numerical Constant Creation (CC) in SR. Given a mathematical formula expressed as a tree structure, the leaf nodes are either variables or constants. Such constants are usually unknown in SR problems, and GP, as well as many of its derivatives, lack the ability to precisely approximate these values. This is due to the inherently discrete encoding of GP-like methods which are more suited for combinatorial searches than real-valued optimization tasks. Previously, several attempts have been made to resolve this issue, and the dominant solutions have been to either embed a real-valued local optimizer or to develop additional numerically oriented genetic operators. In the second and final major contribution of this thesis, an entirely new and unified approach to SR with CC is proposed. Again, through the adoption of the robust real-valued optimization algorithm known as DE, constants and PGEP programs will be simultaneously evolved in such a way that the values of the leaf nodes will be approximated as the tree structure is itself changing. Experimental results from several SR benchmarks are presented and analyzed. The results demonstrate the feasibility of the proposed algorithm and suggest that exotic or computationally expensive methods are not necessary for successful CC.

1. INTRODUCTION

1.1 Symbolic Regression

1.2 Overview

Given a set of numerical inputs and the corresponding outputs, one frequently wants to find a mathematical relationship that accurately models the provided data. This is the task of regression in general, but Symbolic Regression (SR), which may also be referred to as Symbolic Function Identification (SFI), stands in stark contrast to conventional regression analysis. It extends the idea of regression to accommodate the estimation of a symbolic model. This type of regression is distinct from classical parametric regression, which is only concerned with the estimation of a fixed number of coefficients or parameters.

Moreover, Symbolic Regression (SR) may then be considered a non-parametric regression technique, but it is of the most generalized form. That is, SR does not assume any particular functional form (e.g., linear or non-linear) or fixed model structure (e.g., order). Further distinguishing SR from other non-parametric techniques is that it is derivative free and does not rely on smoothing or kernels in any way. Therefore, in summary, the goal of SR is to regressively estimate a symbolic model that fits the given data and models the unknown generative process under few assumptions.

Some of the oldest and most widely used methods meant to discover SR models belong to the paradigm of Evolutionary Algorithms (EAs). Due to several unique characteristics and its almost ubiquitous association with SR, Genetic Programming (GP) (Koza 1992) is probably the most popular technique used to solve this interesting computational task. While several competing non-evolutionary SR algorithms were introduced around the same time (Langley et al. 1983; Barzdins and Barzdins 1993), they were primarily driven by heuristics, which made them poorly suited for discovering innovative, novel, or insightful solutions. This also severely limited these earlier alternative techniques to well established or highly understood domains. Thus, it has been within the broad class of meta-heuristical algorithms where almost exclusively, all advancements in the area of SR have occurred over the past two decades since GP was first introduced and advocated by John Koza. Virtually all meta-heuristics for SR are inspired by biological organisms, social dynamics, evolutionary processes, or other naturally occurring phenomena (Koza 1992; Ryan et al. 1998; Ferreira 2001; Li et al. 2005; Spector 2001; Johnson 2003; Green et al. 2004).

Accordingly, any future reference to or comment about SR will be made within the broad context of meta-heuristics, and should be applicable to most, if not all, variants of GP unless explicitly stated otherwise. Nevertheless, before discussing any particular approach to SR in greater detail, a more thorough understanding of and appreciation for SR is warranted. A brief but comprehensive and up-to-date survey of SR now follows.

1.2.1 Model Interpretability, Understandability, and Transparency

The ability to accurately model data is obviously useful, but SR does offer an interesting combination of advantages that makes it uniquely attractive. One benefit in particular is that of human interpretability and understandability (Kotanchek et al. 2007). Although the same observations could be modeled using a non-SR technique, the fitted solutions are, from a human perspective, potentially incomprehensible and untrustworthy “black boxes”. On the contrary, expressions discovered by any SR compliant algorithm are more like “white boxes”, which are completely transparent and open. This permits models to be interpreted and understood, allowing for insights into a previously enigmatic process to be more easily gained. With the exception of Decision Tree Learning (DTL) (Looks 2005; Liu 2007), which actually only deals with classification problems and is thus not as generally applicable as SR (Kotanchek et al. 2007), few other Machine Learning (ML) algorithms can advertise such capabilities.

More specifically, insights are revealed through the size, shape, composition, and contexts of the structured solution. For example, in the case of a multivariate process, each input may be either completely ignored or freely reused. Conveniently, this permits for an automatic reduction in dimensionality through the pruning of noisy, irrelevant, or confounding inputs (Langdon and Buxton 2004; Poli et al. 2008). After determining the significant input variables, more succinct[1], comprehensible, and authentic solutions can be found with SR. A modeled solution will not only expose the important inputs, but also reveal their contribution to or influence on the sole output. Furthermore, interactions and relationships between the different inputs are captured within this same structure.

However, just because a model can be mechanically understood, does not necessarily mean that an explanation to the underlying phenomenon will be readily apparent. Initially, a model

[1]It is entirely possible that contrived solutions will be significantly more complex and overly bloated in canonical GP unless known enhancements are employed to combat and reduce bloat.

may not make sense or even be intuitive due the unorthodox practices employed by many of the different SR algorithms. Tools like a Computer Algebra System (CAS) (Cohen 2002) can make the analysis of evolved formulae easier and more convenient (Poli et al. 2008). The formulae can be automatically simplified into some standardized form, manipulated into more compact expressions, or rewritten using algebraic and trigonometric identifies. Substitutions may also be used when repetitive or frequent terms appear throughout the expression and a more readable formulation is desired. Furthermore, SR models may be subjected to common statistical analyses or visualization techniques. It should now be evident that an SR model is immensely flexible, and its universal form allows it to be easily combined with existing techniques that are already widely available and universally accepted.

1.2.2 Human Intervention and Domain Knowledge

As mentioned above, a model which fits the data is discovered by the chosen technique. Typically, this is accomplished with a moderate or even a limited amount of knowledge provided by an external supervisor. Not surprisingly, this makes SR very useful when a problem is initially poorly understood, certain assumptions are violated or cannot be confidently made, exploratory research is being undertaken, certain doubts about conventional wisdom exist, or details concerning a solution's structural properties are scarce or even nonexistent. In SR, the supervisor is generally relieved from these perplexing issues, and the determination is done in an unbiased and inductive fashion. So, when compared to more conventional forms of regression analysis, SR can automatically infer an authentic model, including both the structure and parameters, from the data alone. Thus, the data driven paradigm of SR can reduce or even eliminate the possibility of accidental errors, mistaken prejudices, or empirical guesses.

Although most SR algorithms have parameters that can be fine-tuned, well established conventions exist to significantly reduce the effort needed to select good parameters, e.g., Design of Experiments (DOE) (Bartz-Beielstein 2006). Moreover, various enhancements have tried to automate parameter selection by means of embedded adaptive processes (Angeline 1996; Vafaee et al. 2007). While such parameters may be problem dependent, the parameters do not necessarily require additional information about the problem itself. These parameters are more concerned with how the SR algorithm traverses the infinitely large landscape of all possible solutions.

Fully transparent solutions also allow for the seamless integration of domain specific knowledge provided by an expert. This can be performed either *a priori* or *a posteriori*. In the former case, the knowledge provided by an expert must first be manually converted into the form of a seed (Poli et al. 2008). Then, the SR system may iteratively refine the seed through its progeny or share its genetic material with its descendants. If deemed useful, the material will survive and prosper, otherwise it will quickly become rare and eventually extinct. Alternatively, in the latter case, the domain specific knowledge is integrated into the model by augmenting it with any relevant information that could improve its prediction abilities. Both cases assume some knowledge exists, is readily available, and can be converted into a compatible form. However, not all SR practitioners have this luxury.

1.2.3 Model Ensembles

Since virtually all SR algorithms are actually random stochastic processes and thus non-deterministic, multiple plausible solutions to a single problem will be discovered. Usually after enough repeated attempts, a good enough model will be discovered. However, when more difficult problems are encountered, individual models may perform less than satisfactory on average. While individually, these solutions may be unacceptable, the models can be combined together into an ensemble (Kotanchek et al. 2007). As the whole is greater than its parts, higher confidence and more accurate predictions can be achieved with an ensemble of diverse independent models.

Hypothetically, certain individual models may perform poorly over particular regions of the input space. A piecewise combination of the models might be appropriate where each model is responsible for the region in which it performs the best. Alternatively, a consensus among the models might be preferred. In the case of SR, the consensus could be based on some measure (e.g., standard deviation or inter-quartile range) and an aggregated value (e.g., mean, median or mode) (Kotanchek 2010). Now, through the use of an ensemble, large deviations from the middle can be easily detected with outliers being identified and suppressed.

Alternatively, instead of splitting up the input space after evolution, the input space may be divided into subsets before evolution (Kotanchek et al. 2007). After the models have been evolved, the best model for each region is responsible for estimating unseen points. Moreover, different functional building blocks (e.g., variables and functions) can be used to increase the diversity of the ensemble (Kotanchek et al. 2007). This means that various models with different degrees of linearity and levels of complexity can be included in the ensemble as well. Overall, an ensemble can achieve greater consistency, promote mutual reinforcement, improve accuracy, and prevent explosive failures.

1.2.4 Summary

As it should now be apparent, the paradigm of SR is accompanied by some loss of control. This can be hard to accept and somewhat troubling at first. But as demonstrated, it can alleviate the supervisor from considerable responsibility and even be applied in situations were classical or even more modern regression techniques are not appropriate. While SR bestows transparency and flexibility, it alone does not guarantee generalization and extrapolation, assure regularity or reproducibility, prove correctness, or promise optimality.

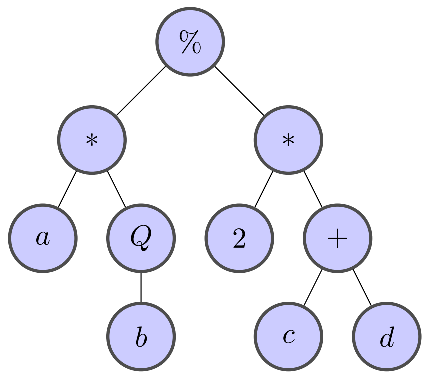
The trustworthiness of SR models has therefore received much attention because untrustworthy models stifle greater adoption of SR and cast doubt on the practical usefulness of the models. Trustable Symbolic Regression (TSR) (Kotanchek et al. 2007) attempts to address some of these issues by building on earlier works like (Sánchez 2000; Keijzer 2003; Valigiani et al. 2004). The aforementioned research increases the robustness and confidence of SR when uncertainty is involved. Unfortunately, the adoption of such improvements appears to be slow and limited at best.

2. BACKGROUND

2.1 Prefix Gene Expression Programming

Prefix Gene Expression Programming (PGEP) is a recently devised EA algorithm which although extremely simple in structure and function, provides for an efficient yet powerful approach to the synthesis of computer programs. Applied to areas such as Symbolic Regression (SR) (Li 2006), text summarization (Xie et al. 2004), and classification rule mining (Zhou et al. 2003), Prefix Gene Expression Programming (PGEP) has been reported to outperform many traditional ML techniques and other existing EAs. Borrowing the fixed-length linear encoding scheme from GAs and adopting the ramified non-linear tree structures of GP, PGEP has successfully separated the genotype from the phenotype through a static process of ontogeny. This precise translation from the linear genotype, or chromosome, to a hierarchical realization of the phenotype, or Expression Tree (ET), permits PGEP to maintain the advantages of an easily modifiable and unconstrained autonomous genome, while reaping the benefits of adaptable structures that allow for sophisticated behavior (Ferreira 2006).

An example of a linear PGEP chromosome with a fixed-length of 14 can be seen in Figure 1C and the corresponding encoded mathematical expression is visible in Figure 1B. Each chromosome is composed of uniquely indexed elements called genes, which belong to the gene



(A) Expression Tree

$$\frac{a\sqrt{b}}{2\,(c+d)}$$

(B) Encoded expression

$$\underbrace{\% \;\; \underbrace{* \;\; a \;\; \underbrace{Q \;\; b}_{3}}_{2} \;\; \underbrace{* \;\; 2 \;\; \underbrace{+ \;\; c \;\; d}_{5}}_{4}}_{1} \;\; \underbrace{- \;\; 1 \;\; b \;\; 3}_{\text{junk}}$$

(C) Linear chromosome

Figure 1. Shows a simple example of a PGEP computer program structured as an ET, the linearized chromosome, and the encoded mathematical expression.

set G. In the case of Figure 1, where Q[1] denotes the square root and %[2] represents division, $G = \{+, -, *, \%, Q, 1, 2, 3, a, b, c, d\}$. Typically G can be divided into two disjoint subsets, which in the case of Figure 1 are the terminal set $T = \{1, 2, 3, a, b, c, d\}$ and the function set $F = \{+, -, *, \%, Q\}$. As just demonstrated, T usually consists of the input variables (attributes) and selected constants, where F contains all the functional operators with an arity greater than

[1]In most works, including this one, the square root function is defined as simply taking the absolute value of the sole argument to extend the domain of the function to the negative numbers without introducing complex numbers.

[2]The protected division function guards against divisions by zero by returning the worst possible fitness. The pathological behavior of the discontinuity at zero is also commonly handled by returning a reasonable arbitrary value like zero or one. However, it was empirically shown in (Sprogar 2008) that such arbitrary values promote bloat and more peculiarly structured models. More sophisticated and mathematically appealing techniques like Interval Arithmetic (IA) (Keijzer 2003) and Affine Arithmetic (AA) (Pennachin et al. 2010) can be used to detect such pathologies during evolution over the continuous input space of the model and not just at discrete points.

or equal to one. It can also be seen in Figure 1 that the encoded expression naturally terminates within the provided bounds. This is the preferred behavior and only results in "junk" or superfluous genes that are ultimately harmless and thus safely ignored.

Through the adoption of a prefix notation encoding scheme, PGEP allows for the seamless and unambiguous translation between the chromosome and the ET (Li 2006). The chromosome in Figure 1C may then be converted to the ET in Figure 1A by iterating over genes of the chromosome and filling out the ET in a depth-first fashion. During this translation process, the tree continually grows by branching out according to the arity of the encountered genes. Thus, when any node of the terminal set is encountered it naturally terminates the appropriate branch of the ET, resulting in trees of various sizes, shapes, and complexities. The inverse translation is accomplished by performing a standard pre-order traversal of the ET, and the order in which a gene is visited determines its position in the linear chromosome.

For those familiar with the original Gene Expression Programming (GEP) algorithm (Ferreira 2006), the most noteworthy difference between GEP and PGEP is the chosen encoding scheme which dictates the organizational structure of linear chromosomes. In GEP, an alternate scheme termed Karva notation is used and it necessitates a breadth-first style expansion of a linear chromosome into an ET (Ferreira 2006). As a result, GEP chromosomes are composed of two distinct parts, namely a head and a tail. Any gene may appear in the head, but only terminal genes are permitted in the tail. Unlike GEP, PGEP obviously makes no distinction between a head and a tail, and simply requires that the encoded expression naturally terminates. By performing a simple arity check, incomplete expressions are detected and invalid chromosomes are identified (Li 2006). Once identified, an invalid chromosome is replaced with a newly generated chromosomal offspring, which is in turn subjected to the same arity test. Additional reproductions are carried out until a valid chromosome is born.

One additional consequence of the prefix notation encoding scheme is that the gene proximity and hierarchy present in the expression tree is maintained by the linear PGEP chromosome (Li 2006). This phenomenon is apparent in the relationship between the internal nodes of Figure 1A and the braces in Figure 1C. That is, when the chromosome is expanded, the number underneath a brace indicates the order in which the root of the corresponding non-trivial sub-tree was visited. Compared to GEP, which adopts a layered approach and scatters genes throughout the head and tail of a chromosome, PGEP preserves the branches as a whole and is thus easier to interpret in its linearized form. PGEP is then considered to have a more resilient encoding as disruptive linear operations are yielded less destructive. To gain an even more thorough understanding of the PGEP algorithm and its purported benefits, the interested reader may wish to consult the authority on PGEP, mainly (Li 2006).

2.2 Constant Creation

2.2.1 Motivation

Probably one of the most widely used applications of GP and its extensions like PGEP, has been to that of SR. Given a training set consisting of several cases of inputs and the corresponding output, the goal is to construct a symbolic expression for an unknown mathematical function which adequately models the training data. PGEP attempts to solve such problems by automatically synthesizing a computer program using an evolutionary inspired approach, e.g., survival of fittest and natural selection. Originally identified by Koza (Koza 1992) as a weakness of GP, the inability to discover good numerical constants has in the past prevented GP from being successfully applied to more complex, real-world SR problems. To demonstrate, recall the trivial example in Figure 1. Unless it was known in advance that the two was needed, GP and PGEP would both also have to evolve an approximate value to this constant. Without explicitly including constants in the gene set, this particular constant might be synthesized with an expression like $(x \div x) + (x \div x)$. While this is not a very exciting case, real values of higher precision like π or any other scientific constant are more difficult to evolve without explicit Constant Creation (CC) mechanisms.

The first real attempt to deal with the problem of CC was appropriately presented in (Koza 1992). Named the Ephemeral Random Constant (ERC) and denoted by $\Re$, this new terminal was used to introduce a fixed-sized pool of randomly generated constants into the initial population. By making many constants available to the evolutionary process, new constants can be assembled with traditional arithmetic operators and exchanged between chromosomes. That is, the values of constants are not explicitly manipulated, but instead, multiple constants are combined. Thus, as the evolutionary process progresses, better constants are expected to emerge while the less useful constants should disappear. Initially, similar rational was adopted in PGEP, but instead of small random numbers, prime numbers were used. Other, more sophisticated proposals have followed and most of these methods can be organized into three broad categories.

2.2.2 Embedded Optimizers

The first category employs a second optimization algorithm that is embedded inside the GP-like environment for a localized search. This approach then uses the optimization algorithm of choice to "tweak" the values of the constants. Probably the most relevant literature to the work in this thesis is that of (Zhang et al. 2007). There, Differential Evolution (DE) was shown to significantly improve the performance of GEP on two SR problems with real-valued constants. An additional gene called the Random Number Gene (RNG) was included in the terminal set and independent instances of this gene were introduced into a GEP population, both initially and via mutation. For each chromosome in the GEP population, a fresh and independent DE population was initialized with the constants in the chromosome as a seed. The DE algorithm was then invoked for a fixed number of generations in order to improve the constants. As anticipated, the number of times the RNG gene appeared in a chromosome would vary, and this quantity conveniently defined the dimensions of the corresponding DE vectors. Once optimized, the values would be inserted back into the GEP population where normal GEP operators were randomly applied thereafter. Although the basic idea is the same, it will be seen later on that the method proposed herein drastically differs in the details, particularly with respect to the embedding.

A very similar technique is described in (Cagnoni et al. 2005), but instead of DE, a bit-based Genetic Algorithm (GA) was used. Unlike the previous work, only SR problems with integer coefficients were considered. Other embeddable alternatives include: a gradient based search (Topchy and Punch 2001), a moving least squares algorithm (Raidl 1998), and a simulated annealing inspired algorithm (Fernandez and Evett 1998). To some degree, these types of searches seem restrictive as the optimization of constants occurs within the context of a single chromosome. Such techniques may produce overly specialized values where the constants are optimal but the structure and makeup of the chromosome are not. This appears to stand in stark contrast to the improvisational search that evolutionary algorithms are so well known for. There is also reason for concern as none of the previous authors have demonstrated how well the solutions generalize on testing sets. Finally, assuming that these methods are applied at every generation and to all chromosomes in a population, they can quickly become rather computationally expensive. But to be fair, (Zhang et al. 2007) and (Cagnoni et al. 2005) explored various levels of interleaving and (Zhang et al. 2007) also presented the best and worst solutions as proof.

2.2.3 Special Operators

The collection of research belonging to the second category primarily focuses on numerically oriented operators. In (Ferreira 2006), each chromosome in a GEP population was given its own fixed-sized set of real-valued constants and the constants are inserted into an expression tree via indices, i.e., a single gene c_i for the i^{th} associated constant. In order to introduce some variation into the constants, a numerical mutation operator was devised. Complementing this new operator, transposition and recombination operators were also developed to respectively "shuffle" constants about the chromosome and "circulate" constants throughout the population. A similar mutation operator was introduced in (Lopes and Weinert 2004), but a local search using a hill-climbing technique was conducted over the constant terminals instead.

Assuming that small improvements in the fitness are caused by minor changes in the constants, (Ryan and Keijzer 2003) introduced two different types of mutation, namely uniform and creep. Adopting a global table of sorted constants, indices into this table were then allowed to be mutated instead of the actual constants. That is, the number and value of constants were fixed throughout the entirety of a run. Similar to the work in (Ryan and Keijzer 2003), (Li et al. 2004) proposed greedy, elitist, and temporal extensions (e.g., the first n generations or every n^{th} generation) to the constant mutation operators of GEP.

2.2.4 Unconventional Methods

Much of the remaining attention given to the area of constant creation has focused on methods which are best described as unconventional. For example, a novel digit concatenation approach driven by Grammatical Evolution (GE) was evaluated in (O'Neill et al. 2003; Dempsey 2005; Byrne et al. 2009). Unlike all the other surveyed methods, this extension was the most harmonious with SR. That is, it too is mostly concerned with symbolic, and not direct numerical manipulations. As opposed to some of the other methods reviewed, the digit concatenation approach promotes incremental improvements while still permitting the quick generation of numerically distant and unrestricted constants, which is a weakness of some of the special operators previously reviewed in Section 2.2.3. It also has the advantage of being extremely easy to integrate into the GE algorithm since only the grammar must be edited.

Preferring to avoid the noise introduced by non-optimal numerical constants, (Keijzer 2003) took a considerably different approach and focused on improving the general shape of a function. This was accomplished by a Linear Scaling (LS) technique which performs a linear regression on the output of the evolved program with respect to the targets in the training set. This only introduces two new constants, specifically one for the slope and another for the intercept. Together, these two terms shift (e.g., upward or downward) and scale (e.g., reflect, amplify, or shrink) the shape of a program. Neither of these two constants were persistent, i.e., the constants were not integrated into the GP chromosome and did not transcend a generation. Unfortunately, this last method does not appear to be universally applicable as a linear relationship is assumed. However, LS appears to be a very simple, efficient, and elegant approach to CC. It is based on a tested analytical algorithm that guarantees optimality and requires no additional pass over the training data.

The preliminary empirical benchmarks of LS looked promising and (Pennachin et al. 2010) independently confirmed the reported results. However, (Valigiani et al. 2004) tested LS on a complex real-world problem and concluded that LS exhibited a strong tendency to overfit the training data and was consistently shown to be the worst performing method therein. While it was theoretically proven in (Keijzer 2004) that LS is guaranteed to reduce the training error, (Costelloe and Ryan 2009) concluded that LS by itself does not significantly improve performance in practice. However, when LS was paired with another enhancement named No Same Mates (NSM), statistically significant improvements were in fact observed. It was also shown in (Keijzer 2004) that LS may be generalized to Multiple Linear Scaling (MLS) through the use of branches or multi-genic chromosomes like those proposed in (Ferreira 2006) for GEP.

2.3 Estimation of Distribution Algorithms

Motivated by the solid mathematical foundations of probability theory and inspired by the combination and perturbation of solutions in the population based searches of Evolutionary Computation (EC), Estimation of Distribution Algorithms (EDAs) are innovative search techniques which fundamentally differ from their predecessors. Instead of adopting the highly unpredictable genetic operators traditionally employed within EC based approaches, EDAs utilize probabilistic models to effectively explore the search space for promising solutions (Larrañaga 2002; Shan et al. 2006).

Starting with a randomly initialized population, an EDA builds or learns a probabilistic model from the most promising solutions encountered in the initial population. Usually the most promising solutions of the population are determined by one of the commonly used global selection strategies and the remaining undesirable solutions are discarded. The constructed model is then sampled to generate new and possibly improved solutions which exhibit the useful interactions inferred from a subset of the previous generation's population. Depending on the desired behavior, the sampling may generate a completely new population or merely generate a smaller number of likely individuals, which are then integrated into the population according to some substitution scheme. Once the new population has been formed, promising solutions are again identified and the preceding probabilistic model can be iteratively updated or simply replaced with an entirely new model. The above procedure is then conducted in generational manner until some pre-specified termination criterion is met.

The most interesting idea demonstrated in the EDA paradigm is that of the indirect pursuit of solutions. Since superior individual solutions are neither directly manipulated nor evolved, the probabilistic model must maintain an accurate and structured representation of the relationships observed between genes in the selected solutions. To what degree or extent of the relationship is captured is determined by the limitations of the chosen model (e.g., univariate, bivariate, or multivariate), but as a model increases in complexity, so do the computational costs associated with approximating the model's parameters and or structure (Pelikan et al. 2002). The idea of identifying, capturing, and exploiting interactions between genes is very similar to that of the building block, linkage learning, or substructure phenomena which have been extensively studied, both theoretically and empirically, in GAs (Goldberg 1989), GP (Langdon and Poli 2002), GEP (Ferreira 2006), and PGEP (Li 2006). Instead of relying on various meta-heuristics controlled by experimentally derived parameters which tend to drastically fluctuate between domains and even problems, an EDA produces likely building blocks by assembling combinations of the dominant dependencies encoded in the model.

One type of graphical probabilistic model that has received much attention in the EDA research and is of particular interest to the research presented hereafter, is the Bayesian Network (BN) (Neapolitan 2004). The inherent flexibility of BNs allows for various interpretations of the random variables associated with each node in the model's graphical representation. For example, in Estimation of Distribution Programming (EDP) (Yanai and Iba 2006) the objective is to accurately model the dependencies between directly connected parent and child nodes in promising tree-based solutions. Extensive relationships like ancestral or sibling dependencies are assumed to be too distant or specific, and thus ignored in EDP. As a result, the structure of the underlying BN in EDP is restricted to a complete and directed binary tree of a pre-determined height, and each random variable is an estimated conditional probability distribution over the children of the identical parental position in select tree-based solutions.

As previously mentioned, EDAs indirectly discover solutions through probabilistic models and this unique approach was briefly illustrated through the introduction of EDP. Therefore at this point, it should be somewhat clearer as to how this indirect search is conducted. Or more precisely, instead of searching for promising solutions in a space of tree based representations, EDP searches for desirable solution distributions in the space of tree distributions. A solution distribution may then be conveniently interpreted as a probabilistic generalization which satisfactorily describes the actual solution to a problem (Shan et al. 2006).

2.4 Hidden Markov Models

2.4.1 Introduction and Example

A concise definition of a Hidden Markov Model (HMM) is a finite specification of a process that assigns probabilities to sequences of symbols or observation sequences (Upper 1997). One convenient method commonly used to depict HMMs is a directed graph. In the graph shown in Figure 2, each node represents a distinct hidden state and every edge signifies a unique state transition. An individual state exercises its ability to emit an observation symbol between transitions, but the set of observable symbols may vary from state to state and normally only one symbol is observed per visit.

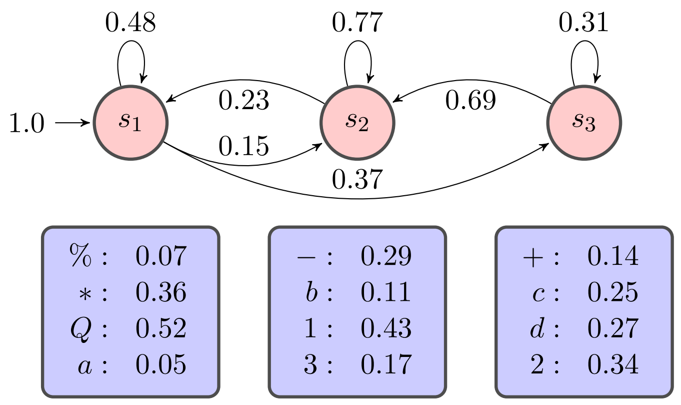


Figure 2. An example of a sparse, partially connected three state HMM with one initial state, twelve probabilistically observable symbols, and seven probabilistic transitions.

For example, recall the PGEP chromosome in Figure 1C. Realizing that the genes may be treated as observable symbols, this model could generate that chromosome by visiting the sequence of states and observing the symbols as shown in Figure 3. But to be of more practical use, the graphical representation must be augmented with some kind of probabilistic information reflecting the behavior of the desired process. That is, there are conditional probability distributions which dictate the observation of symbols and the occurrence of state transitions.

$$\begin{tikzcd}
s_1 \arrow[r] \arrow[d] & s_1 \arrow[r] \arrow[d] & s_1 \arrow[r] \arrow[d] & s_1 \arrow[r] \arrow[d] & s_2 \arrow[r] \arrow[d] & s_1 \arrow[r] \arrow[d] & s_3 \arrow[r] \arrow[d] & s_3 \arrow[r] \arrow[d] & s_3 \arrow[r] \arrow[d] & s_3 \arrow[r] \arrow[d] & s_2 \arrow[r] \arrow[d] & s_2 \arrow[r] \arrow[d] & s_2 \arrow[r] \arrow[d] & s_2 \arrow[d] \\
\% & * & a & Q & b & * & 2 & + & c & d & - & 1 & b & 3
\end{tikzcd}$$

Figure 3. An illustration of the state transitions ($\rightarrow$) and symbol observations ($\downarrow$) that might have generated the PGEP chromosome in Figure 1C using the HMM in Figure 2.

Particularly important to the model under consideration is the notion of time. Like almost all interesting processes, the state of a model may vary with time. But unlike reality where time is continuous, the finite representation of the model forces a reduction in precision and thus only discrete equidistant times are considered. As anticipated, the primary purpose of a state transition is to cause a change in state and such transitions naturally occur at each instant in time. It should be noted that a state transition does not always result in a distinguishable change of state. That is, the actual state before and after a transition may remain the same. This type of behavior is consistent with and essential for processes that can repeat a series of actions over some period of time.

Limiting the scope to simpler HMMs of the first order in time, the probability of a particular state transition is only influenced by two quantities, the probability of occupying the current state at a certain time and the probability of being in some other state at the next instant in time. Furthermore, the observance of a symbol is solely determined by the probability of being in a particular state at a given time and the probability of emitting the symbol while at that same state.

2.4.2 Formal Definition

More formally, a Hidden Markov Model (HMM) may be described as a quintuple $\lambda = (S, V, \mathbf{A}, \mathbf{B}, \mathbf{\Pi})$ of parameters, where $S = \{s_1, \dots s_m\}$ is the set of states of size m, $V = \{v_1, \dots, v_n\}$ is the set of observable symbols of size n, $\mathbf{A}$ is the $m \times m$ state transition probability distribution matrix, $\mathbf{B}$ is the $m \times n$ observation symbol probability distribution matrix, and $\mathbf{\Pi}$ is the initial state probability distribution vector of length m. If q_t denotes the state occupied at time $t > 0$, then the probability distributions $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{\Pi}$ are defined according to Equation 2.1, Equation 2.2, and Equation 2.3, respectively.

$$\mathbf{A} = \{a_{i,j} = \mathbb{P}(q_t = s_j | q_{t-1} = s_i)\} \qquad 1 \leq i, j \leq m \tag{2.1}$$

$$\mathbf{B} = \{b_{i,j} = \mathbb{P}(v_j | q_t = s_i)\} \qquad 1 \leq i \leq m, 1 \leq j \leq n \tag{2.2}$$

$$\mathbf{\Pi} = \{\pi_i = \mathbb{P}(q_0 = s_i)\} \qquad 1 \leq i \leq m \tag{2.3}$$

However, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{\Pi}$ must also conform to the stochastic constraints expressed in Equation 2.4, Equation 2.5, and Equation 2.6, respectively.

$$a_{i,j} \geq 0 \qquad 1 \leq i, j \leq m \qquad \sum_{j=1}^{m} a_{i,j} = 1 \qquad 1 \leq i \leq m \tag{2.4}$$

$$b_{i,j} \geq 0 \qquad 1 \leq i \leq m, 1 \leq j \leq n \qquad \sum_{j=1}^{n} b_{i,j} = 1 \qquad 1 \leq i \leq m \tag{2.5}$$

$$\pi_i \geq 0 \qquad 1 \leq i \leq m \qquad \sum_{i=1}^{m} \pi_i = 1 \tag{2.6}$$

2.4.3 Conventional Parameter Estimation

Although there are typically three canonical problems associated with HMMs (Rabiner 1989), the work in this thesis will only be concerned with one of the three. This particular problem is certainly the most difficult and probably the most important, and it is that of adjusting a model's parameters in order to maximize the emission probabilities of a set of observation sequences $\Theta = \{O_1, O_2, \ldots, O_l\}$. In other words, given a model λ and the set Θ, how can the parameters of λ be adjusted such that the probability of observing Θ is maximized? The seminal technique used to deal with this challenging task is the Baum-Welch (BW) training algorithm (Rabiner 1989). BW is an example of an Expectation-Maximization (EM) (Dempster et al. 1977) algorithm, which iteratively updates and improves the parameters of an HMM (Rabiner 1989). Although BW is extensively used to re-estimate a model's parameters in practice, it tends to get stuck in local optimums, is unable to infer the optimum number of hidden states, and is sensitive to a model's initial parameters. But in light of these drawbacks, HMMs along with BW have been widely adopted and shown to work well in practice.

2.4.4 Biologically Inspired Parameter Estimation

The increased adoption of HMMs in a variety of scientific areas has lead to a resurgent interest in the development of new and improved training methods. Some of the most recently published research (Won et al. 2005; Volkert 2006; Nootyaskool and Kruatrachue 2006) has focused on evolving the probabilistic parameters and topologies of HMMs. These approaches use GAs with arbitrarily applied ad-hoc operators to add or delete states and transitions, swap one or more parameters between models, or randomly disturb the parameters of a model. One immediate advantage of these customized GAs is that the definitions of the operators are able to explicitly consider and obey the stochastic constraints. While this is convenient, many highly specialized operators can be invented and this may lead to an excessive number of control parameters.

Other works (Rasmussen and Krink 2003; Aupetit et al. 2006; Xue et al. 2006) have also investigated the feasibility of utilizing linear combinations for optimum model discovery. The particular method by which these works obtain new models is through the use of the increasingly popular Particle Swarm Optimization (PSO) algorithm (Kennedy and Eberhart 1995). While inspired less by evolution and more by social dynamics, PSO adapts the social interactions present in animal swarms to search spaces in optimization problems. But unlike the previously introduced GAs, the surveyed PSOs constantly violate the strict constraints imposed by Equation 2.4, Equation 2.5, and Equation 2.6. Various penalty schemes and repair mechanisms have therefore been developed in order to handle or enforce these known constraints. Furthermore, the particles in PSO are generally of a fixed dimension and this quality makes the exploration of the state space more difficult in practice.

Still, many of these previously mentioned works have not completely discarded BW as it has been hypothesized (Volkert 2006) that an HMM search space induces a unique terrain which cannot be easily navigated. This claim has been supported by the empirical investigations of individual works, mainly (Volkert 2006). Similar in spirit to the works concerned with PSO, the approach described herein uses a linear combination of multiple models in conjunction with an embedded BW training algorithm. A closely related optimization algorithm known as DE has also been investigated for HMM estimation in (Sá et al. 2008). Like PSO, DE relies on scaled vector differences, but in all actuality, DE may be viewed as a more generalized vector based search with PSO as a novel variant.

2.5 Differential Evolution

Differential Evolution (DE) is an example of a global, derivative-free optimization algorithm that combines a multi-point based search with a generate-and-test paradigm (Price et al. 2005). Similar to other EAs, DE adopts a generational approach and maintains a constant sized population of solutions encoded as fixed-length, real-valued vectors. Exploration of the search space is conducted by means of innovative trial vectors which are the result of perturbations and combinations of other vectors. One noteworthy advantage of the DE algorithm is that the step-size is dynamic in nature (Price et al. 2005). That is, the magnitude of movements throughout the search space are free to increase or decrease.

In fact, since the step-size is determined with respect to a small subset of the population, it is not abnormal for the step-size to vary between generations or even during a single generation. This greatly decreases the likelihood that the DE algorithm will get stuck in a local optimum as it has the ability to escape when the entire population has not yet completely converged. Adopting the original notation and terminology from (Price et al. 2005), the classic version of DE is now introduced in more detail.

At any given generation g, the Np sized population $\mathbf{P}_{\mathbf{x},g}$ is defined according to Equation 2.7 where i denotes the i^{th} vector $\mathbf{x}_{i,g}$ of $\mathbf{P}_{\mathbf{x},g}$, j denotes the j^{th} parameter $x_{j,i,g}$ of $\mathbf{x}_{i,g}$, and D is the number of parameters in or the dimension of $\mathbf{x}_{i,g}$.

$$\mathbf{P}_{\mathbf{x},g} = (\mathbf{x}_{i,g}), g = 0, \ldots, g_{max}, i = 0, \ldots, Np-1 \quad (2.7)$$

$$\mathbf{x}_{i,g} = (x_{j,i,g}), j = 0, \ldots, D-1 \quad (2.8)$$

When $g = 0$, the initial population $\mathbf{P}_{\mathbf{x},0}$ is randomly initialized according to Equation 2.9 where $b_{j,L}$ and $b_{j,U}$ respectively represent the initial lower and upper bounds of $x_{j,i,0}$, and $\text{rand}_j(0,1)$ returns a uniformly distributed random number from $[0,1)$.

$$x_{j,i,0} = \text{rand}_j(0,1)(b_{j,U} - b_{j,L}) + b_{j,L} \quad (2.9)$$

For $g > 0$, an intermediate population $\mathbf{P}_{\mathbf{v},g}$ can be generated by one of the Differential Mutation (DM) operators defined in Equation 2.10 or Equation 2.11.

$$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F \cdot (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}) \quad (2.10)$$

$$\mathbf{v}_{i,g} = \mathbf{x}_{e,g} + F \cdot (\mathbf{x}_{r1,g} + \mathbf{x}_{r2,g} - \mathbf{x}_{r3,g} - \mathbf{x}_{r4,g}) \quad (2.11)$$

In Equation 2.10 and Equation 2.11, $\mathbf{x}_{r0,g}$ and $\mathbf{x}_{e,g}$ are known as the base vectors, and $\mathbf{x}_{r1,g}$, $\mathbf{x}_{r2,g}$, $\mathbf{x}_{r3,g}$, and $\mathbf{x}_{r4,g}$ are called the differencing vectors. The indices of these vectors are randomly selected such that $i \neq r0 \neq r1 \neq r2 \neq r3 \neq r4$ as this avoids obvious degenerate vector combinations, i.e., $\mathbf{x}_{i,g} \neq \mathbf{v}_{i,g}$. Unlike the other vectors, $\mathbf{x}_{e,g}$ does not have an index and is called the elite vector, which is the best vector encountered up to the g^{th} generation. Besides the number of differences, the main distinction between Equation 2.10 and Equation 2.11 is that the latter strategy is greedy, focusing the search in the general vicinity of the elite vector. Commonly referred to as the scaling factor, $F \in (0, 1+)$ and "amplifies" or "dampens" the perturbations by relatively adjusting the step-size (Price et al. 2005). While F can be randomized, it is assumed herein that F remains fixed throughout the duration of a run.

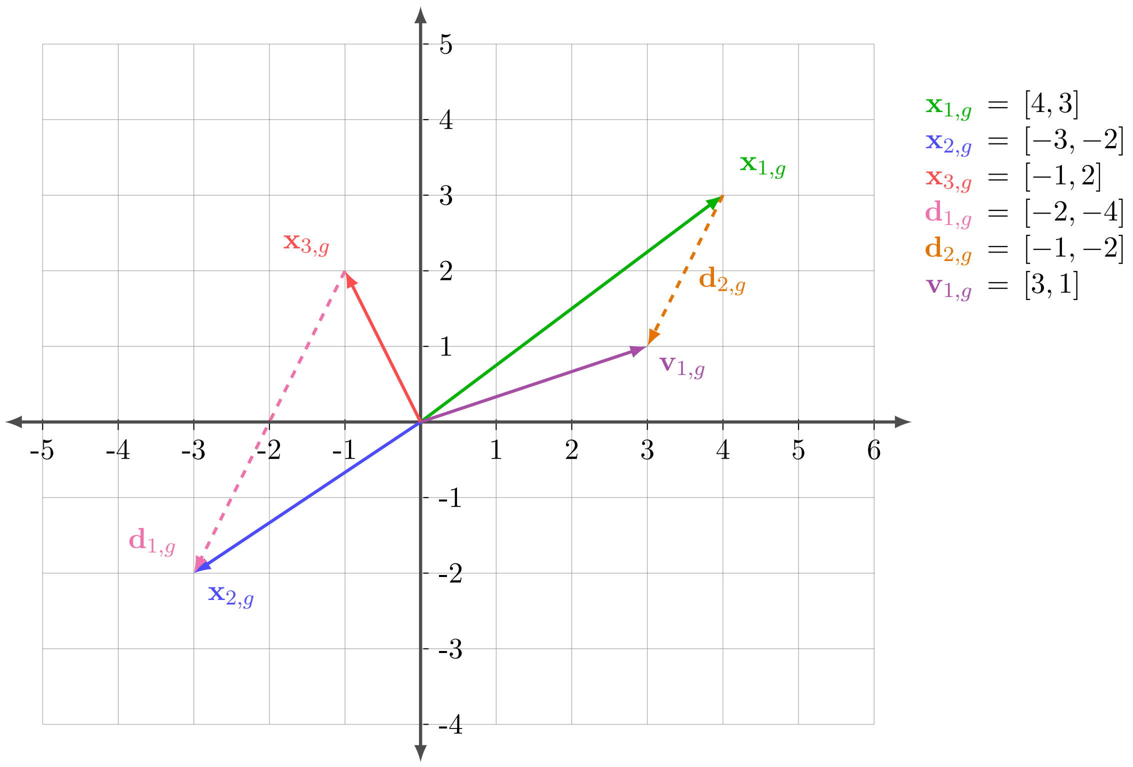

$\mathbf{x}_{1,g} = [4, 3]$
$\mathbf{x}_{2,g} = [-3, -2]$
$\mathbf{x}_{3,g} = [-1, 2]$
$\mathbf{d}_{1,g} = [-2, -4]$
$\mathbf{d}_{2,g} = [-1, -2]$
$\mathbf{v}_{1,g} = [3, 1]$
$\mathbf{x}_{1,g}$
$\mathbf{x}_{3,g}$
$\mathbf{d}_{2,g}$
$\mathbf{v}_{1,g}$
$\mathbf{d}_{1,g}$
$\mathbf{x}_{2,g}$

Figure 4. A concrete example of the DE/rand/1 DM strategy in two dimensional space with a scaling factor of $F = 0.5$. The base vector is $\mathbf{x}_{1,g}$, the distinct vectors used for the difference are $\mathbf{x}_{2,g}$ and $\mathbf{x}_{3,g}$, the unscaled difference vector is $\mathbf{d}_{1,g}$, the scaled difference vector is $\mathbf{d}_{2,g}$, and the resulting mutant vector is $\mathbf{v}_{1,g}$.

Numerous other DM strategies have been developed (Price et al. 2005; Feoktistov 2006). So many in fact, that certain notational conventions have emerged to succinctly refer to the various strategies. For example, Equation 2.10 and Equation 2.11 are referred to as DE/rand/1 and DE/best/2, respectively. The *best* or *rand* part describes how the base vector is chosen where the trailing number details the exact number of scaled differences.

After mutation, a trial population $\mathbf{P}_{\mathbf{u},g}$ is produced using the uniform crossover operator as defined in Equation 2.12.

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \mathrm{rand}_j\left(0,1\right) \leq Cr \text{ or } j = j_{\mathrm{rand}}, \\ x_{j,i,g} & \text{otherwise.} \end{cases} \tag{2.12}$$

This operation combines $\mathbf{x}_{i,g}$ with $\mathbf{v}_{i,g}$ to yield $\mathbf{u}_{i,g}$ by using the crossover probability $Cr \in [0,1]$ to approximate the fraction of parameters that $\mathbf{u}_{i,g}$ inherits from $\mathbf{v}_{i,g}$ (Price et al. 2005). Similar in purpose to the mutually exclusive indices in DM, j_{rand} forces innovation by ensuring that $\mathbf{u}_{i,g} \neq \mathbf{x}_{i,g}$. Cr may also be randomized, but like before, it is assumed that Cr is held constant over all generations.

Finally, DE utilizes the local, greedy, and reproductionless selection strategy described in Equation 2.13.

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f(\mathbf{u}_{i,g}) \leq f(\mathbf{x}_{i,g}), \\ \mathbf{x}_{i,g} & \text{otherwise.} \end{cases} \tag{2.13}$$

Here, f is the objective function and the optimization task is to find a vector $\mathbf{x}^*$ such that $f(\mathbf{x}^*) = 0.0$, i.e., the goal is to find a global optimum $\mathbf{x}^*$ which minimizes $f(\mathbf{x}^*)$. Although less strict, it is often more convenient to establish an error threshold ϵ where any $\mathbf{x}^*$ which satisfies $f(\mathbf{x}^*) \leq \epsilon$ is also treated as optimal. An optimization task is then considered to be a success when $\mathbf{x}^*$ is found or a failure if g_{max} is reached.

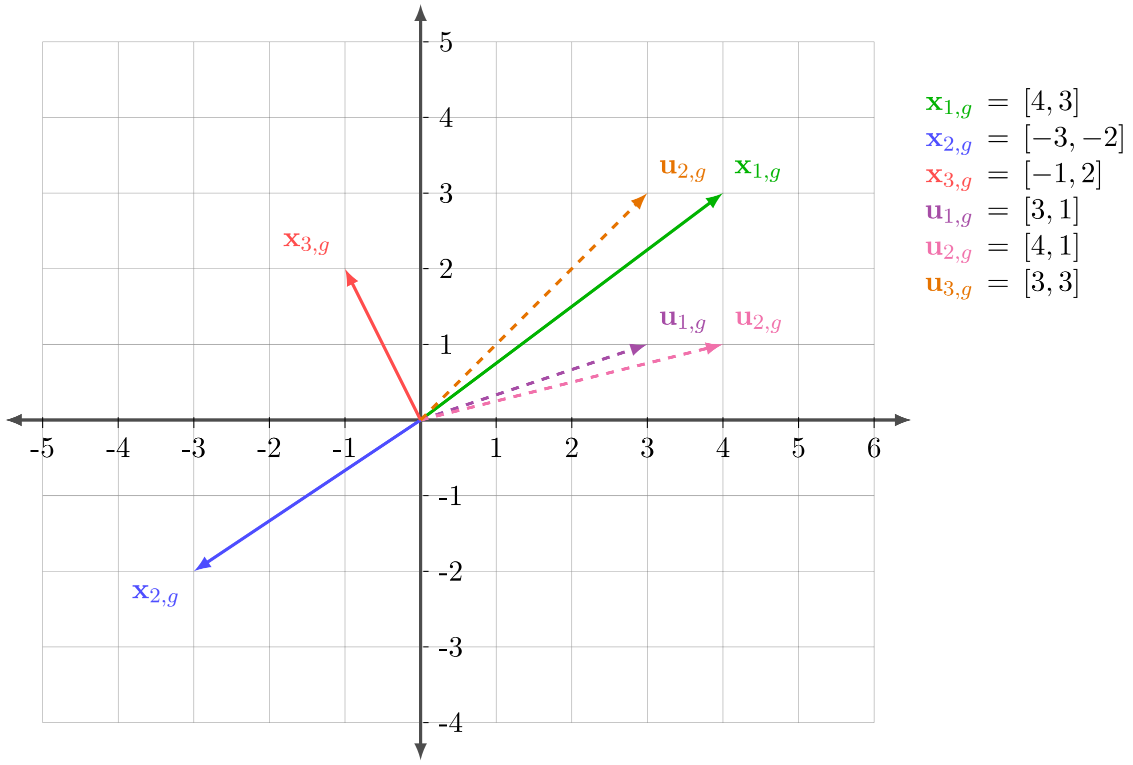


Figure 5. Demonstrates all possible trial vector combinations produced by uniformly crossing over $\mathbf{x}_{1,g}$ and $\mathbf{v}_{1,g}$ in two dimensional space. The trial vector $\mathbf{u}_{1,g}$ adopts both perturbations introduced by DE/rand/1 into the mutant vector $\mathbf{v}_{1,g}$ where $\mathbf{u}_{2,g}$ and $\mathbf{u}_{3,g}$ each only adopt a single but distinct mutational perturbation.

3. NUMERICAL OPTIMIZATION OF PROBABILISTIC MODELS FOR PROGRAM SYNTHESIS

3.1 Overview

In this chapter, a novel combination of techniques which straddles the boundaries of two divisionary camps is proposed. Admittedly, the mathematical assurances that the EDA (Shan et al. 2006) philosophy offers is attractive, e.g., capturing relational and positional dependencies between discrete elements in multiple sequences and then encoding this knowledge in a single probabilistic graphical model. But at the same time, the complexity of learning optimal parameters and topologies for the most useful, and thus desirable probabilistic graphical models, is intractable (Chickering 1996).

Still, preferring to avoid the consequences of local stagnation, which gradient and Newtonian based learning algorithms are highly susceptible to, the eligibility of biologically inspired search algorithms for the purposes of optimum model discovery must be seriously considered. Through the use of competition, mutation, and reproduction, probabilistic models are evolved and assessed. One such model which has previously failed to make an appearance in the EDA literature is the HMM, and thus its feasibility will be explored herein.

Well-suited to evolve the parameters of an HMM, the DE algorithm is recruited for the purposes of global numerical optimization, which was previously described in Section 2.5. But while not completely abandoning more classical parameter estimators for HMMs, the BW training algorithm mentioned back in Section 2.4.3 is selectively applied as a complimentary and reinforcing local maximizer. Moreover, instead of using the evolved models in a sequence based classification system, the probabilistic properties of the model are utilized to generate a sample of likely linear PGEP chromosomes, which are then transformed into ET for fitness evaluations.

The fittest models, as determined by a sample of chromosomes, will survive, undergo vector-based perturbations, and as the evolutionary process continues, hopefully generate fitter solutions. This proposed approach then indirectly introduces chromosomal variation through the direct application of genetic operators to the underlying probabilistic model's representation. Or in other words, a probabilistic model assumes the role of the genotype, the resulting tree structures undertake the responsibilities of the phenotype, and the sampling of the model acts as a dynamic ontological process. This algorithm has been called Probabilistically Guided Prefix Gene Expression Programming (PG-PGEP) (Cerny et al. 2008a) and will now be more fully described in detail.

3.2 Probabilistically Guided Prefix Gene Expression Programming Algorithm

The aim is to adopt an approach which employs a population of probabilistic models to guide the search for a fit SR solution. In doing so, the direct application of genetic operators to a symbolically encoded chromosome is abandoned, and a strictly generative methodology is optimistically favored. The parameters of a probabilistic model which reside in the continuous, but constrained real-valued domain are evolved, and subsequently utilized to influence the generation of several solutions. Coincidentally, DE fulfills the evolutionary needs of a real-valued representation, HMMs are probabilistic models which can be easily and efficiently evolved, and finally, PGEP is a generally applicable sequence based problem solving technique.

Due to the specific requirements of the intended application domain of SR, the task at hand can best be described as finding a model that emits a sequence which adequately expresses an unknown mathematical function. And thereby inferring the underlying hidden states of the evolved HMM. This is attempted by only considering a set of numerical inputs and the corresponding outputs of a training set. Compared to other common applications of HMMs, this problem is quite different in that there is no readily available corpus of reliable example observations. In fact, the usage of HMMs here is rather unconventional in that HMMs are used in a generative fashion as opposed to assessing the probability of emitting a set of particular observational sequences. Thus, it is inappropriate to favor models by more traditional criteria like those appearing in (Won et al. 2005; Volkert 2006; Nootyaskool and Kruatrachue 2006; Rasmussen and Krink 2003; Aupetit et al. 2006; Xue et al. 2006).

Nevertheless, there is also another source of difficulty inherit in the problem to be solved. Despite strong similarities between two sequences or solutions, the actual outputs produced by the encoded PGEP programs may be drastically different. For instance, consider what would happen if the root node in Figure 1A was replaced with a different function of the same arity. While this would fail to re-shape the ET it would, in most cases, significantly alter the outputs of the encoded program when evaluated on the same data set.

Several crucial aspects of this algorithm will now be discussed in greater detail. This includes a description of the vectorized representation of HMMs, the generation and sampling schemes used in determining the quality of the evolved models, and additional details about the hybridization of DE and BW. A complete outline of this proposed evolutionary search can also be seen in Figure 6.

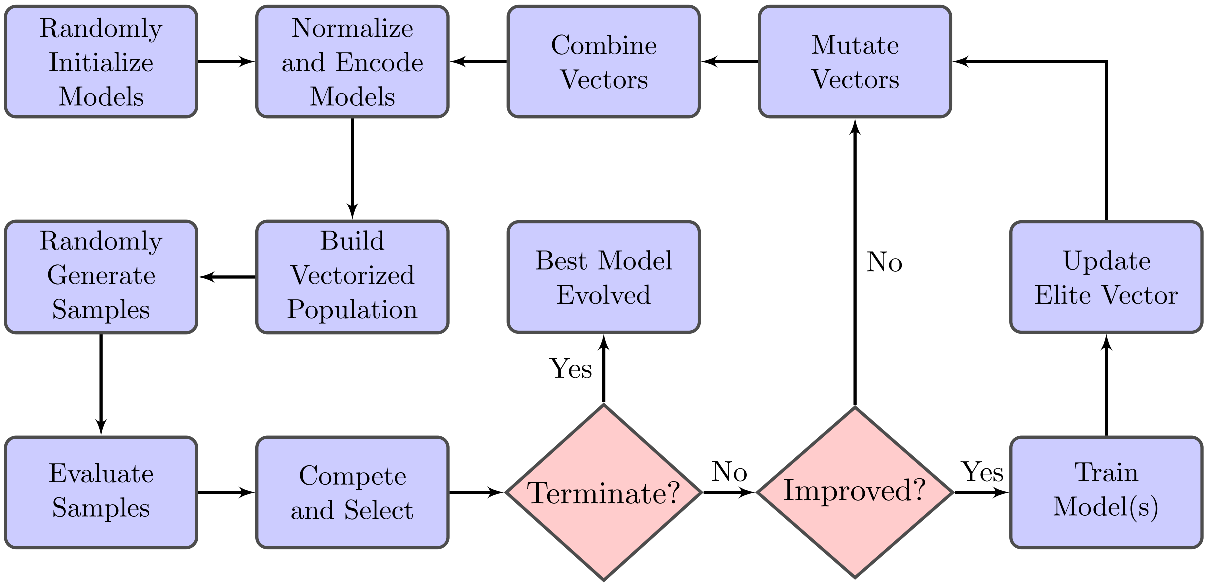


Figure 6. A high-level overview describing the flow of the PG-PGEP algorithm. The first step appears in the upper left-hand corner, the last step is located in the middle, and the remaining steps belong to the main loop.

3.2.1 Hidden Markov Model Encoding Scheme

As is illustrated in Figure 7, any HMM can be easily encoded by arranging the rows of $\mathbf{\Pi}$, $\mathbf{A}$, and $\mathbf{B}$ into a single real-valued vector $\mathbf{x}$. The number of states in an HMM – as empirically determined to be sufficiently complex to undertake a problem – is fixed amongst all individuals and between generations. Now due to the previously highlighted fact that DE has been chosen as the mechanism to evolve the parameters of an HMM, it is wise to reexamine the definition of the DM operators as they appear in Equation 2.10 and Equation 2.11. Clearly, there is no guarantee that the trial vectors produced by this operator will respect the established constraints. Therefore, special techniques which avoid violating or explicitly enforce the constraints of Equation 2.4, Equation 2.5, and Equation 2.6 must be considered first.

$$\mathbf{x} = (\underbrace{\pi_1,\ldots,\pi_m}_{\mathbf{\Pi}}, \underbrace{\underbrace{a_{1,1},\ldots,a_{1,m}}_{\mathbf{A}_1},\ldots,\underbrace{a_{m,1},\ldots,a_{m,m}}_{\mathbf{A}_m}}_{\mathbf{A}}, \underbrace{\underbrace{b_{1,1},\ldots,b_{1,n}}_{\mathbf{B}_1},\ldots,\underbrace{b_{m,1},\ldots,b_{m,n}}_{\mathbf{B}_m}}_{\mathbf{B}})$$

Figure 7. The vectorized representation of an HMM with $|\mathbf{x}| = m + m^2 + m \times n$ real-valued parameters or dimensions.

Constraint aware variants of DE have been previously proposed in (Price et al. 2005), but these mainly rely on the imposition of harsh penalties or require the distinction between feasible and infeasible models. Since even the slightest perturbation of a model can result in a violation of a HMM's complex and interdependent constraints, such techniques do not seem well suited for PG-PGEP. Fortunately, a normalization procedure can both consistently produce valid HMMs and not overly impede the search. Thus, in order to enforce the constraints imposed on **Π**, **A**, and **B**, the appropriate parameters are repaired according to Equation 3.1, Equation 3.2, and Equation 3.3, respectively. Interesting in its own right, the definition of normalization introduces a degree of flexibility into the search. That is, it is now possible to explore models inside of the constrained search space Λ with points in the unconstrained super search space $\mathbb{R}^{|\mathbf{x}|}$.

Finally, since the formulations appearing in Equation 3.1, Equation 3.2, and Equation 3.3 do not default to zero and instead accommodate negative values by taking the absolute value, it is not necessary to establish arbitrary lower bounds. Notice that it would otherwise be possible for the summations in the denominators of Equation 3.1, Equation 3.2, and Equation 3.3 to be equal to zero, thus yielding undefined models, which would also need to be guarded against. Furthermore, the use of arbitrary lower bounds could inadvertently and effectively halt the optimization of certain parameters as zero valued differences may become frequent and persistent.

$$\pi_i = \frac{|\pi_i|}{\sum_{j=1}^{m} |\pi_j|} \qquad 1 \leq i \leq m \tag{3.1}$$

$$a_{i,j} = \frac{|a_{i,j}|}{\sum_{k=1}^{m} |a_{i,k}|} \qquad 1 \leq i, j \leq m \tag{3.2}$$

$$b_{i,j} = \frac{|b_{i,j}|}{\sum_{k=1}^{n} |b_{i,k}|} \qquad 1 \leq i \leq m, 1 \leq j \leq n \tag{3.3}$$

3.2.2 Fitness Evaluations

In most cases, the quality of an HMM is fully or partially determined by the Forward-Backward Algorithm (FBA) (Rabiner 1989) on a given set of example observation sequences. That is, HMMs of higher quality are more likely to produce the presented sequences than HMMs of lower quality. But unfortunately, when SR problems are considered, only the numerical inputs and outputs of the training set are known. In fact, since an observation sequence encodes a solution to a problem, any available observation sequence must only be a speculation as otherwise a solution would have already been discovered. Therefore, it is nearly impossible to evaluate the quality of an HMM in any conventional sense.

As SR is only concerned with discovering a single observation sequence – specifically a sequence which encodes one of potentially many acceptable solutions – the quality of an HMM will be solely characterized by the fitness of the best chromosome generated by a particular model. With this in mind, the quality of an HMM can be interpreted as the expected optimal fitness of a generated chromosome. For although it is possible that better chromosomes may be produced, the corresponding sequences may either be very rare or transcend the fixed-length observations. This definition has two major advantages. First, it allows for the incorporation of a standard measurement of error, and second, it is still able to accurately reflect a model's likely performance through a winner-takes-all style proposition.

3.2.3 Prefix Gene Expression Programming Chromosome Sampling

Fortunately, the structure of an HMM promotes a straightforward and computationally inexpensive means of sequence generation. Naïve in nature, but greedy in practice, the employed generation scheme exploits the formal definition of an HMM to randomly traverse states and observe symbols in an alternating fashion. Utilizing a casino style roulette wheel approach entrenched in many selection strategies, $2m + 1$ roulette wheels are constructed from the rows of $\mathbf{\Pi}$, $\mathbf{A}$, and $\mathbf{B}$. That is, one roulette wheel R_α is built to choose the initial state, m additional roulette wheels $R_{\beta_1}, \ldots, R_{\beta_m}$ are constructed to pick an observation symbol at each state, and another m roulette wheels $R_{\gamma_1}, \ldots, R_{\gamma_m}$ are assembled for each state to select the next state.

Attributable to the stochastic constraints imposed by the definition of an HMM, each of the $2m+1$ roulette wheels has a circumference of exactly 1.0, implying that at least one symbol or state will always be accessible. This conveniently guarantees that a sequence of any length may be generated, but the composition of such a sequence is obviously dependent upon the path traveled through the model. An illustration of the generation scheme is available in Figure 8 and the complimentary pseudocode is presented in Figure 9.

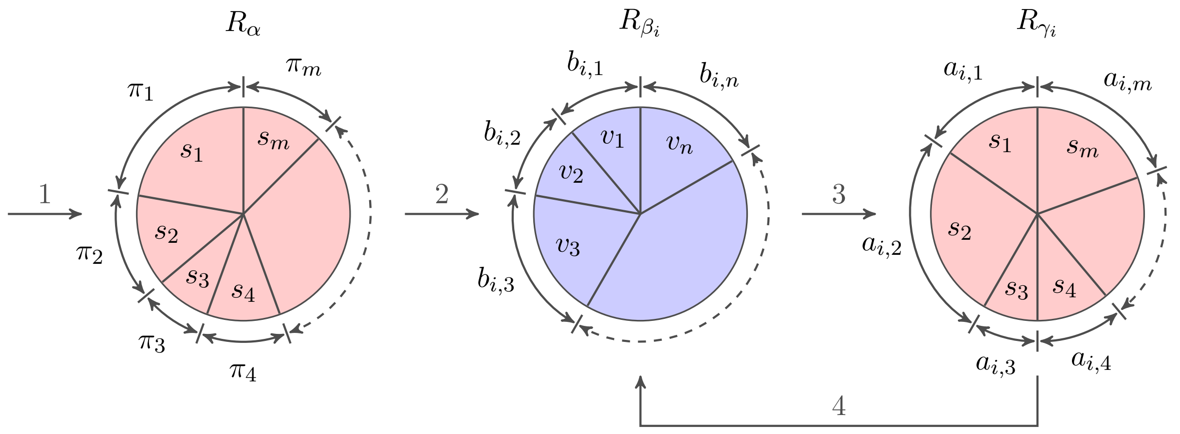


Figure 8. A illustration depicting how the roulette wheels are constructed from the matrices of an HMM and the order in which the roulette wheels are spun to obtain a sequence of genes. The initial state, observation symbol, and next state wheels are located on the left, in the middle, and on the right, respectively.

Sampling simply entails the generation of Z valid (naturally terminating) chromosomes from a given model. Consequently, a sample may contain distinct sequences which encode the same program. Such programs may be equivalent or identical, i.e., same behavior with different makeups or exact duplicates. Initially, the decision to permit duplicates may seem problematic as the diversity of the sample will suffer. But despite this concern, duplicates are actually beneficial as it will reflect the model's bias.

For now that a model's true stochastic behavior is allowed to emerge, the quantification of a model's fitness as described in Section 3.2.2 makes intuitive sense. That is, the search should not only pursue models which generate the best solutions, but also those which are more likely to generate the best solutions. One final comment with regards to the implications of the sampling and generation scheme is the lack of a well-behaved or deterministic[1] fitness mapping. That is, although the fitness function is not dynamic in nature[2], identical models could produce unique samples, and thus the assigned fitness values may be numerically distinct and possibly distant.

3.2.4 Hybridized Training

Acknowledging the unanimous consensus conveyed throughout the evolvable HMM literature, this proposed evolutionary process has been hybridized with the previously discussed BW

[1]Although it is possible to use the Viterbi algorithm (Rabiner 1989) to determine the most-likely state sequence given an observation, it is computationally prohibitive to to generate a complete and exhaustive sample.

[2]PGEP chromosomes are always evaluated in a consistent and static manner, i.e., the training set is fixed in size and composition for all generations and trials.

```
inputs : length of a chromosome $l$
         initial state vector $\mathbf{\Pi}$
         state transition matrix $\mathbf{A}$
         observation symbol matrix $\mathbf{B}$
output: chromosome of genes $v_1\ v_2\ \ldots\ v_{l-1}\ v_l$
begin
   % loop until a valid chromosome has been generated
   repeat
      % select an initial state $s_i$ by spinning $R_\alpha$
      $state$ ← SpinWheel($\mathbf{\Pi}$)
      % randomly select $l$ genes
      for $j \leftarrow 1$ to $l$ do
         % transition to next state from current state $s_i$ by spinning $R_{\beta_i}$
         state ← SpinWheel ($state$, $\mathbf{A}$)
         % emit symbolic gene $v_j$ based on current state $s_i$ by spinning $R_{\gamma_i}$
         gene ← SpinWheel ($state$, $\mathbf{B}$)
         % append gene $v_j$ to chromosome
         chromosome ← Append($chromosome$, $gene$)
      end
   until Validate($chromosome$)
   % valid chromosome with length $l$
   return $chromosome$
end
```

Figure 9. Pseudocode used to generate one fixed-length PGEP chromosome from an HMM using a casino style roulette wheel inspired approach.

training method. But since this variant of PGEP utilizes the structure of a probabilistic model to conduct a search, an alternative approach that selectively applies BW has been devised. A single iteration of BW is invoked only when an improved solution is encountered and it is only applied to the model which emitted the improved sequence. Relative to the existing parameters of the applicable model, the probability of observing the valid (non-junk) gene sequence of the improved solution is then locally maximized. The newly trained model is then appointed the elite rank, and will serve as the base vector $\mathbf{x}_{e,g+1}$ in all future vector innovations $\mathbf{v}_{i,g+1}$.

3.2.5 Experimental Setup

Due to its popularity and practicality as discussed in Section 1.1, SR will serve as the primary benchmark for the proposed PG-PGEP algorithm. As a reminder, goal of this particular task is to find a symbolically encoded mathematical expression to an unknown function, using only a numerically oriented training set. Great difficulty lies within accomplishing this task as in addition to finding an approximation, several other factors are unknown. For instance, to obtain a good solution in a reasonable amount of time, a gene set which can sufficiently express the target function is needed. But since the target function is itself unknown, the ideal composition of the gene set is ambiguous at best. For low dimensional problems like those investigated here, plots of the points might aid in the selection of a gene set. For example, noticeable oscillations or periodicity indicate that common trigonometric functions should be included in the gene set to accelerate the search.

As expected, multiple problems which exhibit distinctive behavior and are of varying degrees of complexity have been selected. The difficulty of such problems is anecdotal at best because there is no universally systematic way to quantify the difficultly of a problem without exhaustively enumerating the entire search space. Among the investigated benchmarks are several synthetic functions which have made an appearance in other research and are reproduced in Equation 3.4 (Koza 1992; Ferreira 2001; Costelloe and Ryan 2009), Equation 3.5 (Koza 1992; Ferreira 2001), Equation 3.6 (Yanai and Iba 2006), and Equation 3.7 (Keijzer 2003; Topchy and Punch 2001; Keijzer 2003; Pennachin et al. 2010). To the left of each equation is a uniquely identifying name (i.e., f_n), which will be used to refer to that problem during the remainder of this thesis. Since these functions are known in advance, it is possible to solve these problems exactly. Attempts will be made with both the original PGEP algorithm and the proposed PG-PGEP algorithm, and this will offer an opportunity to compare the two paradigms.

$$f_1\left(x\right) = x^4 + x^3 + x^2 + x \tag{3.4}$$

$$f_2\left(x\right) = 5x^4 + 4x^3 + 3x^2 + 2x + 1 \tag{3.5}$$

$$f_3\left(x\right) = x\cos(x)\sin(x)\left(\sin^2(x)\cos(x) - 1\right) \tag{3.6}$$

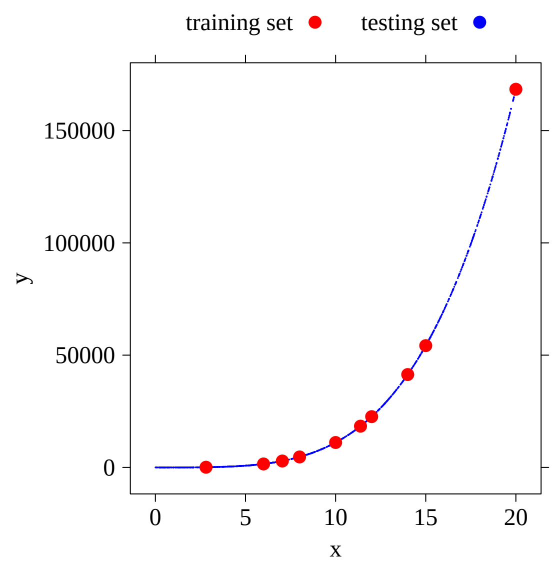
$$f_4\left(x, y\right) = xy + \sin\big((x-1)(y-1)\big) \tag{3.7}$$

Various aspects of the training and testing sets are described in Table I. This includes the interval(s) on which the cases where sampled, what method was used to conduct the sampling (e.g., randomly, arbitrarily, or incrementally with the specified step size), and some basic descriptive statistics about the two sets like the mean (μ) and the standard deviation (σ) of the inputs. Figure 20 contains plots of the training and testing sets summarized in Table I. The points belonging to the training and testing sets are displayed as big red and tiny blue dots, respectively. In most cases, the general shape of the target functions should be distinguishable from the test set because the points are more abundant and were randomly generated using a uniform distribution.

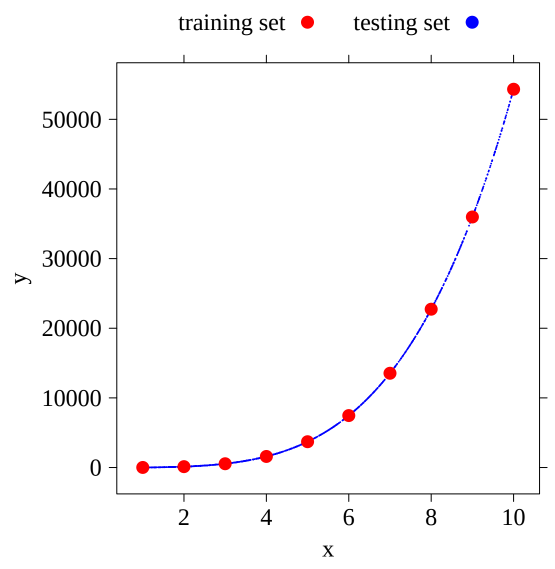
TABLE I
SUMMARIES OF THE FOUR TRAINING AND TESTING SETS USED TO BENCHMARK THE PG-PGEP ALGORITHM.

		Training Set			**Testing Set**		
	Space	**Size**	**Method**	$\boldsymbol{\mu \pm \sigma}$	**Size**	**Method**	$\boldsymbol{\mu \pm \sigma}$
f_1	$x \in [0, 20]$	10	Arbitrary	10.623 ± 4.983	1000	Random	11.309 ± 5.073
f_2	$x \in [1, 10]$	10	$+1.0$	5.5 ± 3.028	1000	Random	5.556 ± 2.634
f_3	$x \in [0, 5.8]$	30	$+0.2$	2.9 ± 1.761	1000	Random	2.9 ± 1.643
f_4	$x \in [-3, 3]$	20	Random	0.073 ± 1.978	2000	Random	0.027 ± 1.755
	$y \in [-3, 3]$		Random	0.226 ± 1.791		Random	0.02 ± 1.691

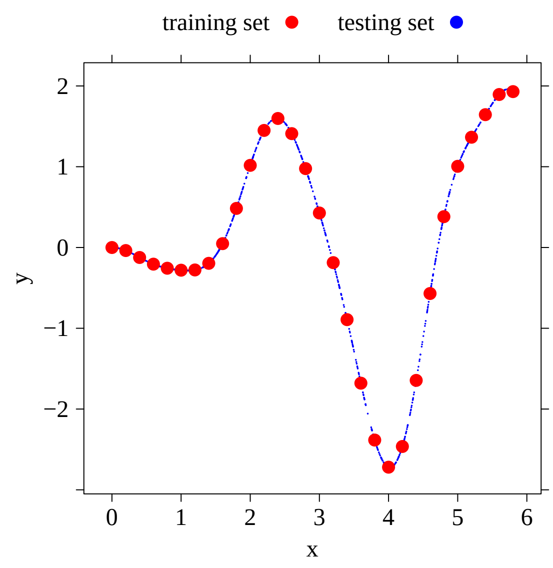
Arbitrary, yet sufficiently long PGEP chromosomes have been used and unless otherwise noted, measure 64 genes in length. Also, problem specific gene sets appear in Equation 3.8 and Equation 3.9. Attempts for the quartic polynomial without integer coefficients (f_1) and quartic polynomial with integer coefficients (f_2) are restricted to the minimalistic gene set G_1. No constants are included in G_1 so all integer coefficients must be synthesized during a run in order to achieve a perfect solution to f_2. However, due to the presence of integer coefficients



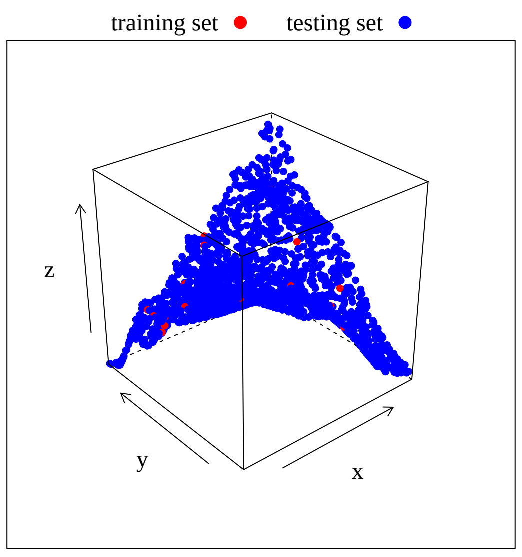
(A) Quartic polynomial without coefficients



(B) Quartic polynomial with coefficients



(C) Univariate trigonometric function



(D) Bivariate trigonometric function

Figure 10. The scatter plots, appearing in a clockwise direction from the top left to the bottom left, are for the quartic polynomials without and with integer coefficients, and the univariate and bivariate trigonometric functions summarized in Table I.

in f_2, longer chromosomes measuring 96 genes in length are warranted when paired with G_1. Solutions to the univariate sinusoide (f_3) and multivariate sinusoide (f_4) are constructed from the expanded gene set G_2, which includes two basic trigonometric functions, specifically sine and cosine.

$$G_1 = \{x, +, -, *, \%\} \tag{3.8}$$

$$G_2 = \{x, \sin, \cos, +, -, *, \%\} \tag{3.9}$$

The columns PG-PGEP and PGEP in Table II group process specific control parameters. Reviewing the control parameters for PG-PGEP: Np is the number of vectorized models in the DE population, m is the number of states in an HMM, F is the differential mutation scaling factor, Cr is the rate of uniform crossover, and Z is the sample size. In the case of the classic PGEP algorithm, distinct notation is now introduced: Pop is the number of symbolically encoded chromosomes residing in the population of a single generation and $Gens$ is the maximum number of generations. Crs, Mut, and Rot are the probabilities of applying traditional PGEP operators like linear crossover (both one-point and two-point), point mutation, and rotation, respectively. Due to the radically different nature of each process, some of these quantities are not truly comparable. Most notably are the population sizes (Np and Pop) and the generational restrictions ($g_{\max}$ and $Gens$). But in spite of this, the total number of fitness evaluations are purposefully equal ($Np \times Z \times g_{\max} = Pop \times Gens = 3.0 \times 10^6$) as this provides a comparable process independent quantity.

TABLE II
AN OVERVIEW OF THE CONTROL PARAMETERS USED ON ALL PROBLEMS FOR THE SPECIFIED EVOLUTIONARY PROCESS

PG-PGEP						**PGEP**				
Np	$g_{\max}$	F	Cr	Z	m	Pop	$Gens$	Crs	Mut	Rot
30	100	0.8	0.1	1000	6	1000	3000	0.7	0.02	0.02

Lastly, the Root Mean Squared Error (RMSE) will serve to evaluate the quality of a chromosome and thus an HMM as well. A slightly modified version of the RMSE for PG-PGEP appears in Equation 3.10 where $\hat{f}_i$ is the fitness of the i^{th} HMM in a population, S_i is a sample from the i^{th} HMM, p is a program (chromosome) belonging to S_i, n is the size of the training set, p_j is the actual output of p on the j^{th} training case, and finally, t_j is the target output for the j^{th} training case. But for ease of presentation, the remainder of the results will be expressed in terms of a scaled and maximized fitness which is defined in Equation 3.11. Thus any fitness $\hat{f}_i' \in [0, 1000]$, and the extremes represent the worst and best attainable values, respectively.

$$\hat{f}_i = \min_{p \in S_i} \sqrt{\frac{1}{n} \sum_{j=1}^{n} (p_j - t_j)^2} \tag{3.10}$$

$$\hat{f}'_i = 1000 \left(\frac{1}{1 + f_i} \right) \tag{3.11}$$

3.2.6 Empirical Results

An examination of Table III will reveal many of the commonly quoted end-of-run performance measures. In order to capture the stochastic behavior of these two competing searches, the quantities appearing in Table III have been determined from a total of 50 independent runs. Each symbolic regression problem is identified by name and two sets of results are reported. One for PGEP and the other for PG-PGEP, and such results appear in their own discernible rows. In addition, process specific results are composed of two separate but related entries identified by subscripts. Namely, those obtained during training and those which detail the generalization performance over the testing set.

The *Best Fitness* header covers various quantities which describe the best end-of-run scaled fitness values ($\hat{f}'$). The $\hat{f}'_{\text{max}}$ and $\hat{f}'_{\text{min}}$ columns identify the overall fittest and least fit final solutions when strictly evaluated on the training set. Since $\hat{f}'_{\text{max}}$ is determined solely with respect to the training set, the generalized performance on the testing set will reveal whether or not the supposedly best solution is deceptive and overfits the training data. Moreover, $\overline{\hat{f}'}$ and $\sigma_{\hat{f}'}$ represent the mean and standard deviation of all end-of-run fitness values. The *Size* column lists select details about the sizes of the best end-of-run programs. Here, each quantity

TABLE III
BEST END-OF-RUN RESULTS OBTAINED FOR PGEP AND PG-PGEP OVER 50 INDEPENDENT RUNS ON EACH OF THE FOUR SR BENCHMARK PROBLEMS.

		Scaled Best Fitness ($\hat{f}'$)				**Size** (s)			
		$\hat{f}'_{\text{max}}$	$\hat{f}'_{\text{min}}$	$\overline{\hat{f}'}$	$\sigma_{\hat{f}'}$	s_{max}	s_{min}	$\bar{s}$	σ_s
f_1	PGEP$_{\text{train}}$	1000.0	1000.0	1000.0	0.0	$\frac{13}{64}$	$\frac{27}{64}$	$\frac{16.04}{64}$	4.062
	PGEP$_{\text{test}}$	1000.0	1000.0	1000.0	0.0				
	PG-PGEP$_{\text{train}}$	1000.0	1000.0	1000.0	0.0	$\frac{13}{64}$	$\frac{51}{64}$	$\frac{19.12}{64}$	7.604
	PG-PGEP$_{\text{test}}$	1000.0	1000.0	1000.0	0.0				
f_2	PGEP$_{\text{train}}$	1000.0	1.975	298.380	367.408	$\frac{35}{96}$	$\frac{27}{96}$	$\frac{35.24}{96}$	8.766
	PGEP$_{\text{test}}$	1000.0	2.181	298.912	363.358				
	PG-PGEP$_{\text{train}}$	1000.0	11.030	307.185	257.535	$\frac{49}{96}$	$\frac{65}{96}$	$\frac{44.72}{96}$	10.881
	PG-PGEP$_{\text{test}}$	1000.0	12.558	307.659	257.067				
f_3	PGEP$_{\text{train}}$	808.545	606.731	768.479	43.408	$\frac{14}{64}$	$\frac{15}{64}$	$\frac{11.28}{64}$	1.415
	PGEP$_{\text{test}}$	814.309	605.806	769.950	43.548				
	PG-PGEP$_{\text{train}}$	1000.0	691.321	869.308	96.675	$\frac{18}{64}$	$\frac{10}{64}$	$\frac{25.40}{64}$	11.701
	PG-PGEP$_{\text{test}}$	1000.0	697.359	873.538	93.474				
f_4	PGEP$_{\text{train}}$	724.178	613.746	641.529	14.855	$\frac{8}{64}$	$\frac{13}{64}$	$\frac{12.80}{64}$	5.307
	PGEP$_{\text{test}}$	714.383	556.348	563.338	27.057				
	PG-PGEP$_{\text{train}}$	1000.0	583.850	690.399	114.965	$\frac{16}{64}$	$\frac{27}{64}$	$\frac{25.22}{64}$	12.877
	PG-PGEP$_{\text{test}}$	1000.0	596.912	593.672	197.874				

appears in fractional form and this indicates the fraction of a chromosome that was consumed by the encoded expression. $s_{\max}$ and $s_{\min}$ are then the lengths of the programs associated with the fittest ($\hat{f}'_{\max}$) and least fit ($\hat{f}'_{\min}$) end-of-run solutions, respectively. Also, $\bar{s}$ and σ_s are the mean and standard deviation of the sizes associated with all the end-of-run programs. The sizes of the end-of-run programs are fixed and thus size specific entries in Table III span both the *train* and *test* rows. If two or more fittest (least fit) solutions are encountered, then the most (least) parsimonious solution is reported in Table III.

$$\text{ENES} = \frac{1}{\text{SA}} \cdot \sum_{k}^{\text{NA}} \text{E}_k \tag{3.12}$$

While Table III reports the quality of the evolved solutions, Table IV emphasizes the computational effort needed to obtain those solutions. The primary measure used is the Expected Number of Evaluations per Success (ENES) (Price et al. 2005). Equation 3.12 defines the ENES where SA is the number of successful attempts achieved, NA is the total number of attempts, and E_k is the number of evaluations incurred on the k^{th} attempt. For the purposes of this work, an attempt is considered successful when a maximized and scaled fitness of at least 999.999 (or equivalently a RMSE of 1.0×10^{-6}) has been achieved. If the problem is extremely difficult, too few evaluations were used, not enough runs were attempted, or if the parameter settings were ineffective, then the ENES may be undefined (Price et al. 2005). Lastly, Table IV also presents the Ratio of Success (RS), which is defined as SA/NA and was recorded for each problem.

Complimenting the end-of-run results in Table III and Table IV are Figure 11, Figure 13, Figure 15, and Figure 17, which contain the convergence curves. These curves are generated

TABLE IV
LISTS THE RS AND THE ENES FOR BOTH PGEP AND PG-PGEP ON ALL FOUR SR BENCHMARK PROBLEMS.

	f_1		f_2		f_3		f_4	
	RS	**ENES**	**RS**	**ENES**	**RS**	**ENES**	**RS**	**ENES**
PGEP	$\frac{50}{50}$	4.42×10^5	$\frac{8}{50}$	1.85×10^7	$\frac{0}{50}$	Undefined	$\frac{0}{50}$	Undefined
PG-PGEP	$\frac{50}{50}$	6.24×10^4	$\frac{3}{50}$	4.91×10^7	$\frac{14}{50}$	9.56×10^6	$\frac{2}{50}$	7.36×10^7

by plotting the mean of the best scaled fitness against the expended number of fitness evaluations. These particular curves convey the mean rate of convergence to a perfect solution, for a particular problem and algorithm combination. Taller and steeper curves indicate that, on average, fitter solutions were achieved with fewer total fitness evaluations. Where as a convergence curve which exhibits a shallower ascent suggests that smaller incremental improvements were obtained. Depending on other reported statistics, vertically shorter curves could be the result of many mediocre solutions or possibly a mix of very poor and very good solutions. The standard deviation can help distinguish between these two scenarios.

Surrounding the convergence curves in Figure 11, Figure 13, Figure 15, and Figure 17 are standard deviation ribbons for the scaled best fitness with respect to the number of fitness evaluations incurred. The total height of the ribbon at any point is equal to two standard deviations, plus one above and minus one below the mean curve. Such ribbons reveal the volatility of each process over the duration of all attempts. Larger deviations from the centered curve indicate a greater degree of inconsistency between runs and therefore the results at that point may be considered less predictable. Conversely, narrower ribbons which straddle the mean curve suggest a greater degree of reproducibility over certain parts of the runs. Flatter edges of the ribbons represent periods of stagnation where no or only negligible differences in fitness were obtained. Following each convergence curve is its corresponding size curve. Figure 12, Figure 14, Figure 16, and Figure 18 reveal the mean and standard deviation of the unsimplified size of the best programs in terms of the number of genes.

3.3 Analysis and Discussion

3.3.1 First Symbolic Regression Problem

Given only the tabulated results in Table III, it does not appear that there is a significant difference between the performance of PG-PGEP and PGEP on the trivial problem f_1. But as Figure 11 shows and Table IV confirms, PGEP requires many more evaluations on average to achieve the same end-of-run performance. Unlike PG-PGEP which quickly achieves perfection, PGEP has a more prolonged and gradual convergence. This is evident in the step-like shape of PGEP's convergence curve and the surrounding standard deviation ribbon in Figure 11. That is, by recognizing that the narrowing of the ribbon in Figure 11 corresponds to the discovery of one or more perfect solutions, it can be seen that PGEP has a comparatively delayed rate of convergence. In addition, similar spreads in Figure 11 were temporarily observed, but PG-PGEP recovers from this more chaotic period relatively quickly.

By considering the sizes of the encoded programs for f_1, it is apparent from Figure 12 that PG-PGEP initially overcompensates with longer expressions but then quickly regulates the sizes. Preferring instead to start off with shorter solutions, PGEP gradually increases sizes

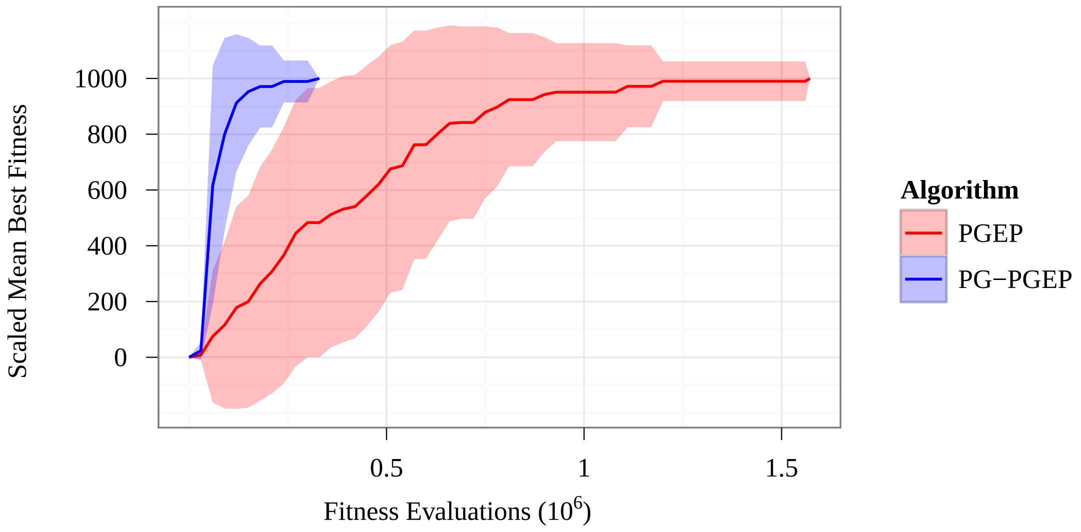


Figure 11. Convergence curves for both the PG-PGEP and PGEP algorithms on f_1.

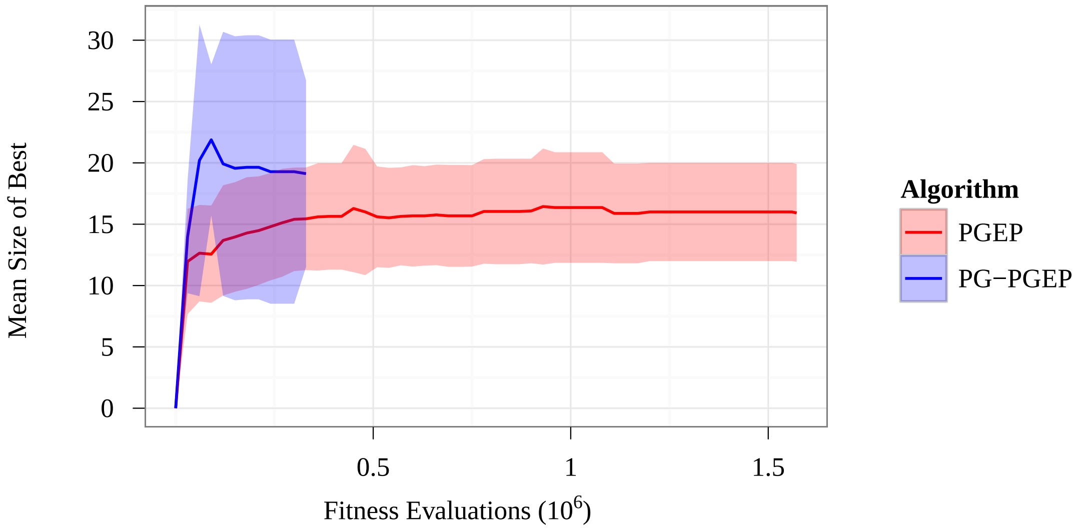


Figure 12. Size curves for both the PG-PGEP and PGEP algorithms on f_1.

as necessary. These contrasting behaviors could easily be explained by the aforementioned differences in the two methods. In the case of PGEP, a more resilient encoding is adopted and this has the potential to hinder innovation causing more slowly growing programs. As for PG-PGEP, the speed in which perfect solutions were obtained may be attributed to its tendency to over represent solutions with a less parsimonious form. That is, PG-PGEP must exploit the cyclic properties of an HMM to generate a perfect solution. Notice that the structure of f_1 is intuitively suited for PG-PGEP as the terms are simple, similar, and linked together with the same function. This repetitive behavior, which although not necessarily optimal with respect to size, seems particularly useful here.

3.3.2 Second Symbolic Regression Problem

Moving onto f_2, Figure 13 reveals some similarities between the two convergence curves. Both PG-PGEP and PGEP exhibit much shallower convergence curves, but also notice that PGEP lags behind PG-PGEP for the first half of the search. However, once the midpoint is passed, PGEP quickly catches up and almost mirrors the progress of PG-PGEP. Focusing on the standard deviation ribbon, it can be seen that after initially stagnating, PGEP becomes extremely innovative. Almost from the start, PGEP surpasses PG-PGEP in terms of volatility and recovers from the initial disparity in fitness. This increased activity is then sustained throughout the remainder of the runs as is shown in Figure 13.

Table III displays a minor improvement in the end-of-run average best fitness $\overline{\hat{f}'}$ obtained for f_2 with PG-PGEP. But Table IV also discloses that PG-PGEP produced a significantly worse RS and thus a significantly larger ENES. In fact, when PGEP was recruited to solve f_2, almost

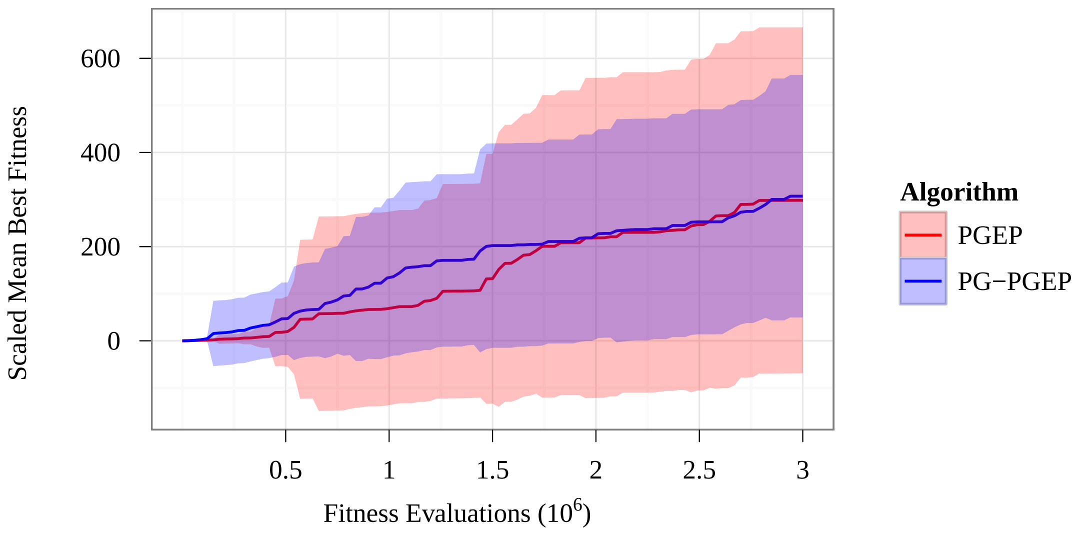


Figure 13. Convergence curves for both the PG-PGEP and PGEP algorithms on f_2.

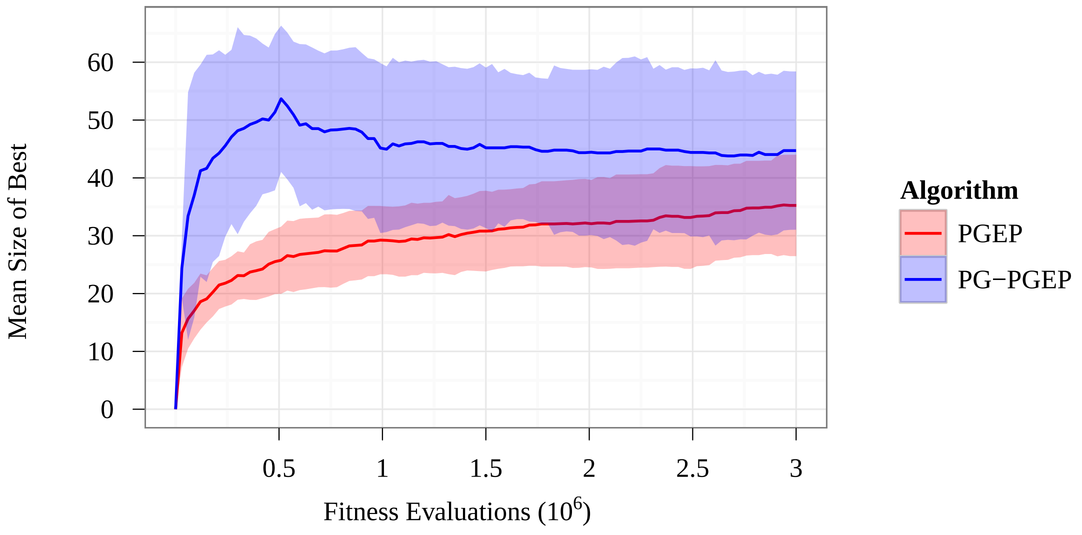


Figure 14. Size curves for both the PG-PGEP and PGEP algorithms on f_2.

three times as many successful attempts were achieved. However, it should be noted that PGEP had a significantly worse case overall best fitness $\hat{f}'_{\text{min}}$ for f_2. The simultaneous cause for both PGEP's improved and inferior results is the larger spread visualized by the standard deviation ribbon in Figure 13 and the larger end-of-run standard deviation $\sigma_{\hat{f}'}$ listed in Table III. It therefore can be seen that in order to obtain peak performance on a more difficult problem with a restricted gene set, some consistency must be sacrificed for the purposes of exploration. Consequently, some abysmal performance will also be incurred.

As mentioned previously, PG-PGEP initially favors longer solutions. Such spiking behavior is observed again in Figure 14, but like before a parsimonious pressure is naturally exerted later on in the search. Also notice that PGEP's increased volatility occurs as the mean size grows and more innovations occur. While PGEP initially hesitates with poorly fit solutions, the observed recovery reflects the spontaneous emergence of good integer coefficients. According to the RS in Table IV, PG-PGEP had a much harder time synthesizing the proper integer coefficients for this particular problem. Although the terms are similar to those of f_1, the way in which solutions must be synthesized is quite different. That is, alternations between multiplicative and additive terms are needed to synthesize good expressions. It would then seem like the synthesis of good integer coefficients is better handled by the existing mechanisms of PGEP. To be more specific, PGEP's ability to maintain exact solutions and manipulate those solutions with crossover and mutation appears more suited for the task of CC. This was especially true in the case of the trailing constant 1.0 in f_2, which tended to have a very low probability of being emitted by PG-PGEP compared to the other terms.

3.3.3 Third Symbolic Regression Problem

Compared to the convergence curves for f_1 and f_2 in Figure 11 and Figure 13, the convergence curves for f_3 in Figure 15 have a noticeably different shape. Here, both PG-PGEP and PGEP quickly obtain mediocre solutions and then the two curves exhibit upward climbs that slowly taper off. The lower end-of-run average $\overline{\hat{f}'}$, the fittest $\hat{f}'_{\max}$ and least fit $\hat{f}'_{\min}$ solutions, and the zero valued RS in Table IV account for the poorer performance of PGEP. Now although PG-PGEP realizes a similarly shaped convergence curve for f_3, the vertically higher location of PG-PGEP's curve indicates an improved overall performance. This is apparent when the end-of-run results of PG-PGEP and PGEP in Table III and Table IV are compared. PGEP's deviation ribbon in Figure 15 demonstrates increasing consistency as the runs progress, but this behavior is premature as the rate of convergence in the associated curve of Figure 15 steadily decreases and does not come close to reaching a maximal point. Furthermore, as Table IV discloses, PGEP never encounters a perfect solution during any of the trials. PG-PGEP on the other hand does not experience premature convergence and the greater degree of volatility actually corresponds to the disparity in fitness which is revealed by the significantly higher RS in Table IV.

Also quite different from the previous problems is how PG-PGEP now generates longer solutions over most of the runs. But while this occurs, the size curve is also not nearly as smooth. Possibly indicating that while the same inherent parsimonious pressure is being exerted, innovative pressures are resisting and pushing back. The parsimony pressures in Figure 16 would then appear to be weaker, allowing for more complex but better solutions to eventually prevail.

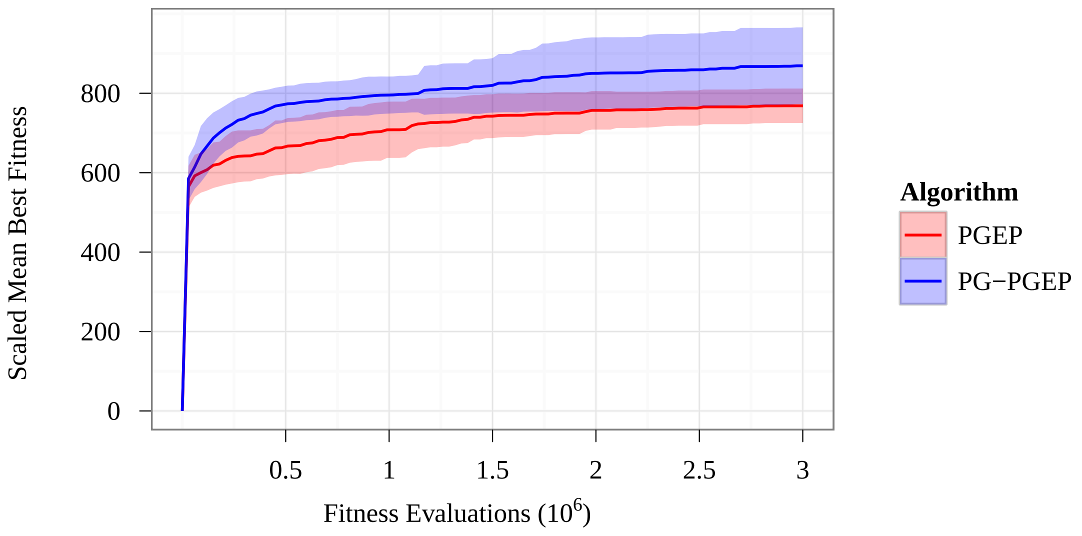


Figure 15. Convergence curves for both the PG-PGEP and PGEP algorithms on f_3.

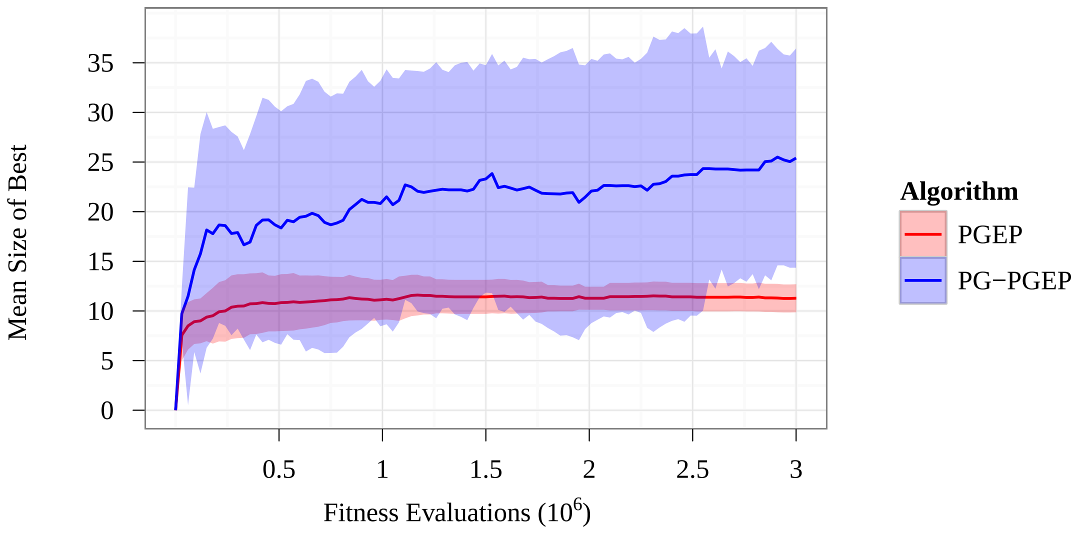


Figure 16. Size curves for both the PG-PGEP and PGEP algorithms on f_3.

Reconciling the behavior of PGEP in Figure 15 with the size curve in Figure 16, it appears that throughout most of the attempts very little variety in solution size was experienced. This suggests that PGEP prematurely converged and that future improvements were not nearly as frequent as crossover was the primary innovator.

3.3.4 Fourth Symbolic Regression Problem

Upon an inspection of Figure 17, it can be seen that PG-PGEP and PGEP exhibit similar behavior for the first half of the search. But in the latter half of the search, the convergence curve for PGEP tapers off and PG-PGEP maintains its rate of convergence. This permits PG-PGEP the opportunity to obtain the better end-of-run results shown in Table III. Except for the worse least fit solution $\hat{f}'_{\min}$, PG-PGEP achieves better end-of-run results in the form of a higher average $\overline{\hat{f}'}$ and a superior fittest solution $\hat{f}'_{\max}$. PG-PGEP also encounters multiple perfect solutions as Table IV discloses and the larger end-of-run spread $\sigma_{\hat{f}'}$ allows. In addition, PGEP's poorer end-of-run performance occurs with a significantly tighter spread $\sigma_{\hat{f}'}$. This is also reflected in the smaller differences between all the end-of-run results ($\overline{\hat{f}'}$, $\hat{f}'_{\max}$, and $\hat{f}'_{\min}$) in Table III.

As more evaluations are expended, an increasing disparity in best fitness between PG-PGEP and PGEP is apparent. While PG-PGEP continues to find improved solutions, the convergence curve for PGEP levels off. The ribbons in Figure 17 also reveals that PGEP almost but not always completely stagnates. Similar to the behavior exhibited by PGEP in f_3, the attempts for f_4 with PGEP indicate fewer successful innovations and suggest that PGEP has a tendency to prematurely converge when undertaking problems which have wave-like shapes or oscillating

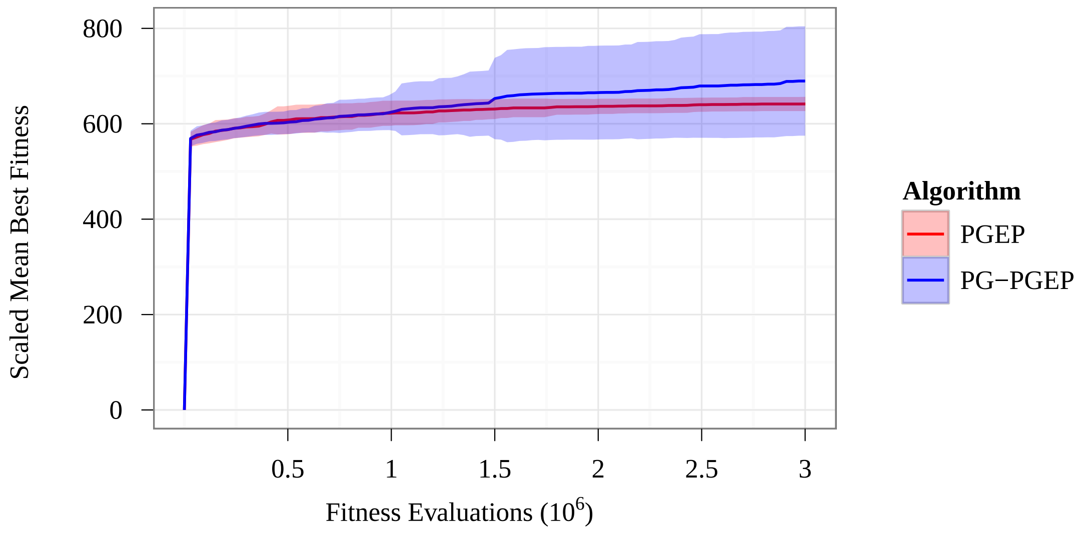


Figure 17. Convergence curves for both the PG-PGEP and PGEP algorithms on f_4.

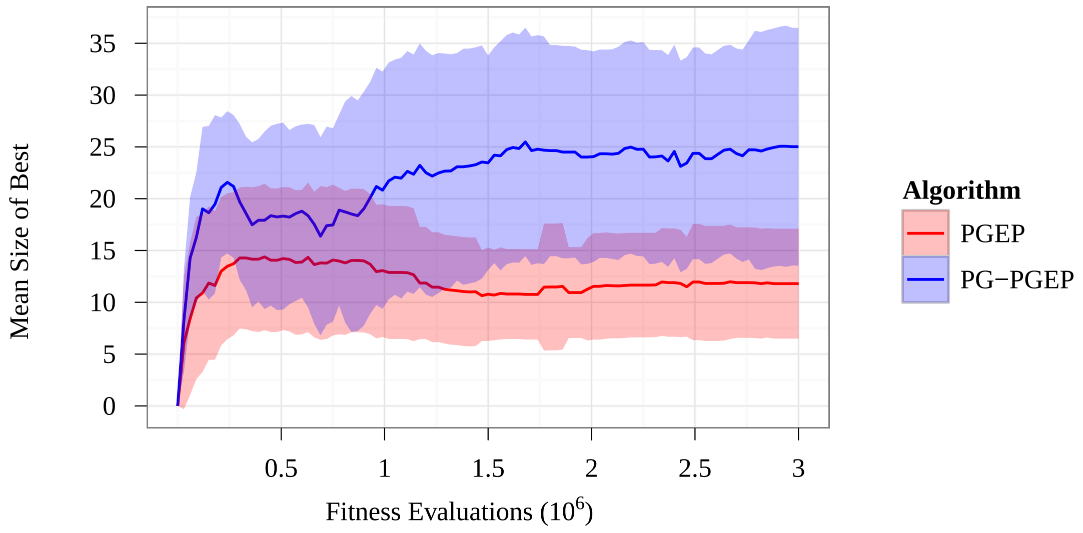


Figure 18. Size curves for both the PG-PGEP and PGEP algorithms on f_4.

surfaces. Another indicator which might also explain the differences in the end-of-run results in the mean size of the best solutions. After PGEP initially experiences an increase in the mean size it steadily decreases and then mostly stabilizes. This gradual decrease in mean size shown in Figure 18 is inconsistent with the behavior of the previous problems. Also notice that the sizes of PGEP's solutions are about half of PG-PGEP's, suggesting that PGEP cannot obtain the longer and more complex programs needed to sufficiently express fitter solutions.

3.4 Summary

A novel evolutionary algorithm which abandoned a well-behaved ontological process in favor a probabilistically oriented one has been proposed in this chapter. Promising preliminary results were acquired on several artificial SR problems. This suggests that a much broader range of problems must be considered as well, and such results would reveal a more complete and diversified picture of PG-PGEP's comparative performance. Interesting results were realized in the form of an increased overall success rate, steeper convergence curves, and noticeable performance gains on two problems, which were demonstrated to be very difficult for the original PGEP algorithm.

4. NUMERICAL OPTIMIZATION OF CONTINUOUS REPRESENTATIONS FOR PROGRAM SYNTHESIS

4.1 Overview

In light of the advantages of PGEP over GEP and GP that where mentioned in Section 2.1, the discrete encoding adopted by PGEP is not well suited to accommodate unknown numerical terms. In other words, given the essential components or minimalistic building blocks which can include numbers, PGEP alone is unable to synthesize mathematical expressions which require the approximation of numerical constants to a high precision. In this chapter, a new type of genotype for PGEP is proposed. This new genotype maintains the linear and fixed-length characteristics of the original PGEP, but replaces the original symbolic representation with a continuous one.

But with this new encoding comes the need for an alternative approach to drive the combinatorial search of PGEP. The robust and powerful real-valued optimizer described back in Section 2.5, namely DE, is recruited to be such a driver. Programs are evolved by interpreting the real-valued parameters as discrete integer values which are in turn converted into linear PGEP chromosomes, and finally, expanded into ETs for evaluation. Numerical constants exist in the same representation, are stored after the encoded PGEP expression, extracted when needed, and then incorporated into the ET. This algorithm has been appropriately called Differentially Evolved Prefix Gene Expression Programming (DE-PGEP) (Cerny et al. 2008b) and will constitute the second and last major contribution of this thesis.

4.2 Motivation

The original motivation behind the development of the PGEP algorithm was to combat the disruptive nature of crossover (Li 2006), i.e., crossover had the tendency to destroy good building blocks. While PGEP mitigated the affects of this phenomenon through the adoption of a prefix notation encoding scheme, which preserves both the hierarchy and proximity of genes in the expression tree. This simple, but fundamental improvement was significant as GEP almost exclusively relied on linear crossover operations to exchange contiguous segments of genetic material between chromosomes (Ferreira 2006). Other independent operators like mutation and rotation were still employed, but the importance of such operators was treated as negligible. This view has been supported by past experiments (Li 2006) which suggest that the probability of applying the mutation operator be less than 5%. In contrast, the total probability of applying the crossover operator(s) was normally 70% or even greater. While several tree-based schemata theorems have reinforced this position (Langdon and Poli 2002), these conclusions have also been contested. It has been argued in (Luke and Spector 1998) that the use of certain operators is dependent on the problem and even the control parameters. More relevant to this work is that of (Ferreira 2002b), which contradictorily claims that mutation can be the most powerful operator in GEP. So although unfortunate, the empirical investigations may still be unavoidable and one cannot blindly assume that crossover will be the dominate operator in any GP-like system.

In Differentially Evolved Prefix Gene Expression Programming (DE-PGEP), the continuous linear crossover and point mutation operators of PGEP are replaced with a weighted vector based differencing approach that uses an interspersed crossover to blend mutational perturbations. The mechanism by which programs will be automatically synthesized is DE and this will also conveniently allow for the creation and refinement of numerical constants. Unlike PGEP, where crossover and mutation are independent, in DE-PGEP these two operations are intimately intertwined. That is, without crossover no mutations would be adopted and similarly, without mutation crossover would be ineffective. Thus, instead of relying on arbitrarily defined operators, each with their own unique control parameters, DE-PGEP only relies on two well defined operators where the interactions between the pair are well studied (Zaharie 2002; 2007).

This proposed method is not exotic, does not introduce any new specialized operators, and does not incur the overhead of an embedded optimizer. While the algorithm still requires that an arbitrary number of constants be allotted to each chromosome, a simple heuristic is given to estimate this quantity. Furthermore, previous research (Koza 1992; Ferreira 2002a; Zhang et al. 2007) confirms that only a small number of unique constants are actually used in the best end-of-run solutions. So the fixed number of constants is not considered a severe limitation that is obtrusive or detrimental.

4.3 Related Research

Many researchers have investigated the feasibility of DE for problems that are discrete or combinatorial in nature (Price et al. 2005). But to the author's best knowledge, there are only three other algorithms in the public domain that explore the feasibility of applying DE to the synthesis of symbolically encoded computer programs. Specifically, DE has been previously recruited to drive the Grammatical Evolution (GE) and Analytic Programming (AP) (Zelinka et al. 2004; 2005) algorithms with the former algorithm being appropriately named Grammatical Differential Evolution (GDE) (O'Neill and Brabazon 2006). More recently though, Geometrical Differential Evolution (GDE*) has been proposed in (Moraglio and Silva 2010) and is actually a generalization of DE for general metric spaces.

The first two areas of research empirically focused on radically different problem domains. While AP had depth and was mostly concerned with the Boolean k-even and k-symmetric problems, GDE had breadth as four diverse problems were investigated. This included: the "Santa Fe Ant Trail", the "Mastermind", the three Boolean multiplexer, and the easy quartic SR function in Equation 3.4. Here, a similar approach is adopted, but various aspects of the algorithm differ and the sole focus is on SR with CC. Moreover, GDE failed to utilize the inherent numerical representation of DE for the purposes of CC. Although AP is used for SR with dynamical systems in (Zelinka et al. 2008), the three CC mechanisms described therein also do not use DE in a non-embedded fashion where programs and constants are symbiotically co-evolved as in DE-PGEP.

Finally, both a theoretical and empirical approach to GDE* was taken in (Moraglio and Silva 2010). Instead of reusing the existing DE operators, two new crossover operators were introduced. These two special operators were specifically designed to search the space of GP program trees with extension rays. The only SR benchmark in (Moraglio and Silva 2010) was the same quartic function as in Equation 3.4. Thus, GDE* also does not explicitly address the problem of CC.

4.4 Differentially Evolved Prefix Gene Expression Programming Algorithm

4.4.1 Ontological Process

Any continuous DE trial vector $\mathbf{u}_{i,g}$ can be easily and consistently constrained by a repair operation. This is accomplished through the use of the simple formula described in Equation 4.1, where the vector notation introduced back in Section 2.5 is used and the usual notational conventions apply for the absolute value and set cardinality operations on the gene set G. Furthermore, % is the common residue (or the modulus operation) over the real numbers.

$$u_{j,i,g} = |x_{j,i,g}| \,\%\, |G| \tag{4.1}$$

The repaired vector then undergoes a discretization operation and thus it can be safely assumed that Equation 4.2 always yields valid indices but not necessarily a valid chromosome. Like in PGEP, invalid chromosomes are themselves not repaired and instead replaced by performing another round of random manipulations (Li 2006).

$$u_{j,i,g} = \lfloor w_{j,i,g} \rfloor \tag{4.2}$$

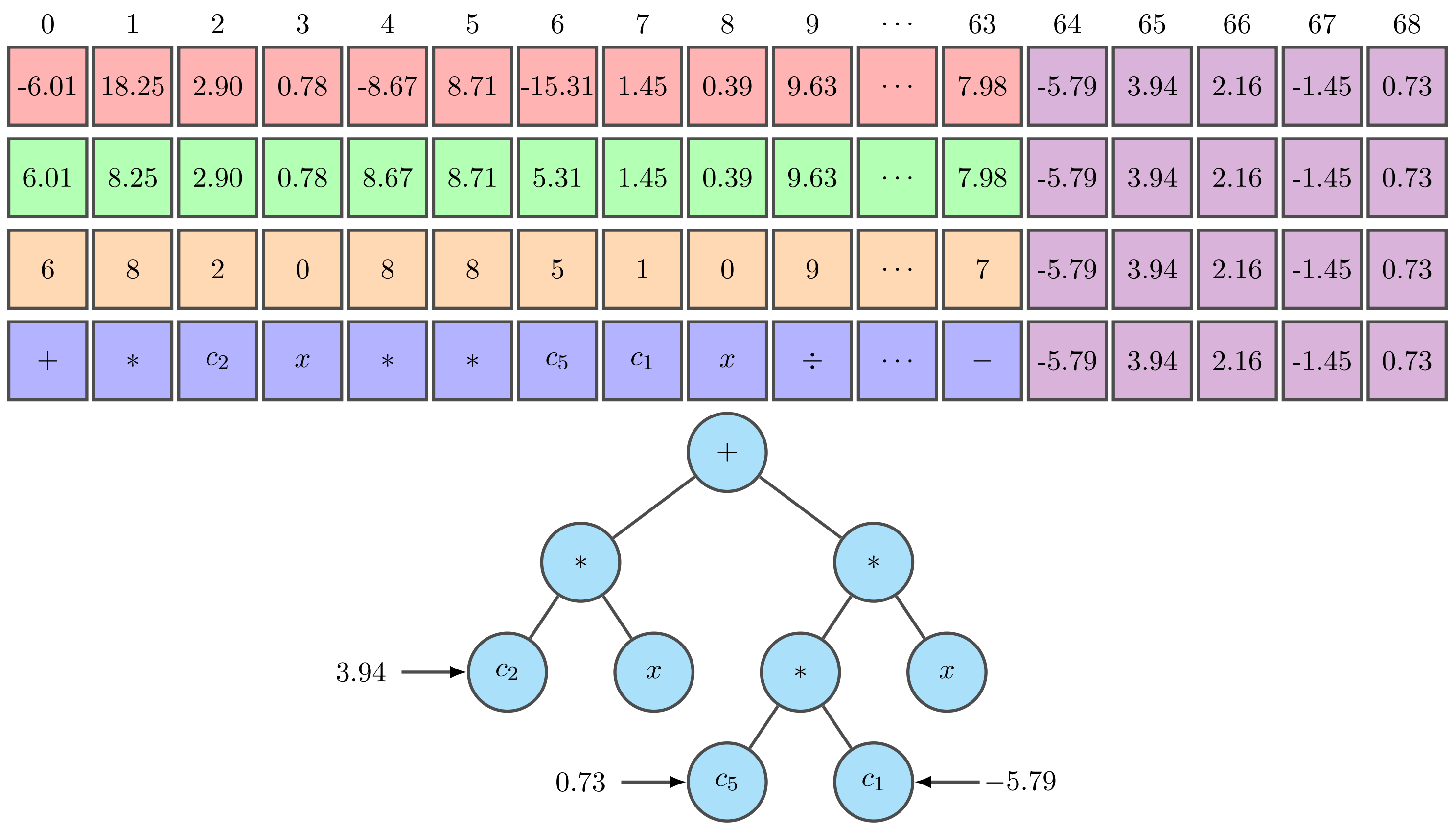
Together, Equation 4.1 and Equation 4.2 effectively mirror and compress the entire search space $\mathbb{R}^D$ into $\mathbb{Z}^D_{|G|}$. The repair applied in Equation 4.1 takes into account magnitude but not direction. Wrapping only happens when $u_{j,i,g} \geq |G|$, but the number of wraps is not limited and it strictly occurs in a uni-directional manner, specifically from high to low.

Each parameter $u_{j,i,g}$ of $\mathbf{u}_{i,g}$ may then be interpreted as the gene with the corresponding zero-based index into G. This exactly defines a PGEP chromosome which can be expanded into an ET using the methods previously described in Section 2.1. Therefore, if the user-defined length of all PGEP chromosomes is l, the dimension D of all DE vectors will be equal to l. As a result, a many-to-one mapping between DE vectors and a linear PGEP chromosome is defined by DE-PGEP. To facilitate a better understanding of the ontological process, Figure 19 demonstrates the steps needed to repair and expand an invalid real-valued vector into a valid PGEP ET through the use of a simple example.

4.4.2 Search Space Definition

Since the user-defined gene set G remains fixed throughout the entirety of a PGEP run, a unique integer index from $[0, |G| - 1]$ must be assigned to every gene belonging to G at the onset of a DE-PGEP run. Obviously, many possible orderings exist, and the optimal ordering is in and of itself a separate problem specific optimization problem entirely. Conceptually, the order of the genes defines the terrain of the fitness landscape, which is something that is unheard of in GEP or even PGEP. Therefore, a random mapping of genes to indices will affect reproducibility as there are $|G|!$ uniquely ordered permutations of G in all.

Conceivably, different orderings could ease or harden the difficulty of a problem. Or, in other words, certain orderings could hasten or inhibit the evolution of fit solutions. For example, the distance between certain genes could be minimized according to some hypothetical criteria such



0	1	2	3	4	5	6	7	8	9	$\cdots$	63	64	65	66	67	68
-6.01	18.25	2.90	0.78	-8.67	8.71	-15.31	1.45	0.39	9.63	$\cdots$	7.98	-5.79	3.94	2.16	-1.45	0.73
6.01	8.25	2.90	0.78	8.67	8.71	5.31	1.45	0.39	9.63	$\cdots$	7.98	-5.79	3.94	2.16	-1.45	0.73
6	8	2	0	8	8	5	1	0	9	$\cdots$	7	-5.79	3.94	2.16	-1.45	0.73
$+$	$*$	c_2	x	$*$	$*$	c_5	c_1	x	$\div$	$\cdots$	$-$	-5.79	3.94	2.16	-1.45	0.73

Figure 19. Demonstrates the steps involved in the ontological process of DE-PGEP. The original, repaired, discretized, encoded, and constant vectors are colored red, green, orange, blue, and violet, respectively. The corresponding ET is colored cyan and centered at the bottom. Gene indices are zero-based and determined by the ordered gene set $G = \{x, c_1, c_2, c_3, c_4, c_5, +, -, *, \div\}$, which has a cardinality of 10. Substitutions of numerical constants for the symbolic placeholders in the ET are depicted by arrows pointing at the applicable leaf nodes. Specifically, c_1, c_2, and c_5 are substituted with -5.79, 3.94, and 0.73, which are the values of the vector parameters at positions 64, 65, and 68.

that perturbations are more successful on average. If this were done adaptively during a DE-PGEP run, then the fitness landscape would be characterized as dynamic. The arity of the included genes should also be considered when constructing an ordered gene set. Since the arity of genes determines the size and shape of the expanded ET, the neighboring genes could determine whether exploration or exploitation is undertaken. Adjacent genes of similar arity are more conducive to exploitation where genes of dissimilar arity would promote exploration as the shape and size of an ET could be radically altered.

The same can also be said about the behavior of the terminals and functions contained within the gene set. This has the potential to foster greater innovation, but could also produce more volatile processes that are unable to find superior vectors. Due to the ramifications that different orderings may pose on the fitness landscape, static orderings of genes are employed. The arrangements used herein are grouped by arity and place genes with similar functional behavior in smaller neighborhoods to ease and promote transitions between genes. That is, more homogeneous as opposed to heterogeneous arrangements of genes are preferred in an attempt to achieve a balance between exploration and exploitation. The exact orderings accompanying the gene sets are divulged later on in Section 4.5 for the experimental results reported in Section 4.6.

4.4.3 Neutral Mutations

One more subtle, but interesting consequence of this mapping is that any value belonging to the half-closed interval $[\lfloor |x_{j,i,g}| \rfloor, \lceil |x_{j,i,g}| \rceil)$ is equivalent with respect to the final outcome. That is, the same index is realized, the same gene is selected, and an identical ET is built. Representations like this have been termed redundant. With redundancy, new and fundamentally different types of mutation are now possible.

Prevalent in nature, neutral mutations produce genotypical differences without affecting phenotypical behavior (Ferreira 2002c; Rothlauf 2006). Or in other words, the genetic material is manipulated without changing the exhibited behavior, and thus not adversely affecting the quality of an individual. Extending this idea to artificial search spaces, formations termed neutral ridges may appear and "drifting" along these ridges can allow access to previously unreachable areas of the search space (Shipman et al. 2000). Conceivably preventing stagnation and increasing performance as there are now more traversable paths to the abundance of optimal solutions residing in the search space (Rothlauf 2006; Shipman et al. 2000).

4.4.4 Constant Creation

Next, in order to accommodate numerical constants, a predefined number of genes $c_1, \ldots, c_m$ are added to T. Each such terminal will represent a distinct constant which resides at or near the end of $\mathbf{u}_{i,g}$, meaning that D will now be equal to $l + m$. Therefore, the constant terminals will have different values depending on the chromosomal context, i.e., constants are local, not global, in scope. Furthermore, when a chromosome is expanded into an ET $\tau_{i,g}$, the value of each constant is retrieved from $\mathbf{u}_{i,g}$. Specifically, for $k = 0, \ldots, m-1$, the value of each c_{k+1} in $\tau_{i,g}$ is $u_{l+k,i,g}$. That is, the terminals $c_1, \ldots, c_m$ are simply placeholders and the actual constants are substituted into the ET at the time of evaluation. For clarity, Figure 19 also demonstrates the substitution of constants into an ET based on the description given above. This scheme not only ensures that a local pool of reusable constants is readily available to each chromosome, but it also allows for the specialization of constants within the context of a single chromosome.

By virtue of Equation 2.10 and Equation 2.12, constants with the same parameter index j influence each other's values. So even though the constants may appear in different positions and quantities in their respective ETs, the interplay between incompatible constants is essential for variation. That is, diverse constants are crucial to the efficient exploration and exploitation of the constants sub-space. With this in mind, one can see that the positions of all genes are determined in a similar manner, and while constants are local in value, the CC of DE-PGEP also works at the global or population wide level.

For future reference, the two sub-vectors of $\mathbf{u}_{i,g}$ will be denoted by $\mathbf{w}_{i,g}$ and $\mathbf{z}_{i,g}$ as these respectively represent the expression vector and the associated vector of constants. The relationship between $\mathbf{u}_{i,g}$ and its two sub-vectors $\mathbf{w}_{i,g}$ and $\mathbf{z}_{i,g}$ is now more formally defined in Equation 4.3.

$$\begin{aligned} \mathbf{w}_{i,g} &= \left(u_{j,i,g}\right), j = 0, \ldots, l-1 \\ \mathbf{z}_{i,g} &= \left(u_{j,i,g}\right), j = l, \ldots, l+m-1 \end{aligned} \tag{4.3}$$

Consequentially, a DE-PGEP population will contain exactly $Np \times m$ evolvable constants. However, only $\mathbf{w}_{i,g}$ is subjected to the mirroring and wrapping of Equation 4.1 and the truncation of Equation 4.1. Constants in $\mathbf{z}_{i,g}$ are therefore not constrained in any manner and are free to increase in magnitude in either the positive or negative directions.

4.4.5 Initialization Revisited

Recalling Equation 2.9, the task of vector initialization must also be revisited for CC. For $\mathbf{w}_{i,g}$ the bounds are simply defined by the user-defined gene set G, but for $\mathbf{z}_{i,g}$ a good set of upper and lower bounds is problem dependent and most likely unknown in advance. Therefore, without any additional knowledge, the recommendation in (Price et al. 2005) is followed and the complete and updated initialization bounds are defined in Equation 4.4.

$$\begin{aligned} w_{j,i,g} &= \text{rand}_j(0,1)|G| \\ z_{j,i,g} &= \text{rand}_j(0,1) \end{aligned} \tag{4.4}$$

In some cases, the initialization bounds just defined can be problematic. If the global optimum is not contained inside the initialization bounds, far initialization has occurred and a great deal of computational effort may be required just to reach the general vicinity of the optimum (Price et al. 2005).

4.5 Experimental Setup

4.5.1 Benchmark Problems

In order to evaluate the performance of DE-PGEP, a collection of SR problems which have become *De facto* benchmarks for CC have been assembled. The aim was to construct a diverse set of benchmarks which includes functions of varying complexity, shape, and type. In addition, SR problems with different constants, both small and large were sought. The quadratic polynomial (f_5) in Equation 4.5 can be found in (Koza 1992; Ferreira 2006), the cubic polynomial (f_6) in Equation 4.6 has been previously investigated in (Zhang et al. 2007; Fernandez and Evett 1998; Li et al. 2004), and finally, the sinusodial function (f_7) in Equation 4.7 has appeared in (Ryan and Keijzer 2003; Keijzer 2003; Costelloe and Ryan 2009; Pennachin et al. 2010). Such citations are far from exhaustive but serve to demonstrate that the selected problems are deeply entrenched in both past and present GP literature.

$$f_5(x) = 2.718x^2 + 3.141636x \tag{4.5}$$

$$f_6(x) = x^3 - 0.3x^2 - 0.4x - 0.6 \tag{4.6}$$

$$f_7(x) = 0.3x\sin(2\pi x) \tag{4.7}$$

Of all the benchmarks, the quadratic polynomial in Equation 4.5 is anecdotally the easiest, and its rational coefficients, from left to right, are similar to Euler's number and π. Although not exceedingly difficult, the cubic polynomial in Equation 4.6 may be considered slightly more difficult than Equation 4.5 as it has a higher order and more constants need to be approximated. Since it has been empirically shown in (Ryan and Keijzer 2003; Keijzer 2003; Pennachin et al. 2010) that Equation 4.7 is more difficult because it becomes increasingly unpredictable as the interval under consideration is widened, two distinct instances of this problem are also investigated herein.

4.5.2 Fitness Functions

In order to facilitate the meaningful and fair comparisons of DE-PGEP to other existing CC techniques, several common fitness functions have been adopted. Specifically, the Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Normalized Root Mean Squared Error (NRMSE) functions, which are defined in Equation 4.8, Equation 4.9, and Equation 4.10, respectively.

$$\hat{f}_{MSE}(\mathbf{u}_{i,g}) = \frac{1}{n}\sum_{j=1}^{n}(p_j - t_j)^2 \tag{4.8}$$

$$\hat{f}_{RMSE}(\mathbf{u}_{i,g}) = \sqrt{\hat{f}_{MSE}(\mathbf{u}_{i,g})} \tag{4.9}$$

$$\hat{f}_{NRMSE}(\mathbf{u}_{i,g}) = \frac{\sqrt{\frac{n}{n-1}\hat{f}_{MSE}(\mathbf{u}_{i,g})}}{\hat{\sigma}_T} \tag{4.10}$$

The terms in Equation 4.8 are as follows: $f_{MSE}(\mathbf{u}_{i,g})$ is the MSE of $\mathbf{u}_{i,g}$, p is the program (chromosome) which $\mathbf{u}_{i,g}$ encodes, n is the size of the training or testing set, p_j is the actual output of p on the j^{th} training or testing case, and t_j is the target output for the j^{th} case. The MSE makes any error measurement independent of the data set size but the units are still squared. The RMSE in Equation 4.9 is then used to express all errors in terms of the same units as the data. It is also compatible with the selection strategy of DE as described in

Equation 2.13. Thus, the RMSE is used as the objective function for the minimization task at hand in DE-PGEP.

Finally, $\hat{\sigma}_T$ in Equation 4.10 represents the sample standard deviation of the output targets in the training or testing sets. Since the training and testing sets are only samples and not populations, a correction for an unbiased estimation with $n - 1$ degrees of freedom is needed. Thus, the total error of the residuals must only be considered an estimate of the true error. The denominator of $\hat{\sigma}_T$ introduced in the formula of the NRMSE normalizes any error measurement, making it independent of the data's dynamic range (Jäske 1996). Consequently, the definition in Equation 4.10 of the NRMSE can only be undefined when every point in the data set is the same, which is extremely rare. Furthermore, compared to the MSE or the RMSE, the NRMSE is more easily consumed when reported as percentage points (i.e., $100\times$NRMSE). In particular, 0% conveys that p is a perfect solution while 100% indicates that p always responds with the mean of the data set (Keijzer 2003; Pennachin et al. 2010). The NRMSE is solely meant for reporting purposes and should increase the comprehensibility of the presented results. In full disclosure, the NRMSE was not used anywhere in DE-PGEP's implementation.

4.5.3 Data Sets

Displayed in Table V are brief summaries of how the training and testing sets were sampled. For future reference, *Id.* introduces an unique identifier for each training and testing set pair, which is denoted by the subscripted lowercase Greek letter Alpha (α). The *Interval*, *Size*, and *Method* columns respectively detail the input intervals on which the samplings occurred, the size of the sampled sets, and the method in which the sets were sampled. In the case of the

TABLE V
DESCRIPTIONS OF THE SAMPLING METHODS USED TO GENERATE THE TRAINING AND TESTING SETS USED FOR BENCHMARKING THE DE-PGEP ALGORITHM.

		Training Set		**Testing Set**	
Id.	**Interval**	**Size**	**Method**	**Size**	**Sampling**
α_1	$[-10, 10]$	10	Random[a]	1000	Random
α_2	$[-10, 10]$	21	+1.0	1000	Random
α_3	$[-0.5, 0.5]$	21	+0.05	1000	Random
α_4	$[-1, 1]$	41	+0.05	2000	Random

[a] Exact training set is borrowed from (Ferreira 2006).

sampling method, *Random* denotes a uniform random sampling over the specified input interval and a number indicates the increment between regularly spaced points over the input interval.

Accompanying Table V is Table VI, which uses descriptive statistics to summarize the sampled training and testing sets. The *Var.* column identifies the independent (Ind.) or dependent (Dep.) variable of the corresponding data set. Each row then summarizes one particular variable for each of the training and testing sets. The *Min.* and *Max.* columns report the extremes of the variable, respectively. The mean and sample standard deviation also appear under the *Mean* and *Std. Dev.* columns, respectively. Upon a close inspection of Table VI, it can be inferred that f_5 must extrapolate well in order to minimize the residual errors on the testing set. The remaining problems, specifically f_6 and f_7, are characterized by interpolation because the independent variables of the testing set are entirely contained by those of the training set. Scatter plots of both the training and testing tests can be seen in Figure 20.

Although there is no complete agreement or clear consensus with regards to the standardized gene sets that should be used in each of the benchmark problems, the dominant operators appearing in the respectively cited works are generally adopted. The ordered gene sets appearing in Equation 4.11 and Equation 4.12 are those that will be utilized throughout the remainder of this chapter.

$$G_1 = \{x, c_1, \ldots, c_m, +, -, *, \div\} \tag{4.11}$$

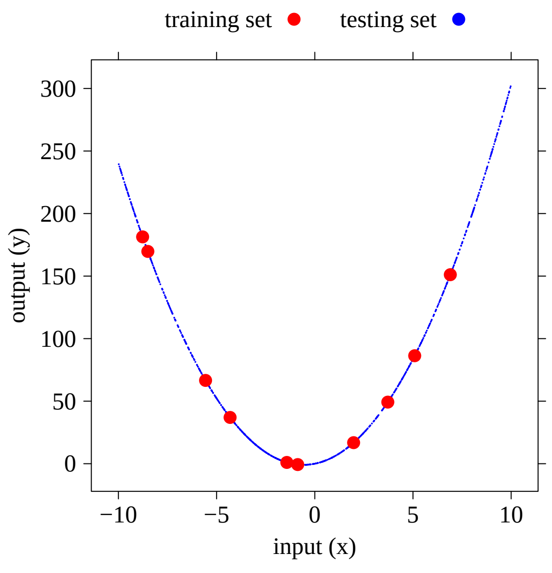
$$G_2 = \{x, c_1, \ldots, c_m, \sin, \cos, \text{sqrt}, \ln, \exp, +, -, *, \div\} \tag{4.12}$$

Besides the common mathematical operators, $+$, $-$, $*$, and $\div$[1], there are several new operators which may require some clarification. First off, in Equation 4.12, sin and cos represent the trigonometric sine and cosine operators. Furthermore, exp, sqrt, and ln in Equation 4.12 are the operators which represent the exponential, square root[2], and natural logarithm[3] functions,

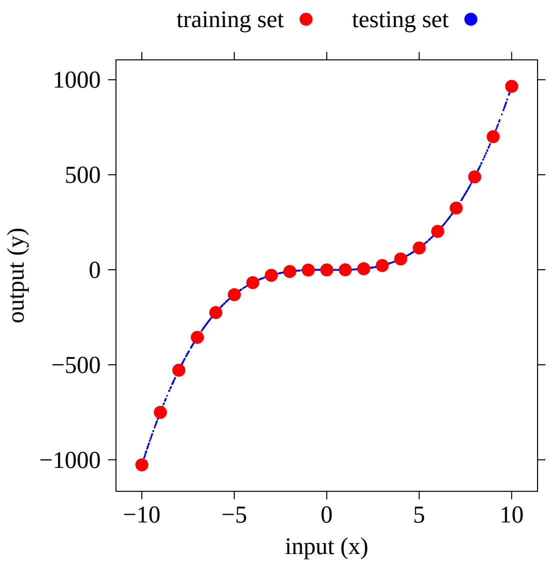
[1]The protected division function guards against divisions by zero by returning a special *undefined* value which defines the output of the program and indicates that the chromosome should be assigned the worst possible fitness.

[2]In order to avoid the use of imaginary numbers, the square root operator is protected and re-defined as follows: for all $x \in \mathbb{R}$, $\sqrt{x} = \sqrt{|x|}$.

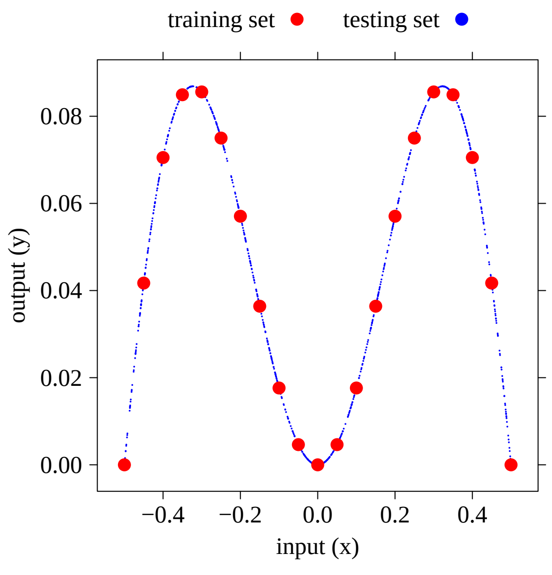
[3]The protected natural logarithm operator guards against undefined behavior and is reasonably re-defined as follows: for $x \neq 0$, $\ln(x) = \ln(|x|)$ and for $x = 0$, $\ln(x) = -745$.



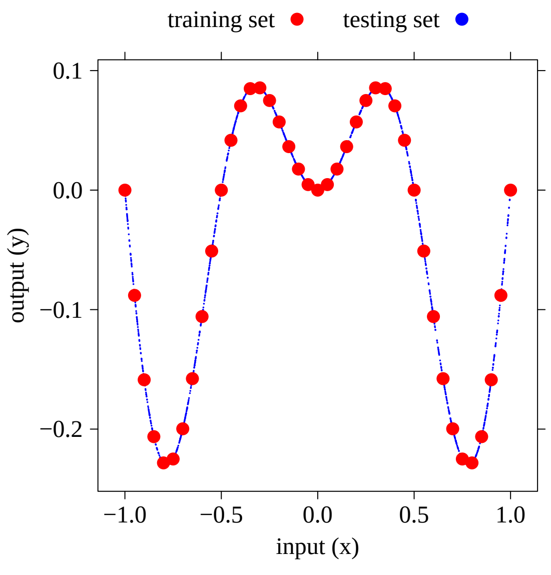
(A) Quadratic polynomial



(B) Cubic polynomail



(C) Narrow sinusoidal function



(D) Wide sinusoidal function

Figure 20. The scatter plots, appearing in a clockwise direction from the top left to the bottom left, are for the quadratic and cubic polynomials, and the two sinusoidal functions summarized in Table VI.

TABLE VI

STATISTICALLY SUMMARIZES THE TRAINING AND TESTING SETS USED FOR BENCHMARKING THE DE-PGEP ALGORITHM.

		Training Set				**Testing Set**			
Id.	**Var.**	**Min.**	**Max.**	**Mean**	**Std. Dev.**	**Min.**	**Max.**	**Mean**	**Std. Dev.**
α_1	Ind.	−8.767	6.901	−1.175	5.567	−9.981	9.974	−0.202	5.819
	Dep.	−0.677	181.364	75.864	69.090	−0.901	301.726	91.411	82.800
α_2	Ind.	−10.000	10.000	0.000	6.205	−9.989	9.982	−0.126	5.748
	Dep.	−1026.600	965.400	−11.600	442.630	−1023.339	960.266	−23.176	373.389
α_3	Ind.	−0.500	0.500	0.000	0.310	−0.500	0.500	−0.006	0.292
	Dep.	0.000	0.087	0.045	0.032	0.000	0.087	0.046	0.029
α_4	Ind.	−1.000	1.000	0.000	0.599	−0.999	0.996	0.004	0.588
	Dep.	−0.228	0.086	−0.046	0.111	−0.230	0.087	−0.049	0.111

respectively. Finally, x is the only input variable, and $c_1, \ldots, c_m$ are the evolvable constant terminals introduced back in Section 4.4.4 where m may vary between experiments.

In Table VII, various control parameters are presented and identified by the appropriate column header. The first column, *Id.*, contains an identifier that represents a unique control parameter combination and is denoted by a subscripted lowercase Greek letter Beta (β). A few of these parameters are problem specific (e.g., gene set), some are borrowed from other sources (e.g., gene, training, or testing sets), but the rest are arbitrary and were determined with little effort to be empirically good. In Table VII, the remaining columns, from left to right, correspond to the following parameters: the gene set (G), the PGEP chromosome length (l), the number of evolvable constants (m), the dimension of a DE vector (D), the DE population size (Np), the Differential Mutation operator (DM Operator), the scaling factor (F), the uniform crossover probability (Cr), the error threshold (ϵ) expressed in terms of the RMSE, and the maximum number of generations (g_{max}). All of these parameters have been previously introduced in Section 2.5, Section 2.1, and Section 4.4.4, and thus should not require any further explanation.

TABLE VII
AN OVERVIEW OF THE CONTROL PARAMETER COMBINATIONS USED DURING TRAINING FOR THE BENCHMARK TRIALS OF THE DE-PGEP ALGORITHM.

Id.	G	l	m	D	Np	**DM Operator**	F	Cr	ϵ	g_{max}
β_1	G_1	32	5	37	25	DE/rand/1	0.2	0.1	0.01	10^5
β_2	G_1	64	5	74	25	DE/rand/1	0.2	0.1	0.01	10^5
β_3	G_2	64	10	74	25	DE/rand/1	0.4	0.1	0.01	10^5

4.6 Empirical Results

Table VIII contains an overview of the fitness results obtained from performing the described experiments. Due to the stochastic nature of DE-PGEP, 40 independent attempts were made per benchmark. Each row presents the results obtained for a single benchmark and it is divided in half where the upper entries reflect the results obtained during training and the lower entries express the generalization results over the testing set. The first column, *Env.*, indicates the environment of the experiments, i.e., a tuple of the target function, the training and testing sets, and the control parameter combination. The *Fitness of Best Chromosomes* header spans the remaining columns, which contain quantities describing the best fitness values obtained in one or all attempts during training and testing. These columns are vertically partitioned into two groups. The quantities on the left are expressed in terms of the RMSE and are useful for making comparisons with many other works. On the right, quantities are expressed as percentages in terms of the NRMSE as such measurements are more meaningful and easily interpreted. Under the NRMSE, the rows of zeros for the quadratic and cubic polynomials are correct and can be attributed to both the accuracy of the evolved expressions and the larger sample standard deviations of the corresponding data sets reported in Table VI.

Any *Min.* column refers to the chromosome with the best fitness value obtained in and over all trials during training, i.e., it had the lowest error on the training set thus indicating that it was the fittest chromosome overall. Similarly, every *Max.* column refers to the chromosome which achieved the worst overall fitness during training, but was still the best fitness obtained in a single run, i.e., it was the chromosome with the highest error on the training set signifying

TABLE VIII

AN OVERVIEW OF THE BEST END-OF-RUN FITNESS VALUES OBTAINED WITH DE-PGEP OVER ALL 40 INDEPENDENT EXPERIMENTAL TRIALS ON THE QUOTED BENCHMARK PROBLEM. ALL RESIDUAL ERRORS ARE REPORTED IN TERMS OF AND GROUPED BY THE UNSCALED RMSE AND THE NRMSE AS PERCENTAGE POINTS. SINCE DE-PGEP ATTEMPTS TO MINIMIZE THE RMSE, SMALLER VALUES IN ALL COLUMNS (EXCEPT FOR STD. DEV.) INDICATE FITTER EXPRESSIONS ON THE CORRESPONDING SR PROBLEM. TIGHTER STANDARD DEVIATIONS ARE A GOOD INDICATOR OF GREATER REPRODUCIBILITY AND CONSISTENCY OVER ALL TRIALS.

	Fitness of Best Chromosomes							
	Unscaled RMSE				**NRMSE in Percentage Points**			
Env.	**Min.**	**Max.**	**Mean**	**Std. Dev.**	**Min.**	**Max.**	**Mean**	**Std. Dev.**
(f_5, α_1, β_1)	8.238×10^{-4}	9.907×10^{-3}	7.433×10^{-3}	2.452×10^{-3}	0	0	0	0
	9.140×10^{-4}	1.494×10^{-2}	9.089×10^{-3}	3.534×10^{-3}	0	0	0	0
(f_6, α_2, β_2)	1.866×10^{-3}	6.172×10^{-2}	9.952×10^{-3}	8.785×10^{-3}	0	0	0	0
	1.658×10^{-3}	5.872×10^{-2}	1.675×10^{-2}	4.656×10^{-2}	0	0	0	0
(f_7, α_3, β_3)	3.245×10^{-3}	2.475×10^{-2}	9.674×10^{-3}	3.028×10^{-3}	10	78	31	10
	3.386×10^{-3}	4.477×10^{-2}	1.497×10^{-2}	1.099×10^{-2}	11	150	50	37
(f_7, α_4, β_3)	9.253×10^{-4}	1.093×10^{-1}	1.355×10^{-2}	1.692×10^{-2}	1	100	12	15
	9.018×10^{-4}	1.111×10^{-1}	2.203×10^{-2}	3.773×10^{-2}	1	100	20	34

it was the least fit best end-of-run chromosome encountered. The two remaining columns, *Mean* and *Std. Dev.*, simply report the mean and sample standard deviation of the best fitness values recorded at the end of and over all 40 independent trials. Since the entries in the *Min.* column are based only on the training set, one will immediately be able to determine whether or not the fittest chromosome has overfitted the training data.

Complementing the fitness oriented results in Table VIII, Table IX reports additional results that are concerned with the structural complexity and composition of the best end-of-run ETs. The first column, *Env.*, has the same meaning as that in Table VIII. However, unlike Table VIII the rows are not composed of training and testing parts because the quantities reported in Table IX are data set agnostic. The *Size of Expression Tree* column presents the size(s) of the corresponding expression tree(s). *Min.* and *Max.* are the sizes of the overall best and worst end-of-run programs. *Mean* is the average size of the best end-of-run programs evolved over all trials, and similarly, *Std. Dev.* is the sample standard deviation of this size.

Quantities representing the number of constants occurring in the ETs are also presented in Table IX under the *Ratio of Constants* heading. That is, *Min.* and *Max.* divulge the ratios of unique to total constants utilized in the ETs of the overall fittest and least fit best end-of-run solutions, respectively. The remaining columns, specifically *Mean* and *Std. Dev.*, respectively report the mean and sample standard deviation of the unique constants and total constants of the best end-of-run programs over all trials. These last two quantities are always displayed in fractional form and are not simplified so that the numerator and denominator can be viewed independently.

TABLE IX
SUMMARIZES VARIOUS OTHER ASPECTS OF THE BEST END-OF-RUN INDIVIDUALS FROM ALL 40 INDEPENDENT EXPERIMENTAL TRIALS INCLUDING: THE SIZE OF EACH ET AND THE RATIOS OF UNIQUE CONSTANTS TO TOTAL CONSTANTS.

	Size of Expression Trees				**Ratio of Constants**			
Env.	**Min.**	**Max.**	**Mean**	**Std. Dev.**	**Min.**	**Max.**	**Mean**	**Std. Dev.**
$(f_5,\, \alpha_1,\, \beta_1)$	17	13	19.750	6.590	$\frac{3}{5}$	$\frac{3}{4}$	$\frac{3.950}{6.575}$	$\frac{0.815}{2.601}$
$(f_6,\, \alpha_2,\, \beta_2)$	43	55	32.950	12.702	$\frac{5}{16}$	$\frac{5}{24}$	$\frac{4.250}{11.350}$	$\frac{0.781}{5.147}$
$(f_7,\, \alpha_3,\, \beta_3)$	23	14	27.550	10.853	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{4.075}{5.750}$	$\frac{1.542}{3.160}$
$(f_7,\, \alpha_4,\, \beta_3)$	10	1	29.900	14.317	$\frac{2}{2}$	$\frac{1}{1}$	$\frac{4.725}{6.850}$	$\frac{1.797}{3.534}$

4.7 Analysis

Even though DE-PGEP utilizes a uniform crossover operator, it is significantly different from most crossover operators employed by GP, GEP, or PGEP. As a refresher, the GP crossover operators typically swap sub-trees between two tree-based chromosomes, and both GEP and PGEP linear crossover operators exchange consecutive segments of genes between two chromosomes. However, in DE-PGEP, the uniform crossover operator is discrete in nature and only adopts neighboring parameters by chance if Cr is low. Furthermore, DE-PGEP does not always swap unaltered genetic material. The genetic material swapped by uniform crossover is either an unperturbed or perturbed version of itself. So, from a GP, GEP, or PGEP perspective, the uniform crossover operator with a low application probability more closely resembles that of point mutation than traditional crossover. In some sense then, Cr can be viewed as the probability of accepting a mutation and F can be seen as the magnitude of that mutation. Yet, the magnitude of mutation has no real parallel in GP, GEP, or PGEP as point mutations are traditionally restricted to genes with similar arity.

Additionally, the uniform crossover operator only manipulates one individual directly where a single crossover in GP, GEP, or PGEP almost always produces two new offspring. The product of a mutation and crossover operation here, in DE-PGEP, can be much more subtle and not nearly as obvious. Due to the presence of neutral mutations, there is no guarantee that any immediate affects of the operations will be noticeable. It may not be until several generations in the future, after more and more perturbations have been compounded, that a noticeable difference in the phenotype emerges (Rothlauf 2006). Finally, even though the real-valued representation of DE increases the size of the search space, "flattening" occurs because of redundancy and vast uniform plateaus may appear (Rothlauf 2006). The transformed landscapes, which have a "staircase" like shape, may then explain the viability of DM as a search space operator for DE-PGEP.

Upon a close inspection of the control parameter combinations in Table VII, it can be seen that Cr is always equal to 0.1 (i.e., 10%). This was empirically determined and anything above 0.3 (i.e., 30%) was consistently useless. Thus, when $D = 74$, approximately only 7 to 8 parameters are independently adopted into the trial vector. With a Cr above 0.3, at least 23 parameters are mutated, which apparently yields too much instability. However, since a vector encodes a variable length program, short programs will be affected less and possibly preserved in their entirety. In this case, the majority of mutations will then occur in the downstream non-coding region, which is composed of introns. Intuitively, a higher Cr paired with a moderate F, will generally yield more genotypical differences. Thus, as Cr increases, more phenotypical differences can occur, presumably resulting in a more chaotic and less successful search. The parameter combinations must then try to maintain a balance between chaos and order. If not, the search will deteriorate and good results like those reported here will be illusive.

While introns do not directly contribute to the fitness of the vector, these parameters can be used to alter other vectors by way of DM. That is, other vectors may encode longer programs and therefore the introns can influence these other vectors without being explicitly enabled. This is very different from GEP or PGEP where introns are present, but only activated or deactivated when an actual crossover or mutation occurs. Together, the combination of introns and discretization actually gives DE-PGEP a "doubly" redundant encoding. Also, it was continuously observed that as the gene set increased in size, F needed to be increased to compensate for the larger search space. This is understandable as a larger scaling factor is required to cycle through the available genes and find useful or good innovations.

Through a close inspection of the results presented in Table VIII and Table IX, some very interesting observations about the stochastic behavior of the DE-PGEP algorithm can be seen:

- From both a training and testing perspective, DE-PGEP is very consistent in the results it produces. Combined with the large number of independent runs and the very small sample standard deviations, one can conclude that the final results are reproducible and not just coincidental.

- Furthermore, the least fit solutions are generally not exceedingly bad by any means, i.e., the overall worst solutions are in most cases good approximations as well, but they are still not as precise as the fittest solutions.
- As the number of employed operators increased, it was empirically determined that m needed to be increased too. Although not conclusive, a simple heuristic based on such experiences and these tabulated results is made: m should be approximately equal to the number of operators plus the number of input variables. The additional diversity afforded by a larger pool of constants seems helpful when more functionality distinct operators are made available. However, significantly more runs with additional control parameter combinations, possibly using DOE, must be undertaken in order to accept such a hypothesis.
- On average, for the best end-of-run ETs, DE-PGEP consistently uses only about one-half to five-eighths of the parameters in a DE vector. Furthermore, the sizes of the fittest solutions in Table IX are quite compact and do not suggest the explosive growth of bloat[1], which is commonly experienced with GP. It is possible, due to the interspersed or fragmented nature of the uniform crossover operator, that a bias against longer program exists and naturally exerts a parsimonious pressure.
- The means reported in Table IX show that the best end-of-run results obtained with DE-PGEP almost always reused the small number constants available to each member of the

[1]Bloat is characterized as the accumulation of useless operations, which are either eventually canceled out (e.g., $x - x$) or do not alter the output of a program (e.g., $1 * x$)

population. While not conclusive, other CC techniques, particularly (Ryan and Keijzer 2003), have reported similar results.

- Only a single anomalous program was evolved during one of the trials for (f_7, α_4, β_3). This is especially apparent in the last two rows of Table VIII where 100 is reported twice in the *Max.* column under the NRMSE. The ET of size unity was composed of exactly one constant as can be verified by examining Table IX. Consequentially, this program always emitted the same value, which indicates that DE-PGEP evolved an approximation to the arithmetic mean of the training set. This particular program was an exception and it did not occur during any other trial. Poor initialization, as discussed in Section 4.4.5, may have been the cause of this rather interestingly anomaly.

- While it was not explicitly reported anywhere in Table IX, the best end-of-sun solutions always used at least one unique constant. In other words, none of the best end-of-run solutions ignored the evolvable constants. Thus, in the problems studied herein, DE-PGEP always utilizes the proposed CC technique.

- Irrespective of the benchmark problem's difficulty and the other control parameters, Np always remained the same. With DE-PGEP, surprisingly small population sizes were used. Normally, in GP, GEP, or PGEP, population sizes vary but are at least between 500 and 1000 in order to avoid premature convergence. Coincidentally, the one-to-one survivor selection strategy of DE is conveniently aligned with the NSM improvement originally proposed in (Gustafson et al. 2005) for GP and revisited in (Costelloe and Ryan 2009). While (Gustafson et al. 2005) showed a statistically significant improvement for NSM alone, (Costelloe and Ryan 2009) failed to corroborate those results. The results in Table VIII clearly reveal that premature convergence was not experienced, possibly because of this NSM-like selection strategy built-in to DE.

In addition to the end-of-run results presented in Table III, plots of the mean best results obtained over all 40 independent experimental trials are presented in Figure 21, Figure 22, Figure 23, and Figure 24 for the environments (f_5, α_1, β_1), (f_6, α_2, β_2), (f_7, α_3, β_3), and (f_7, α_4, β_3), respectively. Fitness curves make an appearance in Figure 21A, Figure 22A, Figure 23A, and Figure 24A, where the vertical axes are expressed in terms of the logarithmically scaled mean of the best NRMSE in percentage points over all trials and generations. For easy reference, dashed horizontal lines appear in each fitness curve and signify the logarithmically scaled mean of the best NRMSE in percentage points on the testing set over all trials and for each particular environment. Next, constant curves are presented in Figure 21B, Figure 22B, Figure 23B, and Figure 24B. Two distinct constant curves appear in each figure where the blue and red curves respectively represent the mean of the unique and total number of constants in the fittest programs over and throughout the entirety of all trials. At any fitness evaluation the total number of constants should always be greater than or equal to the unique number of constants. Lastly, size curves reveal the mean of the fittest programs' sizes and are shown in Figure 21C, Figure 22C, Figure 23C, and Figure 24C. Surrounding each of the above mentioned curves are standard deviation ribbons for each quoted quantity. One sample standard deviation appears above each mean curve while another appears below it.

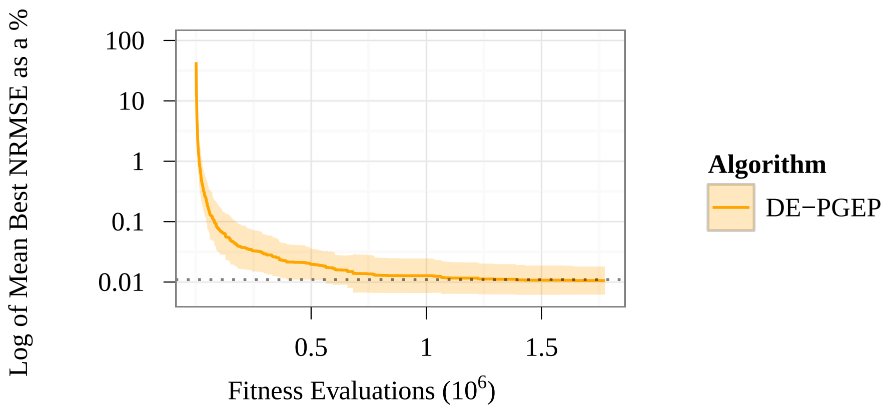
4.8 Example Solutions

4.8.1 Quadratic Polynomial

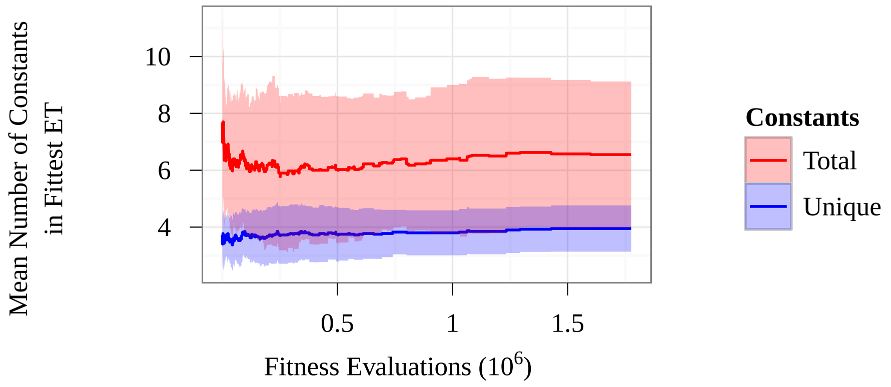
Presented in its unsimplified form, Equation 4.13 is the best program evolved for Equation 4.5 with a size of 17 and training fitnesses of 8.238×10^{-4} and 0% in terms of the RMSE and NRMSE, respectively. Once simplified into Equation 4.14, it is immediately apparent that this program is a very good approximation. In fact, it is almost a perfect solution even though the testing fitness is slightly worse at 9.140×10^{-4} in terms of the RMSE. However, a NRMSE of 0% is still reported for the testing set, which confirms the goodness of fit.

Instead of evolving just two constants with exact values, DE-PGEP utilizes three out of the five uniquely available constants and reuses one of these exactly three times. Although it may not be the most intuitive solution, the idea of using many similar terms to build up good solutions is reasonable and certainly achieves the desired result. For completeness, it should be noted that this particular solution was obtained with (f_5, α_1, β_1).

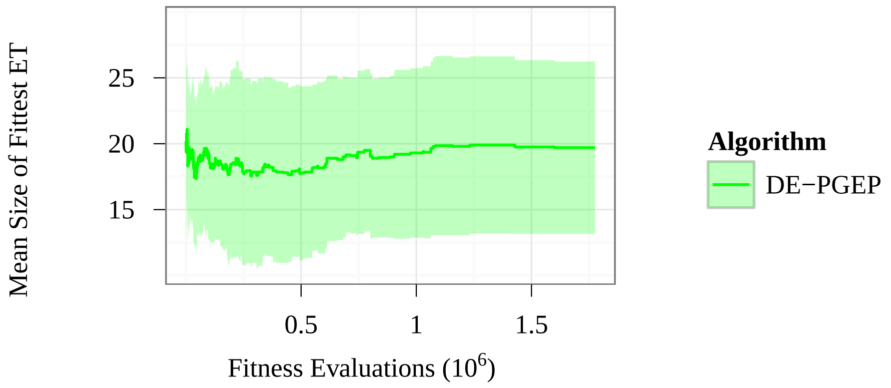
$$
\begin{aligned}
f_5(x) \approx \hat{f}_5(x) &= (x/0.439254) + \big((1.165761 + 1.165761)* \\
&\quad (x * 1.165761) * x\big) + (x * 0.864893) && (4.13) \\
&= 2.717997 * x^2 + 3.141480 * x && (4.14)
\end{aligned}
$$



(A) Plot of the fitness evaluations against the log scaled mean of the best NRMSEs.

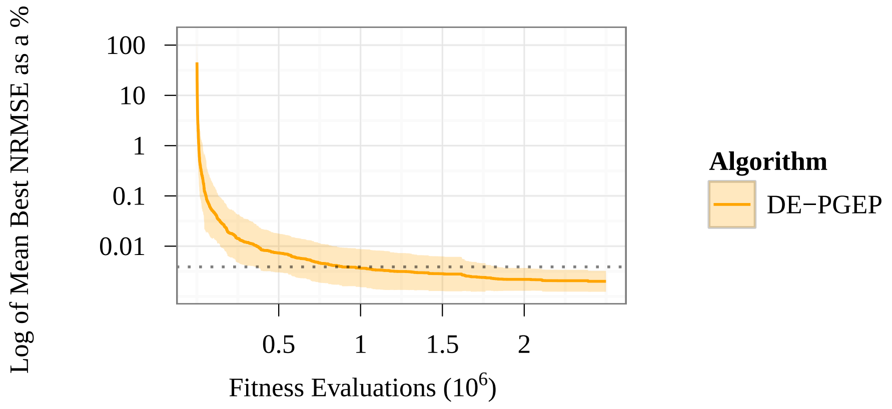


(B) Plot of the fitness evaluations against the mean number of constants in the fittest ETs.

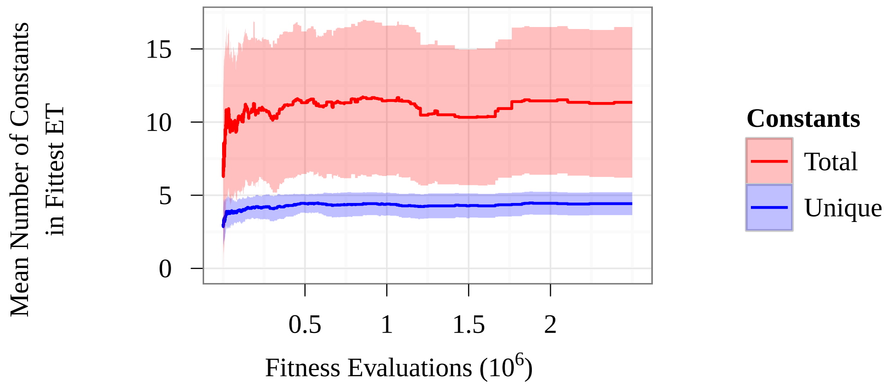


(C) Plot of the fitness evaluations against the mean size of the fittest ETs.

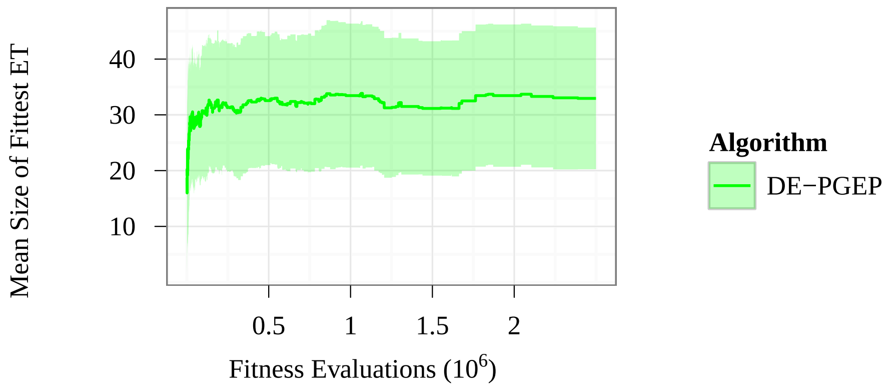
Figure 21. Various plots reporting, from top to bottom, the NRMSE, constant counts, and ET sizes over all 40 independent trials for the quadratic polynomial benchmark.



(A) Plot of the fitness evaluations against the log scaled mean of the best NRMSEs.



(B) Plot of the fitness evaluations against the mean number of constants in the fittest ETs.



(C) Plot of the fitness evaluations against the mean size of the fittest ETs.

Figure 22. Various plots reporting, from top to bottom, the NRMSE, constant counts, and ET sizes over all 40 independent trials for the cubic polynomial benchmark.

4.8.2 Cubic Polynomial

Appearing in Equation 4.15 is the best solution attained with (f_6, α_2, β_2). In its original and unmodified form, the size of the expression tree is rather large, 43 to be exact, and the composition of the expression is somewhat irregular. That is, instead of utilizing the available constants in a strictly conservative manner, all five unique constants appear together a total of 16 times throughout the entire ET. However, two of the five constants are used only once and the rest are used at least three times and at most six. Still, by observing the simplified solution in Equation 4.16, it can be easily seen that this solution is very good with a training RMSE of 1.866×10^{-3} and a training NRMSE of 0%. Interestingly enough, the testing RMSE of 1.685×10^{-3} is slightly better but only negligible as is shown by the testing NRMSE of 0%.

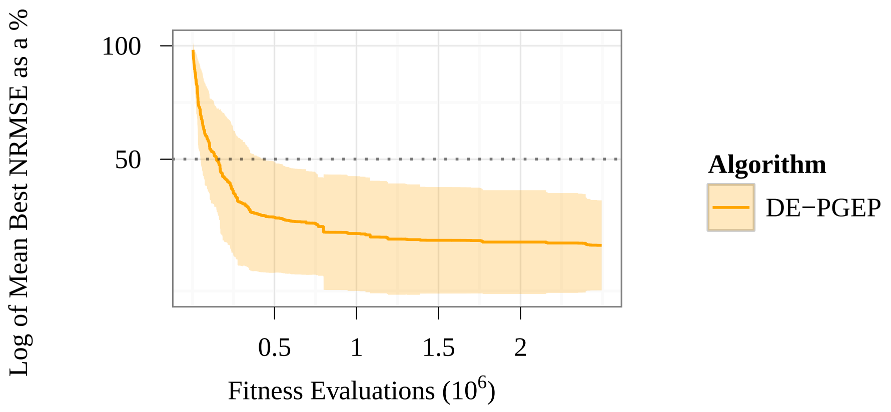
$$\begin{aligned} f_6(x) \approx \hat{f}_6(x) = & \left(c_4 * x\right) + c_2 + \Bigg(\bigg(c_1 + \Big(\big(c_1 - x\big) * x\Big) + c_4 + c_1 + x + \big(c_1 - c_1\big) + \\ & \left(c_5 * c_2\right) + c_3 + \Big(\big(c_2/(c_4 - c_1)\big) - x\Big)\bigg) * \Big(c_2 - x\Big)\Bigg) - c_4 \\ & \text{where } c_1 = 0.349331, c_2 = -0.049276, c_3 = 0.017816, \\ & c_4 = 0.505355, c_5 = 0.361927 \qquad (4.15) \\ = & \; x^3 - 0.300055 * x^2 - 0.400035 * x - 0.598397 \qquad (4.16) \end{aligned}$$

4.8.3 Narrower Sinusoidal Function

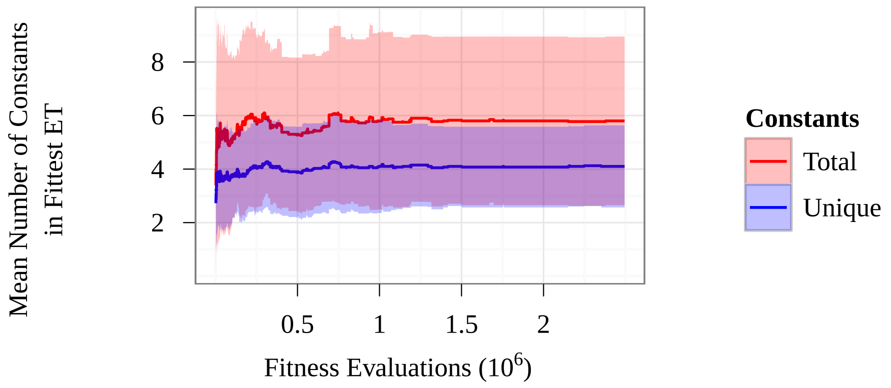
Shifting attention to the next benchmark problem, the best solution discovered for Equation 4.7 is presented in Equation 4.17. Unmodified, the original solution has a size of 23, a training RMSE of 3.245×10^{-3}, and a training NRMSE of 10%. Initially, it appears that this function does not even closely resemble that of the target function. But upon evaluation on the testing set, a RMSE and NRMSE of 3.386×10^{-3} and 11% were achieved, respectively. Here though, all five constants were used exactly once, which seems to suggest that the constants were highly specialized, making them unusable in different contexts. No matter, this solution is still a remarkably good approximation given its structure and composition. A simplified and possibly more understandable version of this solution is presented in Equation 4.18.

$$\begin{aligned} f_7(x) \approx \hat{f}_7(x) = {} & \sqrt{\sin(3.899029 * 10^{84})} * \ln(x + x) * \sin(x * x) * \\ & \Big(0.045098 - \exp\Big(\sqrt{x} / \sin\big(\cos(2.960171 * 10^{72})\big)\Big)\Big) \qquad (4.17) \\ = & \Big(0.998604 * \ln(2 * x) * \sin\big(x^2\big)\Big) * \Big(0.045098 - \exp\big(1.193832 * \sqrt{x}\big)\Big) \qquad (4.18) \end{aligned}$$

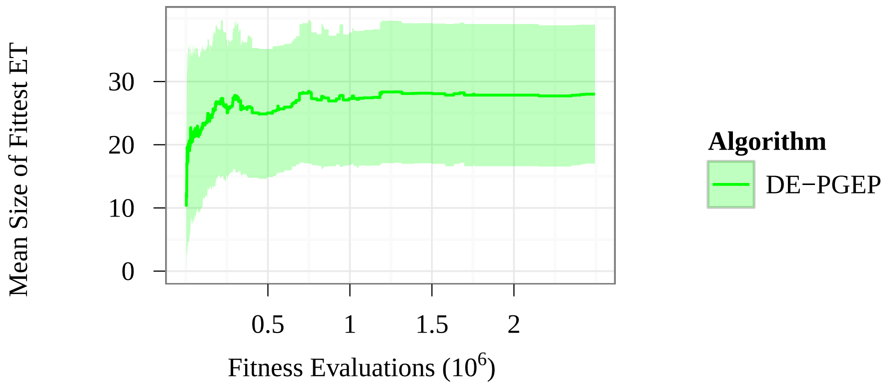
Utilizing otherwise unnecessary operators like exp, ln, and sqrt, DE-PGEP is still able to evolve good numerical constants given the most promising, if not ideal, chromosome. Besides the convoluted appearance of this solution, the rather large values of the constants do however demonstrate that the default initial bounds are not too restrictive, i.e., larger constants are still attainable if needed. This particular solution was obtained with (f_7, α_3, β_3).



(A) Plot of the fitness evaluations against the log scaled mean of the best NRMSEs.



(B) Plot of the fitness evaluations against the mean number of constants in the fittest ETs.



(C) Plot of the fitness evaluations against the mean size of the fittest ETs.

Figure 23. Various plots reporting, from top to bottom, the NRMSE, constant counts, and ET sizes over all 40 independent trials for the narrower sinusoidal function benchmark.

It is possible, given the narrow interval over which training occurred, that DE-PGEP produced a contorted expression because the interval did not permit the target function to exhibit more of its true behavior. But again, the fittest expression discovered by DE-PGEP performed better on the testing set than it did on the training set, implying that solutions do not exhibit overfitting and generalize quite well. As a reminder to the reader, the ln and sqrt functions are protected and thus well-defined over the entire input domain of $[-0.5, 0.5]$.

Unlike G_1, which was used for the quadratic and cubic polynomials, the expanded gene set G_2 in Equation 4.12 contains several non-linear uni-arity functions that can be used to aid the creation of good numerical constants. Both of the original example solutions shown in Equation 4.17 and Equation 4.19 utilize these previously unavailable non-linear functions to transform the evolvable constants. Such functions like the exp and ln can be used to amplify or dampen a constant depending on its sign. The cos and sin instead restrict values to the range of $[-1, 1]$, and can squash constants that have exceedingly large and otherwise useless magnitudes. Thus, it would seem that CC can be accelerated through the use of a functionally diverse gene set. Furthermore, when considering CC in general, it is important to consider the interplay between constants and functions as there are many ways to creatively synthesize good numerical constants.

4.8.4 Wider Sinusoidal Function

When considering a wider window, Equation 4.7 supposedly becomes less predictable. But surprisingly, even the unsimplified expression in Equation 4.19 is strikingly similar in structure to that of the target function. More interestingly though, the simplified version in Equation 4.20

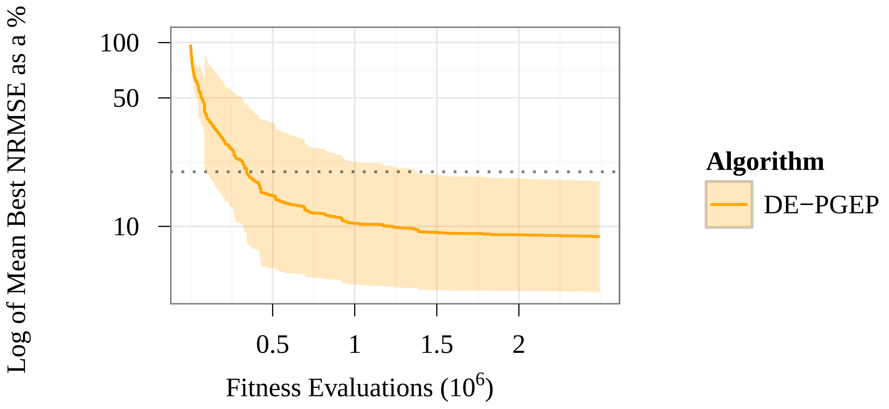
is very different from that of Equation 4.18, which was trained with the exact same control parameters, but on a narrower interval. That is, Equation 4.20 was discovered with (f_7, α_4, β_3) and not (f_7, α_3, β_3). Given that both the RMSE and NRMSE for the mean and best individuals reported in Table VIII for (f_7, α_4, β_3) are lower than those reported for (f_7, α_3, β_3), these results seem to contradict the supposition that this sinusoide is more difficult with a wider window. However, even wider intervals (e.g., $[-2, 2]$ and $[-3, 3]$) have been considered elsewhere.

$$f_7(x) \approx \hat{f}_7'(x) = -0.300998 * x * \sin\Big(x / \cos\big(\ln(-5.645193)\big)\Big) \tag{4.19}$$

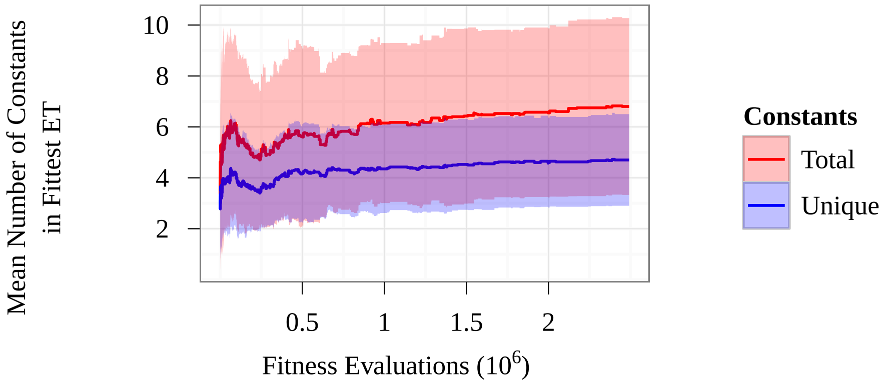
$$= 0.300998 * x * \sin(6.276440 * x) \tag{4.20}$$

4.9 Summary

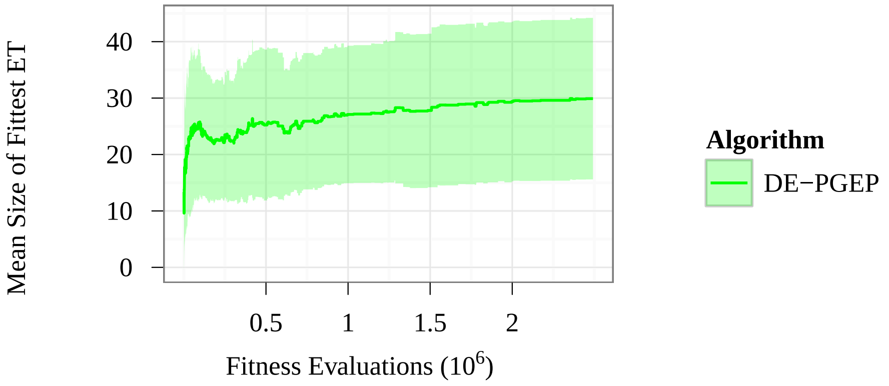
A significantly different approach to Symbolic Regression (SR) with a seamlessly integrated Constant Creation (CC) mechanism has been proposed. Abandoning the discrete representation used by Prefix Gene Expression Programming (PGEP) and adopting a continuous representation, has permitted the use of a robust real-valued optimization algorithm known as Differential Evolution (DE). This conveniently allows for expressions and constants to co-exist in the same vector-based representation and to be simultaneously evolved. Impressive performance was consistently obtained on four experimental benchmarks, the fittest solutions were presented and critiqued, and various aspects of redundant representations and neutral mutations were briefly



(A) Plot of the fitness evaluations against the log scaled mean of the best NRMSEs.



(B) Plot of the fitness evaluations against the mean number of constants in the fittest ETs.



(C) Plot of the fitness evaluations against the mean size of the fittest ETs.

Figure 24. Various plots reporting, from top to bottom, the NRMSE, constant counts, and ET sizes over all 40 independent trials for the wider sinusoidal function benchmark.

introduced and related to the proposed method, which has been called Differentially Evolved Prefix Gene Expression Programming (DE-PGEP).

5. CONCLUSION

5.1 Summary of Contributions

The primary focus of this thesis was on numerical optimization as means to Symbolic Regression (SR) program synthesis. Two major contributions were presented and empirically studied. Both contributions used similar mechanisms, but in different capacities, to synthesize computer programs. These primary mechanisms were Differential Evolution (DE) and Prefix Gene Expression Programming (PGEP), both of which are inspired by natural biological processes and evolutionary concepts like survival of the fittest, random mutations, and the inheritance of beneficial traits through sexual reproduction.

The first major contribution was named Probabilistically Guided Prefix Gene Expression Programming (PG-PGEP) and used an indirect approach to SR program synthesis. That is, instead of directly evolving individual programs, the constrained probabilistic parameters of graphical models, namely Hidden Markov Models (HMMs), were evolved with DE. The HMMs were then used to generate random samples of likely PGEP programs. This non-deterministic sampling scheme of PG-PGEP constitutes a dynamic ontological process that although not novel in design, is in its application.

The second major contribution adopted a complementary approach, which instead used DE to directly evolve PGEP programs. The discrepancy between the discrete encoding of PGEP chromosomes and the continuous vectorized representation of DE was reconciled through the development of a deterministic ontological process. A form of discretization, based on truncation and modular arithmetic, was used to transform the continuous vector representation into a discrete symbolic PGEP chromosome. The continuous representation of DE also conveniently allowed for the development of an seamlessly integrated numerical Constant Creation (CC) technique that is symbiotic in nature. Collectively, these changes warranted the introduction of an entirely new evolutionary process, which has become to be known as the Differentially Evolved Prefix Gene Expression Programming (DE-PGEP) algorithm.

5.2 Future Research

5.2.1 Harmonious Research Directions

Since PG-PGEP and DE-PGEP are very similar in many respects, both algorithms could easily benefit from improvements or enhancements that apply to certain shared components. In particular, this especially applies to the constraint handling techniques of DE. It would be interesting to pursue the development of constrained mutation and crossover operations that retain the spirit of DE's canonical operations. Ideally, such operators would completely avoid repair operations, penalties, and distinctions between feasible and infeasible vectors, while simultaneously respecting the complex interdependent constraints present in the definition of an HMM in PG-PGEP or the much simpler independent upper and lower bounds required by DE-PGEP. Unfortunately, the number of times that the canonical operations resulted in violations of the constraints was not recorded. Thus, it cannot be said with absolute certainty how useful such an improvement might be for either PG-PGEP or DE-PGEP. In any case, the constraints in especially one circumstance are fragile and easily violated, so intuitively, this seems like one promising and fruitful direction for future research.

Furthermore, the length limitations imposed by the vectors of DE are prohibitive in many ways. An effective and efficient variable length implementation of the DE algorithm would be greatly welcomed by both PG-PGEP and DE-PGEP. In PG-PGEP, variable length vectors would allow the HMM topology space to be more fully explored. Of particular interest would be the ability to vary the number of hidden states in the vectorized representation of an HMM, both throughout populations and individual trials. A good starting point for this particular improvement might be that of (O'Neill et al. 2006), where four unique versions of a variable length Particle Swarm Optimization (PSO) algorithm were empirically studied. Variable length vectors would also be beneficial to DE-PGEP since both the length of a chromosome and the number of constants must be fixed at the beginning of a run. While these two settings were mostly constant throughout all experiments conducted in this thesis, adaptive chromosome lengths and constant counts could reduce or eliminate any biases that these settings might have inadvertently introduced.

Similarly, the use of other exogenous control parameters like the DM scaling factor (i.e., F) and the rate of crossover (i.e., Cr) requires both intuition and experience. Manually settings these parameters is tedious, time consuming, and unproductive. Statistical approaches to experimental design like Latin Hypercube Sampling (LHS) (Bartz-Beielstein 2006) are good alternatives, but ideally, the exogenous parameters would be migrated to endogenous control parameters through some embedded self-adaptive process. There is already much research alleviating the supervisor from this task in DE (Teo 2005; Yang et al. 2008; Zhang and Sanderson 2009) and such work would only need to be integrated into either PG-PGEP or DE-PGEP to realize any potential performance improvements. Such ideas might even be further extended to the DM operators as numerous strategies exist and the same strategy might not always be the best given certain initial populations or population configurations encountered during a run.

Finally, since both PG-PGEP and DE-PGEP were only evaluated on synthetic or artificial toy benchmark problems, it would be prudent to evaluate the two algorithms on real-world problems with still unknown or illusive solutions. More challenging problems like the Ocean Color Inverse Problem (OCIP), which has been previously studied in (Valigiani et al. 2004) using canonical and feature enhanced versions of GP, would be an interesting application of the proposed algorithms. Another area of research closely related to SR is Time Series Prediction (TSP). The results from applying GP and GEP to naturally occurring phenomena like the sunspot times series have already been presented in (Ferreira 2002a; Luzia Vidal de Souza and da Rosa 2009; Duan and Povinelli 2001; Lopes and Weinert 2004; Dolin et al. 2002), and it would therefore constitute another candidate for a good real-world benchmark problem. In addition, as a consequence of the nature of TSP, the extrapolation abilities of PG-PGEP and DE-PGEP would be more thoroughly evaluated.

5.2.2 Dichotomous Research Directions

In addition to the shared directions elaborated upon above, PG-PGEP and DE-PGEP both have unique aspects that can should be independently pursued in future research. For PG-PGEP, the main concern is to reduce extraneous sampling and eliminate unnecessary and possibly expensive fitness evaluations. While a straightforward or rudimentary approach could use hashing to detect duplicates, additional overhead would still be incurred during sampling. Thus, any future research on PG-PGEP should really readdress the sampling scheme detailed in Section 3.2.3. A more intelligence sampling scheme, possibly based on the "believability" of sequence, is definitely worth pursing. While the concept of "believability" is still ambiguous, it might be based on the probability of the model emitting a certain sequence as well as the plausibility of the composition of the sequence. Most likely, any such improved sampling scheme would use Dynamic Programming (DP) and might even be based on a modified or extended version of the Viterbi algorithm (Rabiner 1989).

As for DE-PGEP, the two most promising as well as interesting avenues of research are related to the "doubly" redundant encoding and the ability to redefine the search space or fitness landscape of any problem. While redundancy and neutral mutations have been previously studied in (Ferreira 2002c) for GEP, the uniform redundancy as induced by the ontological process described back in Section 4.4.1 needs to be investigated more fully. That is, it needs to be determined whether this redundancy has a statistically significant effect on DE-PGEP, and if so, whether any such effect is either advantageous or disadvantageous. Furthermore, it may also be worthwhile to investigate other ontological processes which result in non-uniform types of redundancy.

Finally, the unique ability to redefine the search search space given a particular problem and a unique gene set may further improve the results obtainable with DE-PGEP. Dynamically altering the fitness landscape has already been studied, but this usually involves dynamically altering the composition of the training set (Gathercole and Ross 1994) or adapting the objective function itself (Uchibe et al. 2002). The proposition being presented here is novel and is instead concerned with the restructuring of the search space to be more conducive to exploration or exploitation through adaptively redefining the mapping between indices and genes. Although large numbers of trials would still be required if both directions were pursued, the use of Common Random Numbers (CRNs) (Bartz-Beielstein 2006) should also be seriously considered in order to induce a positive variance and synchronize all operations for truly comparable results.

CITED LITERATURE

Peter J. Angeline. *Two self-adaptive crossover operators for genetic programming*, pages 89–109. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-01158-1. URL http://portal.acm.org/citation.cfm?id=270195.270204.

Sébastien Aupetit, Nicolas Monmarché, and Mohamed Slimane. Hidden markov models training by a particle swarm optimization algorithm. *Journal of Mathematical Modelling and Algorithms*, 6(2):175–193, May 2006.

Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation*. Natural Computing Series. Springer, 2006.

Janis Barzdins and Guntis Barzdins. Towards Efficient Inductive Synthesis: Rapid Construction of Local Regularities. In Gerhard Brewka, Klaus P. Jantke, and Peter H. Schmitt, editors, *Nonmonotonic and Inductive Logic*, volume 659 of *Lecture Notes in Computer Science*, pages 132–140. Springer, 1993. ISBN 3-540-56433-0.

Jonathan Byrne, Michael O'Neil, Erik Hemberg, and Anthony Brabazon. Analysis of constant creation techniques on the binomial-3 problem with grammatical evolution. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 568–573, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2958-5. URL http://portal.acm.org/citation.cfm?id=1689599.1689673.

Stefano Cagnoni, Daniel Rivero, and Leonardo Vanneschi. A purely evolutionary memetic algorithm as a first step towards symbiotic coevolution. In *Congress on Evolutionary Computation*, pages 1156–1163. IEEE, 2005. ISBN 0-7803-9363-5.

Brian Cerny, Chi Zhou, Weimin Xiao, and Peter Nelson. Probabilistically Guided Prefix Gene Expression Programming. In Natalio Krasnogor, Giuseppe Nicosia, Mario Pavone, and David Pelta, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*, pages 15–26. Springer Berlin / Heidelberg, 2008a. URL http://dx.doi.org/10.1007/978-3-540-78987-1_2.

Brian M. Cerny, Peter C. Nelson, and Chi Zhou. Using Differential Evolution for Symbolic Regression and Numerical Constant Creation. In Maarten Keijzer, editor, *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1195–1202, New York, NY, USA, July 2008b. ACM. ISBN 978-1-60558-130-9. URL http://doi.acm.org/10.1145/1389095.1389331.

David Maxwell Chickering. Learning Bayesian Networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, Lecture Notes in Statistics, pages 121–130. Springer, 1996. ISBN 0387947361.

Joel S. Cohen. *Computer Algebra and Symbolic Computation: Elementary Algorithms*. A K Peters, LTD, Natick, 2002. ISBN 1-56881-158-6.

Dan Costelloe and Conor Ryan. On improving generalisation in genetic programming. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 61–72. Springer Berlin / Heidelberg, 2009. URL http://dx.doi.org/10.1007/978-3-642-01181-8_6.

Ian Dempsey. Constant generation for the financial domain using grammatical evolution. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, GECCO '05, pages 350–353, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1102256.1102334. URL http://doi.acm.org/10.1145/1102256.1102334.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. doi: 10.2307/2984875. URL http://web.mit.edu/6.435/www/Dempster77.pdf.

Brad Dolin, Maribel Garcia Arenas, and Juan J. Merelo Guervós. Opposites Attract: Complementary Phenotype Selection for Crossover in Genetic Programming. In Juan J. Merelo-Guervos, Panagiotis Adamidis, Hans-Georg Beyer, Jose-Luis Fernandez-Villacanas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII*, number 2439 in Lecture Notes in Computer Science, LNCS, pages 142–152, Granada, Spain, September 2002. Springer-Verlag. ISBN 3-540-44139-5. URL http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2439&spage=142.

Minglei Duan and Richard Povinelli. Nonlinear Modeling: Genetic Programming vs. Fast Evolutionary Programming. In Cihan H. Dagli, editor, *Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE 2001)*, pages 171–176, St. Louis, Missouri, USA, November 2001. URL http://povinelli.eece.mu.edu/publications/papers/annie2001a.pdf.

Vitaliy Feoktistov. *Differential Evolution: In Search of Solutions*, volume 5 of *Optimization and Its Applications*. Springer, 2006.

Thomas Fernandez and Matthew P. Evett. Numeric Mutation as an Improvement to Symbolic Regression in Genetic Programming. In *Proceedings of the 7th International Conference on Evolutionary Programming VII*, EP '98, pages 251–260, London, UK, 1998. Springer-Verlag. ISBN 3-540-64891-7. URL http://portal.acm.org/citation.cfm?id=647902.738994.

Cândida Ferreira. Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13(2):87–129, 2001.

Cândida Ferreira. Function Finding and the Creation of Numerical Constants in Gene Expression Programming. In *Proceedings of the 7th Online World Conference on Soft Computing in Industrial Applications (WSC7)*, Granada, Spain, September and October 2002a.

Cândida Ferreira. Mutation, Transposition, and Recombination: An Analysis of the Evolutionary Dynamics. In H. John Caulfield, Shu-Heng Chen, Heng-Da Cheng, Richard J. Duro, Vasant Honavar, Etienne E. Kerre, Mi Lu, Manuel Graña Romay, Timothy K. Shih, Dan Ventura, Paul P. Wang, and Yuanyuan Yang, editors, *JCIS*, pages 614–617. JCIS / Association for Intelligent Machinery, Inc., 2002b. ISBN 0-9707890-1-7.

Cândida Ferreira. Genetic Representation and Genetic Neutrality in Gene Expression Programming. *Advances in Complex Systems*, 5(4):389–408, 2002c. URL http://www.gene-expression-programming.com/webpapers/Ferreira-ACS2002.pdf.

Cândida Ferreira. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Springer-Verlag, second edition, 2006.

Chris Gathercole and Peter Ross. *Dynamic Training Subset Selection for Supervised Learning in Genetic Programming*, volume 866, pages 312–321. Springer-Verlag, 1994. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/94-006.ps.gz.

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

Jennifer Green, Jacqueline L. Whalley, and Colin G. Johnson. Automatic Programming with Ant Colony Optimization. In Mark Withall and Chris Hinde, editors, *Proceedings of the 2004 UK Workshop on Computational Intelligence*, pages 70–77. Loughborough University, September 2004. ISBN 1-874152-11-X. URL http://www.cs.kent.ac.uk/pubs/2004/2000.

Steven Gustafson, Edmund K. Burke, and Natalio Krasnogor. On improving genetic programming for symbolic regression. In David Corne, Zbigniew Michalewicz, Marco Dorigo, Gusz Eiben, David Fogel, Carlos Fonseca, Garrison Greenwood, Tan Kay Chen, Guenther Raidl, Ali Zalzala, Simon Lucas, Ben Paechter, Jennifier Willies, Juan J. Merelo Guervos, Eugene Eberbach, Bob McKay, Alastair Channon, Ashutosh Tiwari, L. Gwenn Volkert, Dan Ashlock, and Marc Schoenauer, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 912–919, Edinburgh, UK, 2-5 September 2005. IEEE Press. ISBN 0-7803-9363-5.

Harri Jäske. Prediction of snspots by GP. In *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, pages 79–87, Vaasa, Finladn, August 1996.

Colin G. Johnson. Artificial Immune System Programming for Symbolic Regression. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 350–358, Essex, April 2003. Springer-Verlag. ISBN 3-540-00971-X. URL http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&spage=350.

Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X. URL http://www.cs.vu.nl/~mkeijzer/publications/eurogp2003.ps.gz.

Maarten Keijzer. Scaled Symbolic Regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004. ISSN 1389-2576. URL http://dx.doi.org/10.1023/B:GENP.0000030195.77571.f9.

James Kennedy and Russell Eberhart. Particle Swarm Optimization. In *Proceedings of the Fourth IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, 1995.

Mark Kotanchek, Guido Smits, and Ekaterina Vladislavleva. Trustable Symoblic Regression Models. In Rick L. Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 12, pages 203 – 222. Springer, Ann Arbor, May 2007.

Mark E. Kotanchek. Real World Data Modeling. In Pelikan and Branke (2010), pages 2863–2896. ISBN 978-1-4503-0072-5.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, December 1992.

W. B. Langdon and B. F. Buxton. Genetic programming for mining DNA chip data from cancer patients. In *Genetic Programming and Evolvable Machines*, pages 251–257. Genetic Programming and Evolvable Machines, September 2004.

William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002. ISBN 3-540-42451-2. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/.

Pat Langley, Gary L. Bradshaw, and Herbert A. Simon. Rediscovering Chemistry with the BACON System. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, chapter 10, pages 307–330. Morgan Kaufmann, San Mateo, CA, 1983.

Pedro Larrañaga. *A Review of Estimation of Distribution Algorithms*, chapter 3, pages 57–100. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, Boston, MA, 2002.

Xin Li. *Self-Emergence of Structures in Gene Expression Programming.* PhD thesis, University of Illinois at Chicago, 2006.

Xin Li, Chi Zhou, Peter C. Nelson, and Thomas M. Tirpak. Investigation of Constant Creation Techniques in the Context of Gene Expression Programming. In Maarten Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, July 2004. URL http://www.cs.uic.edu/~xli1/papers/GEPConstantCreation(GECCO04_LBP).pdf.

Xin Li, Chi Zhou, Weimin Xiao, and Peter C. Nelson. Prefix Gene Expression Programming. In Franz Rothlauf, editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, Washington, D.C., USA, 25-29 June 2005. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005lbp/papers/85-li.pdf.

Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data.* Data Centric Systems and Applications. Springer, 2007.

Moshe Looks. Learning Computer Programs with the Bayesian Optimization Algorithm. Master's thesis, Washington University in St. Louis, St. Louis, Missouri, USA, May 2005. URL http://doi.acm.org/10.1145/1068009.1068134.

Heitor S. Lopes and Wagner R. Weinert. EGIPSYS: an Enhanced Gene Expression Programming Approach for Symbolic Regression Problems. *International Journal of Applied Mathematics and Computer Science*, 14(3):375–384, 2004. URL http://matwbn.icm.edu.pl/ksiazki/amc/amc14/amc1434.pdf. Special Issue: Evolutionary Computation.

Sean Luke and Lee Spector. A Revised Comparison of Crossover and Mutation in Genetic Programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 208–213, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann. ISBN 1-55860-548-7. URL http://www.cs.gmu.edu/~sean/papers/revisedgp98.pdf.

Anselmo C. Neto Luzia Vidal de Souza, Aurora T. R. Pozo and Joel M. C. da Rosa. Genetic Programming and Boosting Technique to Improve Time Series Forecasting. In Wellington Pinheiro dos Santos, editor, *Evolutionary Computation*, chapter 6, pages 103–120. InTech, Vukovar, Croatia, October 2009. URL http://www.intechopen.com/articles/show/title/genetic-programming-and-boosting-technique-to-improve-time-series-forecasting. 978-953-307-008-7.

Alberto Moraglio and Sara Silva. Geometric Differential Evolution on the Space of Genetic Programs. In Anna Isabel Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Genetic Programming*, volume 6021 of *Lecture Notes in Computer Science*, pages 171–183. Springer Berlin / Heidelberg, 2010. URL http://dx.doi.org/10.1007/978-3-642-12148-7_15.

Richard E. Neapolitan. *Learning Bayesian Networks*. Artificial Intelligence. Prentice Hall, 2004.

Supakit Nootyaskool and Boontee Kruatrachue. Hybrid Genetic Algorithm with Baum-Welch Algorithm by Using Diversity Population Technique. In *International Symposium on Communications and Information Technologies, 2006*, pages 15–20. IEEE Press, 2006. ISBN 0-7803-9741-X.

Michael O'Neill and Anthony Brabazon. Grammatical Differential Evolution. In Hamid R. Arabnia, editor, *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, volume 1, pages 231–236, Las Vegas, Nevada, USA, June 2006. CSREA Press. ISBN 1-932415-96-3. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.3012.

Michael O'Neill, Ian Dempsey, Anthony Brabazon, and Conor Ryan. Analysis of a digit concatenation approach to constant creation. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 173–182, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X. URL http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&spage=173.

Michael O'Neill, Finbar Leahy, and Anthony Brabazon. Grammatical Swarm: A Variable-Length Particle Swarm Algorithm. In Nadia Nedjah and Luiza Mourelle, editors, *Swarm*

Intelligent Systems, volume 26 of *Studies in Computational Intelligence*, pages 59–74. Springer Berlin / Heidelberg, 2006. URL http://dx.doi.org/10.1007/978-3-540-33869-7_3.

Martin Pelikan and Jürgen Branke, editors. *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*, 2010. ACM. ISBN 978-1-4503-0072-5.

Martin Pelikan, David E. Goldberg, and Fernando G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21: 5–20, January 2002. ISSN 0926-6003. URL http://portal.acm.org/citation.cfm?id=585660.585661.

Cassio Pennachin, Moshe Looks, and João A. de Vasconcelos. Robust Symbolic Regression with Affine Arithmetic. In Pelikan and Branke (2010), pages 917–924. ISBN 978-1-4503-0072-5. URL http://research.google.com/pubs/archive/36294.pdf.

Riccardo Poli, William B. Langdon, Nicholas F. McPhee, and John R. Koza. *A Field Guide to Genetic Programming.* Lulu Publishing, 2008. URL http://www.gp-field-guide.org.uk.

Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization.* Natural Computing Series. Springer, December 2005. ISBN 9783540209508.

Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Gunther R. Raidl. A Hybrid GP Approach for Numerically Robust Symbolic Regression. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 323–328, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. ISBN 1-55860-548-7. URL http://www.ads.tuwien.ac.at/publications/bib/pdf/raidl-98c.pdf.

Thomas Kiel Rasmussen and Thiemo Krink. Improved Hidden Markov Model Training for Multiple Sequence Alignment by a Particle Swarm Optimization–Evolutionary Algorithm Hybrid. volume 72 of *Computational Intelligence in Bioinformatics*, pages 5–17, November 2003.

Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Springer-Verlag Berlin Heidelberg, 2nd edition, 2006. ISBN 3-540-25059-X.

Conor Ryan and Maarten Keijzer. An analysis of diversity of constants of genetic programming. In *Proceedings of the 6th European conference on Genetic programming*, EuroGP'03, pages 404–413, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-00971-X. URL http://portal.acm.org/citation.cfm?id=1762668.1762708.

Conor Ryan, J.J. Collins, Jj Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83–95. Springer-Verlag, 1998.

Ângela A. R. Sá, Adriano O. Andrade, Alcimar B. Soares, and Slawomir J. Nasuto. Estimation of Hidden Markov Models Parameters using Differential Evolution. In Frank Guerin and Wamberto Weber Vasconcelos, editors, *Proceedings of the AISB 2008 Symposium on Swarm Intelligence Algorithms and Applications*, volume 11, pages 51–56, Aberdeen, Scotland, April 2008. AISB.

Luciano Sánchez. Interval-valued GA-P algorithms. *IEEE Trans. Evolutionary Computation*, 4(1):64–72, 2000.

Yin Shan, Robert I. McKay, Daryl Essam, and Hussein A. Abbass. *A Survey of Probabilistic Model Building Genetic Programming*, volume 33 of *Studies in Computation Intelligence*, pages 121–160. Springer-Verlag, 2006. ISBN 3-540-34953-7.

R. Shipman, M. Shackleton, and I. Harvey. The Use of Neutral Genotype-Phenotype Mappings for Improved Evolutionary Search. *BT Technology Journal*, 18:103–111, October 2000. ISSN 1358-3948. doi: 10.1023/A:1026714927227. URL http://portal.acm.org/citation.cfm?id=592202.592464.

Lee Spector. Autoconstructive Evolution: Push, PushGP, and Pushpop. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 137–146, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. ISBN 1-55860-774-9. URL http://hampshire.edu/lspector/pubs/ace.pdf.

Matej Sprogar. A Study of GP's Division Operators for Symbolic Regression. In *Seventh International Conference on Machine Learning and Applications, ICMLA '08*, pages 286–291, La Jolla, San Diego, USA, December 2008. IEEE. doi: doi:10.1109/ICMLA.2008.84.

Jason Teo. Differential Evolution with Self-adaptive Populations. In Rajiv Khosla, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3681 of *Lecture Notes in Computer Science*, pages 155–156. Springer Berlin / Heidelberg, 2005. URL http://dx.doi.org/10.1007/11552413_183.

Alexander Topchy and William Punch. Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 155–162. Morgan Kaufmann, July 2001. ISBN 1558607749.

Eiji Uchibe, Masakazu Yanase, and Minoru Asada. Behavior generation for a mobile robot based on the adaptive fitness function. *Robotics and Autonomous Systems*, 40(2-3):69–77, 2002.

Daniel Upper. *Theory and Algorithms for Hidden Markov Models and Generalized Hidden Markov Models*. PhD thesis, University of California at Berkeley, 1997.

Fatemeh Vafaee, Peter C. Nelson, Chi Zhou, and Weimin Xiao. Dynamic Adaptation of Genetic Operators' Probabilities. In Natalio Krasnogor, Giuseppe Nicosia, Mario Pavone, and

David A. Pelta, editors, *NICSO*, volume 129 of *Studies in Computational Intelligence*, pages 159–168. Springer, 2007. ISBN 978-3-540-78986-4.

Grégory Valigiani, Cyril Fonlupt, and Pierre Collet. Analysis of GP Improvement Techniques over the Real-World Inverse Problem of Ocean Color. In *EuroGP*, pages 174–186, 2004. ISBN 3-540-21346-5.

L. Gwenn Volkert. Investigating EA Based Training of HMM Using a Sequential Parameter Optimization Approach. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 2742–2749. IEEE Press, July 2006. ISBN 0-7803-9487-9.

Kyoung-Jae Won, Thomas Hamelryck, Adam Prügel-Bennett, and Anders Krogh. Evolving hidden Markov models for protein secondary structure prediction. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, 1(2-5):33–40, September 2005.

Zhuli Xie, Xin Li, Barbara Di Eugenio, Weimin Xiao, Thomas M. Tirpak, and Peter C. Nelson. Using Gene Expression Programming To Construct Sentence Ranking Functions For Text Summarization. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 1381–1384, Geneva, Switzerland, August 2004.

Liping Xue, Junxun Yin, Zhen Ji, and Lai Jiang. A Particle Swarm Optimization for Hidden Markov Model Training. In *The 8th International Conference on Signal Processing*, volume 1. IEEE Press, 2006. ISBN 0-7803-9737-1.

Kohsuke Yanai and Hitoshi Iba. Estimation of Distribution Programming: EDA-based Approach to Program Generation. In Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors, *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 103–122. Springer Berlin / Heidelberg, 2006. URL http://dx.doi.org/10.1007/3-540-32494-1_5.

Z. Yang, K. Tang, and X. Yao. Self-Adaptive Differential Evolution with Neighborhood Search. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence)*, pages 1110–1116, 2008.

Daniela Zaharie. Critical values for the control parameters of Differential Evolution algorithm. In R. Matouek and P. Omera, editors, *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, pages 62–67, Brno, Czech Republic, June 2002.

Daniela Zaharie. A Comparative Analysis of Crossover Variants in Differential Evolution. In *Proceedings of the Interational Multiconference on Conputer Science and Information Technology*, volume 2, pages 171–181, Wisla, Poland, October 2007. URL http://www.proceedings2007.imcsit.org/pliks/115.pdf.

Ivan Zelinka, Zuzana Oplakova, and Lars Nolle. Boolean symmetry function synthesis by means of arbitrary evolutionary algorithms - comparative study. In *Proceedings of the 18th European Simulation Multiconference - ESM 2004*, pages 143–148, Magdeburg, Germany, June 2004. Graham Horton SCS Europe.

Ivan Zelinka, Zuzana Oplatkova, and Lars Nolle. Analytic Programming – Symbolic Regression by Means of Arbitrary Evolutionary Algorithms. *International Journal of Simulation: Systems, Science and Technology*, 6(9):44–56, August 2005. Special Issue.

Ivan Zelinka, Zuzana Oplatkova, and Lars Nolle. Evolutioarny Identification of Dynamical Systems. *International Journal of Simulation: Systems, Science & Technology*, 9(3):47–59, September 2008. Special Issue on: Artificial Intelligence.

J. Zhang and A.C. Sanderson. *Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization*, volume 1 of *Adaptation, Learning, and Optimization*. Springer, 2009. ISBN 9783642015267.

Qiongyun Zhang, Chi Zhou, Weimin Xiao, and Peter C. Nelson. Improving Gene Expression Programming Performance by Using Differential Evolution. In *Proceedings of the Sixth International Conference on Machine Learning and Applications*, ICMLA '07, pages 31–37, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3069-9. URL http://dx.doi.org/10.1109/ICMLA.2007.55.

Chi Zhou, Weimin Xiao, Thomas M. Tirpak, and Peter C. Nelson. Evolving accurate and compact classification rules with gene expression programming. *IEEE Transactions on Evolutionary Computation*, 7(6):519–531, December 2003.

VITA

NAME:	Brian Matthew Cerny
EDUCATION:	M.S., Computer Science, University of Illinois at Chicago, Chicago, Illinois, USA, August 2011
	B.S., Computer Science and Mathematics, Magna Cum Laude, Elmhurst College, Elmhurst, Illinois, USA, May 2006
PROFESSIONAL EXPERIENCE:	Software Engineer, GeeYee, Chicago, Illinois, USA, 2008 - 2011
	Assistant Network Administrator, Timothy Christian Schools, Elmhurst, Illinois, USA, 1999 - 2006
HONORS:	Research Assistantship, Department of Computer Science, University of Illinois at Chicago, Chicago, USA, 2006 - 2008
PROFESSIONAL MEMBERSHIP:	Student member of ACM (Association for Computing Machinery) Student member of IEEE (Institute of Electrical and Electronics Engineers)
PUBLICATIONS:	Brian Cerny, Chi Zhou, Weimin Xiao, and Peter Nelson. Probabilistically Guided Prefix Gene Expression Programming. In Natalio Krasnogor, Giuseppe Nicosia, Mario Pavone, and David Pelta, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*, pages 15–26. Springer Berlin / Heidelberg, 2008a. URL http://dx.doi.org/10.1007/978-3-540-78987-1_2. Selected as one of the best papers presented at NICSO 2007.
	Brian M. Cerny, Peter C. Nelson, and Chi Zhou. Using Differential Evolution for Symbolic Regression and Numerical Constant Creation. In Maarten Keijzer, editor, *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1195–1202, New York, NY, USA, July 2008b. ACM. ISBN 978-1-60558-130-9. URL http://doi.acm.org/10.1145/1389095.1389331.