Using Differential Evolution for Symbolic Regression and Numerical Constant Creation

Brian M. Cerny
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607 USA
bcerny@cs.uic.edu

Peter C. Nelson
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607 USA
nelson@uic.edu

Chi Zhou
Physical & Digital Realization Research Center
Motorola Inc.
Schaumburg, IL 60196 USA
chi.zhou@motorola.com

ABSTRACT

One problem that has plagued *Genetic Programming* (GP) and its derivatives is numerical constant creation. Given a mathematical formula expressed as a tree structure, the leaf nodes are either variables or constants. Such constants are usually unknown in *Symbolic Regression* (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 to 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 operators. In this paper, an entirely new approach is proposed for constant creation. Through the adoption of a robust, real-valued optimization algorithm known as *Differential Evolution* (DE), constants and GP-like 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 constant creation.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*

General Terms

Algorithms

Keywords

Differential Evolution, Constant Creation, Genetic Programming, Genetic Algorithms, Prefix Gene Expression Programming, Symbolic Regression, Neutral Mutations, Redundant Representation, Optimization, Combinatorial Search

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

Given a set of numerical inputs and outputs, one frequently wants to find a mathematical formula which adequately models the provided data. The term given to this task is *Symbolic Regression* (SR) and a widely used method to discover such expressions is the paradigm of *Evolutionary Algorithms* (EAs). A tree structure is generally utilized to evaluate the performance or quality of an expression and the nodes of the tree represent functions, variables, and constants. As SR has become increasingly popular, the need to model more complex functions has naturally occurred and in addition to synthesizing an expression, numerical constants must be approximated as well. A novel method which was developed just for SR with numerical constants is introduced and evaluated in this paper using *Prefix Gene Expression Programming* (PGEP) and *Differential Evolution* (DE). Although the PGEP notation is used to represent the mathematical formulas, the method is generally applicable to other *Genetic Programming* (GP) paradigms. That is, as long as a well-defined method exists to convert a linear encoding to a valid tree structure, this new method is easily adapted.

Originally proposed by Xin Li [1], *Prefix Gene Expression Programming* (PGEP) is an adaptation of the *Gene Expression Programming* (GEP) algorithm invented by Cândida Ferreira [2]. GEP is also an extension of two more traditional *Evolutionary Algorithms* (EAs) which are *Genetic Programming* (GP) [3] and *Genetic Algorithms* (GAs) [4]. Unlike its ancestors, GEP and thus PGEP as well, do not suffer from a constrained search space. That is, GAs and GP do not have separate *genotypes* (genetic makeup) and *phenotypes* (body and behavior). In the case of GAs, binary or real-valued linear representations of a fixed-length are standard where GP uses tree structures composed of discrete symbols. GP is then more suited to handle dynamic or not so well-defined problems where GAs lend themselves to problems of a more static nature, e.g., parameter optimization. GEP combines the desirable aspects of both GAs and GP by adopting a fixed-length, linear genotype and a tree based phenotype. This has the effect of separating the genotype search space from the phenotype solution space. In addition, this permits GEP to utilize the more efficient linear operators of GAs instead of the tree based operations required by GP.

Unfortunately, one consequence of separating the genotype from the phenotype requires that the hierarchy and

proximity of genes be maintained, otherwise crossover operations will be more harmful than helpful. The major difference between PGEP and GEP, is that PGEP exploits an encoding which is more resistant to the destructive forces of linear crossover operations. The preservation of good structural encodings yields better performance as convergence is increased and good building blocks are more likely to be exchanged as a whole. Yet in light of these advantages, the discrete encoding of 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 is unable to synthesize mathematical expressions which require the approximation of numerical constants to a high precision.

In this paper, a new type of genotype for PGEP is proposed. This new genotype maintains the linear and fixed-length aspects of the original PGEP, and replaces the original symbolic representation with a real-valued one. But with this new encoding comes the need for an alternative approach to drive the combinatorial search of PGEP. A robust and powerful real-valued optimizer called *Differential Evolution* (DE) which was also inspired by GAs is recruited. PGEP expressions are then evolved by interpreting real-valued parameters as discrete integer values which are in turn converted to PGEP program components. Numerical constants exist in the same representation, are stored after the encoded PGEP expression, extracted when needed, and then incorporated into the expression.

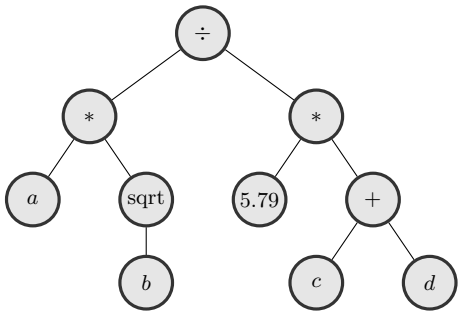
The remainder of this paper is organized as follows: Section 2 introduces all related work which includes PGEP, DE, and existing constant creation techniques. Section 3 then presents the combined *Differential Evolution-Prefix Gene Expression Programming* (DE-PGEP) algorithm in detail. The set of benchmark problems used to evaluate DE-PGEP is described in Section 4.1. The experimental results are presented in Section 4.2, an analysis of DE-PGEP appears in Section 4.3, and example solutions are presented and critiqued in Section 4.4. Section 5 summarizes the research and gives a brief overview of the direction of future work.

2. RELATED WORK

2.1 Prefix Gene Expression Programming

Recently devised, *Prefix Gene Expression Programming* (PGEP) is an *Evolutionary Algorithm* (EA) which although extremely simple in structure and function, provides for an efficient yet powerful approach to the evolution of computer programs. Applied to areas such as symbolic regression [1], text summarization [5], and classification rule mining [6], PGEP has consistently outperformed both traditional machine learning techniques and other existing EAs. Borrowing the fixed-length linear encoding scheme from *Genetic Algorithms* (GAs) and adopting the ramified non-linear tree structures of *Genetic Programming* (GP), PGEP has successfully separated the *genotype* from the *phenotype* through a static *ontological* process. This precise translation from the linear genotype (chromosome) to a hierarchical realization of the phenotype (expression tree), 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 [7].

An example of a linear PGEP chromosome with a fixed-length of sixteen can be seen in Figure 1(b) and the encoded mathematical expression is shown in Figure 1(c). A chromosome is composed of uniquely indexed elements called genes, each of which belongs to the *gene set* G. In the case of Figure 1 where Q denotes the square root and $\div$ represents division, $G = \{+, -, *, \div, \text{sqrt}, a, b, c, d, 0.14, 3.61, 5.79\}$. Typically G can be divided into two disjoint subsets, which in the case of Figure 1 are the *terminal set* $T = \{a, b, c, d, 0.14, 3.61, 5.79\}$ and the *function set* $F = \{+, -, *, \div, \text{sqrt}\}$. As just demonstrated, T usually consists of the input variables (attributes) and selected constants, where F contains all the functions (operators) with an arity greater than or equal to one.



(a) Expression tree

$\div * a \text{ sqrt } b * 5.79 + c \; d - 0.14 \div 3.61 \; b \; d$

(b) Linear chromosome

$$\frac{a\sqrt{b}}{5.79\,(c+d)}$$

(c) Encoded expression

Figure 1: An example of linear PGEP chromosome, the corresponding expression tree, and the encoded mathematical expression

By adopting a prefix notation encoding scheme, PGEP allows for the seamless and unambiguous translation from the chromosome to the expression tree [1]. The chromosome in Figure 1(b) may then be converted to the expression tree in Figure 1(a) by iterating over genes of the chromosome and filling out the expression tree in a depth-first fashion. During this translation process, the expression tree continually grows by branching out according to the arity of the encountered genes. Thus when any terminal gene is encountered, the appropriate branch of the expression tree naturally terminates, and trees of various sizes, shapes, and complexities are produced. It can also be seen in Figure 1 that the encoded expression naturally terminates within the provided bounds. This is the preferred behavior which only results in "junk" or superfluous genes that are ultimately harmless and thus ignored. On the other hand, a branch of the expression tree may also terminate prematurely. Situations such as this can be easily avoided by first subjecting a chromosome to an arity check. Any illegal chromosomes may then be either repaired, discarded, or penalized. To obtain a more thorough understanding of the original PGEP algorithm, the interested reader may wish to consult [1].

2.2 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 [8].

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 [8]. That is, the magnitude of movements throughout the search space are free to increase or decrease automatically. 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 necessary. Adopting the original notation and terminology from [8], 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 1 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 $\mathbf{x}_{i,g}$.

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

$$\mathbf{x}_{i,g} = (x_{j,i,g}), j = 0, \ldots, D-1 \quad (2)$$

When $g = 0$, the initial population $\mathbf{P}_{\mathbf{x},0}$ is randomly initialized according to Equation 3 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 $[0,1)$.

$$x_{j,i,0} = \text{rand}_j(0,1)(b_{j,U} - b_{j,L}) + b_{j,L} \quad (3)$$

For $g > 0$, an *intermediate population* $\mathbf{P}_{\mathbf{v},g}$ is generated by the *differential mutation* operator defined in Equation 4.

$$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}) \quad (4)$$

In Equation 4, $\mathbf{x}_{r0,g}$ is known as the *base vector* where $\mathbf{x}_{r1,g}$ and $\mathbf{x}_{r2,g}$ are called the *difference vectors*. The indices of these vectors are randomly selected such that $i \neq r0 \neq r1 \neq r2$ as this prevents degenerate vector combinations, i.e., $\mathbf{x}_{i,g} \neq \mathbf{v}_{i,g}$. Commonly referred to as the *scaling factor*, $F \in (0, 1+)$ "amplifies" or "dampens" the perturbation by dynamically adjusting the step-size. After mutation, a *trial population* $\mathbf{P}_{\mathbf{u},g}$ is produced using the *uniform crossover* operator defined in Equation 5.

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \text{rand}_j(0,1) \leq Cr \text{ or } j = j_{\text{rand}}, \\ x_{j,i,g} & \text{otherwise.} \end{cases} \quad (5)$$

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 $\mathbf{v}_{i,g}$ [8]. Similar in purpose to the mutually exclusive indices in differential mutation, j_{rand} forces innovation by guaranteeing that $\mathbf{u}_{i,g} \neq \mathbf{x}_{i,g}$. Finally, DE utilizes the local, greedy, and reproductionless selection strategy described in Equation 6.

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f(\mathbf{u}_{i,g}) \leq f(\mathbf{x}_{i,g}), \\ \mathbf{x}_{i,g} & \text{otherwise.} \end{cases} \quad (6)$$

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 optimal. An optimization task is then considered a *success* when $\mathbf{x}^*$ is found or a *failure* if g_{max} is achieved.

2.3 Constant Creation

Probably one of the most widely used applications of GP and its extensions like PGEP has been to that of *Symbolic Regression* (SR). Given a *training set* consisting of several input(s) 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 [3] 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 that 5.79 was needed, PGEP would also have to evolve an approximate value of the constant.

The first attempt to deal with the problem of constant creation was appropriately presented in [3]. Named the *ephermal random constant* and denoted by $\Re$, this new terminal was used to introduce a 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, and instead, multiple constants are combined. Thus, as the evolutionary process progresses, better constants are expected to be emerge while the less useful constants will likely vanish. Other more sophisticated proposals have followed and most of these methods can be organized into three general categories.

2.3.1 Embedded Optimizers

The first category employs a second optimization algorithm which 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 this work is [9]. There, DE was shown to significantly improve the performance of *Gene Expression Programming* (GEP) on two SR problems with real-valued constants. An additional gene called the *Random Number Generator* (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 separate 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 chromosome. 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. Although the basic idea is the same, it will be seen later on that the method proposed in this paper drastically differs in the details.

A very similar technique is described in [10], but instead of DE, a bit-based GA was used. Unlike the previous work, only SR problems with integer coefficients were considered. Other embeddable alternatives include: a gradient based search [11], a moving least squares algorithm [12], and a simulated annealing inspired algorithm [13]. To some de-

gree, these types of searches seem restrictive as the optimization of constants occurs within the context of a single chromosome. Such techniques may produce deceptive solutions, i.e., 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 a *testing set*. 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, [9] and [10] explored various levels of interleaving and [9] also presented the best and worst solutions.

2.3.2 Special Operators

The collection of research belonging to the second category primarily focuses on numerically oriented operators. In [7], each chromosome in a GEP population was given its own fixed-length array of real-valued constants and the constants are inserted into an expression tree via indices, i.e., a single gene c_i in T 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. Assuming that small improvements in the fitness are caused by minor changes in the constants, [14] introduced two different types of mutation, *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. Similar to the work in [14], Li et al [15] proposed greedy, elitist, and temporal extensions to one of or both mutation operators.

2.3.3 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 [16]. 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 constants. It also has the advantage of being extremely easy to integrate into the GE algorithm. Preferring to avoid the noise introduced by non-optimal numerical constants, Keijzer [17] took a considerably different approach and focused on estimating 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 training set. Unfortunately, this last method does not appear to be universally applicable as a linear relationship is assumed.

3. DE-PGEP

The original motivation behind the development of the PGEP algorithm was to combat the disruptive nature of crossover [1], i.e., crossover has the tendency to destroy good building blocks. PGEP mitigated the affects of this phenomenon by a adopting 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 which exchange contiguous segments of genetic material between chromosomes [7].

In this paper, the continuous linear crossover operations in PGEP are replaced with a purely mutation based approach. The mechanism by which programs will be automatically synthesized is DE and this will also conveniently allow for the creation of numerical constants. This proposed method is not exotic, does not introduce any specialized operators, and does not incur the overhead of an embedded optimizer. While the user is still required to specify an arbitrary number of constants, a simple suggestion is given to estimate this quantity.

Many researchers have investigated the feasibility of DE for problems that are discrete or combinatorial in nature [8], but there is only one known piece of work that explores DE as a means to synthesize symbolically encoded computer programs. Mainly, [18] recruited DE to drive a *Grammatical Evolution* (GE) algorithm which was accordingly named *Grammatical Differential Evolution* (GDE). Here, a similar approach is adopted, but various aspects differ. Moreover, GDE failed to utilize the inherent numerical representation of DE for the purposes of constant creation.

Any continuous DE vector $\mathbf{x}_{i,g}$ can be easily and consistently mapped to a discrete vector $\mathbf{w}_{i,g}$. This is accomplished by the simple formula appearing in Equation 7.

$$w_{j,i,g} = \lfloor |x_{j,i,g}| \rfloor \tag{7}$$

The search is constrained by a repair operation and thus it can be safely assumed that Equation 7 always yields a valid index. A trial vector $\mathbf{u}_{i,g}$ is repaired by the formula in Equation 8 where $|G|$ denotes the number of genes in G, $\lfloor u_{j,i,g} \rfloor$ is the floor of $u_{j,i,g}$, and % is the common residue.

$$u_{j,i,g} = (\lfloor |u_{j,i,g}| \rfloor \,\%|G|) + (|u_{j,i,g}| - \lfloor |u_{j,i,g}| \rfloor) \tag{8}$$

Since the user-defined gene set G remains fixed throughout the entirety of a PGEP run, a unique integer index from $[0, |G| - 1]$ may be assigned to every gene $g \in G$ at the onset of a DE-PGEP run. Each parameter $w_{j,i,g}$ of $\mathbf{w}_{i,g}$ may then be interpreted as the gene with the appropriate index. This exactly defines a PGEP chromosome which can be converted to an expression tree using the aforementioned method. Therefore, if the user-defined length of all PGEP chromosomes is l, then D will be equal to l. As a result, a many-to-one mapping is established between a DE vector and a linear PGEP chromosome.

One subtle, but interesting consequence of this mapping is that any value in the half-closed interval $[\lfloor |x_{j,i,g}| \rfloor , \lceil |x_{j,i,g}| \rceil)$ is equivalent with respect to the final outcome, i.e., the same index is realized and thus the same gene is selected. Representations like this have been termed *redundant*. With redundancy, a new and fundamentally different type of mutation is now possible. Prevalent in nature, *neutral mutations* produce genotypical differences without affecting phenotypical behavior [19]. Or in other words, the genetic material is manipulated without adversely affecting the quality of an individual. Extending this idea to artificial search spaces, formations termed *neutral ridges* appear and "drifting" along these neutral ridges allows access to previously unreachable areas of the search space [20]. Conceivably preventing stagnation and increasing performance [19, 20].

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 $\mathbf{x}_{i,g}$, and thus D will now be equal to $l + m$. Furthermore, when a chromosome $\mathbf{a}_{i,g}$ is expanded into an expression tree $t_{i,g}$, the value of each constant is retrieved from $\mathbf{x}_{i,g}$. Specifically, for $k = 0, \ldots, m-1$, the value of all c_{k+1} in $t_{i,g}$ is $c_{k,i,g} = x_{l+k,g}$. That is, the terminals $c_1, \ldots, c_m$ are simply placeholders and the actual constants are substituted into the expression tree at the time of evaluation. This not only ensures that a local pool of reusable constants is available to each chromosome, but it also contributes to the specialization of constants within the context of a single chromosome. By virtue of Equations 4 and 5, 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 expression trees, the interplay between incompatible constants is essential for variation. With this in mind, we can see that the positions and quantities of all genes are determined in a similar manner, and therefore the constant creation process has both local and global qualities.

For future reference, we will denote the two sub-vectors of $\mathbf{x}_{i,g}$ as $\mathbf{e}_{i,g}$ and $\mathbf{c}_{i,g}$ as these respectively represent the expression vector and the associated array of constants. The relationship between $\mathbf{x}_{i,g}$ and its two sub-vectors $\mathbf{e}_{i,g}$ and $\mathbf{c}_{i,g}$ is now more clearly defined in Equation 9.

$$\begin{aligned} \mathbf{e}_{i,g} &= (x_{j,i,g}), j = 0, \ldots, l-1 \\ \mathbf{c}_{i,g} &= (x_{j,i,g}), j = l, \ldots, l+m-1 \end{aligned} \tag{9}$$

Recalling Equation 3, the task of vector initialization must also be addressed. For $\mathbf{e}_{i,g}$ the bounds are simply defined by the user-defined gene set G, but for $\mathbf{c}_{i,g}$ a good set of upper and lower bounds is problem dependent and most likely unknown in advance. Therefore without any additional information, the recommendation in [8] is followed and the complete initialization bounds are defined in Equation 10.

$$\begin{aligned} e_{j,i,g} &= \text{rand}_j(0,1)|G| \\ c_{j,i,g} &= \text{rand}_j(0,1) \end{aligned} \tag{10}$$

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 [8].

4. EXPERIMENTS

4.1 Benchmarks

In order to evaluate the performance of DE-PGEP, a collection of SR problems which have become the standard benchmarks for constant creation have been assembled. The aim was to construct a diverse set of benchmarks which included functions varying in complexity, shape, and type. In addition, SR problems with different constants, both small and large were sought. The selected problems appear in Equations 11–13 and the citations in which a problem appears are displayed immediately to the left of the functions.

$$[3, 7] : f_1(x) = 2.718x^2 + 3.141636x \tag{11}$$

$$[9, 13, 15] : f_2(x) = x^3 - 0.3x^2 - 0.4x - 0.6 \tag{12}$$

$$[14, ?] : f_3(x) = 0.3x \sin(2\pi x) \tag{13}$$

Of all the benchmarks, the 2nd degree polynomial in Equation 11 is the easiest and its rational coefficients, from left to right, are similar to Euler's number and π. Although not exceedingly difficult, the 3rd degree polynomial in Equation 12 can be considered slightly more difficult than Equation 11 as more constants need to be approximated. Since Equation 13 becomes increasingly difficult as the interval under consideration is widened, a couple of different instances of this same problem are investigated too.

To make it possible to compare DE-PGEP to other existing constant creation techniques, we have adopted the common root mean squared error (RMSE) fitness function which is defined in Equation 14.

$$f(\mathbf{x}_{i,g}) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(p_j - t_j)^2} \tag{14}$$

The terms in Equation 14 are as follows: $f(\mathbf{x}_{i,g})$ is the fitness of $\mathbf{x}_{i,g}$, p is the program (chromosome) which $\mathbf{x}_{i,g}$ encodes, n is the size of the training set, p_j is the actual output of p on the j^{th} training case, and t_j is the target output for the j^{th} training case. Compatible with the selection strategy in Equation 6, the RMSE defines an objective function for a minimization task. In this case, the task is to evolve a solution which minimizes the error on a training set. Displayed in Table 1 are brief summaries of the training and testing sets. For future reference, *ID* introduces an unique identifier for each training/testing set pair which is denoted by α. *Interval*, *Size*, and *Sample* columns respectively represent the intervals on which the samplings occurred, the size of the sampled sets, and how the sets were sampled. In the case of the sampling method, *Random* denotes a uniform random sampling from the specified interval and a number denotes the distance between regularly spaced points on the interval.

ID	Interval	Training Set		Testing Set	
		Size	Sampling	Size	Sampling
α_1	$[-10, 10]$	10	Random	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	0.1	2000	Random

Table 1: A brief summary of the training and testing sets used for various benchmark problems.

Although there is no complete agreement on the ideal gene set for benchmarking these problems, we will generally adopt the dominant operators appearing in the cited works. The generic gene sets that appear in Equations 15 and 16 will be those utilized throughout the remainder of this paper.

$$G_1 = \{+, -, *, \div, x, c_1, \ldots, c_m\} \tag{15}$$

$$G_2 = \{+, -, *, \div, \sin, \cos, \exp, \ln, \text{sqrt}, x, c_1, \ldots, c_m\} \tag{16}$$

Besides the common mathematical operators, $+$, $-$, $*$, and $\div$[1], there are several new operators which may require some clarification. First off, in Equation 16, sin and cos represent the trigonometric sine and cosine operators. Furthermore,

[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.

exp, sqrt, and ln in Equation 16 are the operators which represent the exponential function, square root[2], and natural logarithm[3], respectively. Finally, x is the only input variable, and $c_1, \ldots, c_m$ are the constant terminals introduced in Section 3 where m may vary from experiment to experiment. In Table 2, various control parameters are present and identified by the appropriate column header. The first column, *ID*, represents a unique control parameter combination and is denoted by β.

ID	G	l	m	D	Np	F	Cr	ϵ	g_{max}
β_1	G_1	32	5	37	25	0.2	0.1	0.01	10^5
β_2	G_1	64	5	74	25	0.2	0.1	0.01	10^5
β_3	G_2	64	10	74	25	0.4	0.1	0.01	10^5

Table 2: An overview of the possible control parameter combinations used for a training process.

4.2 Results

Table 3 contains an overview of the results obtained from performing several selected experiments. Due to the stochastic behavior of DE-PGEP, 40 independent attempts were made per experiment. Each row presents the results obtained for a single experiment and it is divided in half where the upper entries reflect the results obtained during training and the lower entries express the generalization abilities during testing. The first column, *Env.*, indicates the environment of the experiment, i.e., a tuple of the equation, the training and testing sets, and a control parameter combination. From this point on, any *Min.* column refers to the chromosome with the best fitness value obtained in and over all runs during training, i.e., it had the lowest error on the training set indicating it was the fittest chromosome overall. Similarly, every *Max.* column refers to the chromosome which achieved the worst 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 it was the least fit best chromosome encountered.

The *Fitness of Best Chromosomes* column spans several columns and contains quantities which are expressed in the best fitness values obtained in one or more attempts during training. *Max.* and *Min* have already been described, and the two remaining columns, *Avg.* and *Std. Dev.*, simply report the average and standard deviation of the best fitness values recorded in each and over all 40 trials. The *Expression Length* column presents the size(s) of the corresponding expressions, i.e., *Min.* and *Max.* are the sizes of the overall best and worst programs, and *Avg.* is the average size of the best programs evolved in each run. Quantities which represent the number of constants appearing in the program(s) are also presented in the *Ratio of Constants* column. That is, *Min.*, *Max.*, and *Avg.*, detail the number of unique to total constants utilized in the overall fittest program, overall least fit program, and averages of all best programs, respectively. Since each *Min.* column is expressed in terms of the training set, one will immediately be able to tell whether or not the supposedly fittest chromosome has overfit the training data.

Through an inspection of the results presented in Table 3, we can observe some very interesting things about the behavior of the DE-PGEP:

- 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 standard deviation, one can conclude that the final results are reproducible and not just coincidental.

- Furthermore, the least fit solutions are generally not exceedingly bad, 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 recommendation based on some observations will be made: m should be equal to the number of operators plus the number of input variables. That is, the additional diversity afforded by a larger pool of constants seems helpful when more functionality distinct operators are made available.

4.3 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 GEP/PGEP linear crossover operators exchange contiguous segments of genes between two chromosomes. Now in DE-PGEP, the uniform crossover operator is discrete in nature and only swaps neighboring parameters by chance if Cr is low. So from a GP/GEP/PGEP perspective, the uniform crossover operator with a low application probability clearly resembles that of point mutation. In some sense then, Cr can be viewed as the probability of mutation and F can be seen as the magnitude of the mutation. Yet, the magnitude of mutation has no real parallel in GP/GEP/PGEP.

Additionally, the uniform crossover operator only manipulates one individual directly where a single crossover in GP/GEP/PGEP almost always modifies two. The product of a mutation and crossover operation here can be much more subtle and not nearly as exact. Due to *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 mutations are compounded, that a noticeable difference in the phenotype takes effect [19]. Finally, even though the real-valued representation of DE increases the size of the search space, "flattening" occurs because of redundancy and vast *plateaus* may appear [19]. The transformed landscape may then explain the viability of differential mutation as search operator for DE-PGEP.

Upon close inspection of the control parameter combinations in Table 2, it can be seen that Cr is always equal to 0.1. This was empirically determined and anything above 0.3 was consistently useless. Thus, when $D = 74$, approximately only 7 to 8 parameters are independently adopted

[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 $x = 0$, $\ln(x) = -745$.

Env.	Fitness of Best Chromosomes				Expression Length			Ratio of Constants		
	Min.	Max.	Avg.	Std. Dev.	Min.	Max.	Avg.	Min.	Max.	Avg.
$(11, \alpha_1, \beta_1)$	8.238×10^{-4} 9.140×10^{-4}	9.907×10^{-3} 1.494×10^{-2}	7.433×10^{-3} 9.089×10^{-3}	2.421×10^{-3} 3.489×10^{-3}	17	13	19.750	$\frac{3}{5}$	$\frac{3}{4}$	$\frac{3.950}{6.575}$
$(12, \alpha_2, \beta_2)$	1.866×10^{-3} 1.658×10^{-3}	6.172×10^{-2} 5.872×10^{-2}	9.952×10^{-3} 1.675×10^{-2}	8.674×10^{-3} 4.598×10^{-2}	43	55	32.950	$\frac{5}{16}$	$\frac{5}{24}$	$\frac{4.425}{11.350}$
$(13, \alpha_3, \beta_3)$	3.245×10^{-3} 3.436×10^{-3}	2.476×10^{-2} 2.107×10^{-2}	9.674×10^{-3} 1.441×10^{-2}	2.990×10^{-3} 1.030×10^{-2}	23	14	27.550	$\frac{3}{3}$	$\frac{2}{4}$	$\frac{4.075}{5.750}$
$(13, \alpha_4, \beta_3)$	9.253×10^{-4} 9.018×10^{-4}	1.093×10^{-1} 1.111×10^{-1}	1.274×10^{-2} 2.203×10^{-2}	1.610×10^{-2} 3.773×10^{-2}	10	1	29.900	$\frac{2}{2}$	$\frac{1}{1}$	$\frac{4.725}{6.850}$

Table 3: An overview of the experimental results on the selected benchmarks, Equations 11–13.

into the trial vector. Furthermore, 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 unused parameters which are generally known as *introns*. While introns do not directly contribute to the fitness of the vector, these parameters can be used to alter other vectors by way of differential mutation. That is, other vectors may encode longer programs and therefore the introns can influence other vectors without being explicitly enabled. This is very different from GEP/PGEP where introns are present, but are only activated or deactivated when an actual crossover or mutation occurs.

Intuitively, a higher Cr paired with a moderate F, will generally yield more genotypical differences. So 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 observed here will be illusive. Also, it was continuously observed that as the gene set increased in size, F needed to be increased as well. This is understandable as a larger scaling factor is required to cycle through the available genes and find good or useful innovations.

4.4 Examples of Solutions

Presented in its unsimplified form, Equation 17 is the best program evolved for Equation 11 with a size of 17 and a training fitness of 8.238×10^{-4}. Once simplified into Equation 18, it is immediately apparent that this program is a very good approximation. In fact, it is almost a perfect solution as the training fitness is 9.140×10^{-4}.

$$f_1(x) \approx (x/0.439254) + \big((1.165761 + 1.165761) * (x * 1.165761) * x\big) + (x * 0.864893) \quad (17)$$

$$= 2.717997 * x^2 + 3.141480 * x \quad (18)$$

Instead of evolving just two constants with the exact values, DE-PGEP utilizes three out of the five uniquely available constants and reuses one of them 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 the desired approach. For completeness, this particular solution was obtained using $(11, \alpha_1, \beta_1)$.

Appearing in Equation 19 is the best solution attained using $(12, \alpha_2, \beta_2)$. 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 conservative manner, five unique constants appear a total of 16 times throughout the expression tree. Still, by observing the simplified solution in Equation 20, it can be easily seen that this solution is almost perfect.

$$f_2(x) \approx (c_3 * x) + c_1 + \Bigg(\Big(c_0 + \big((c_0 - x) * x\big) + c_3 + c_0 + x + (c_0 - c_0) + (c_4 * c_1) + c_2 + \big((c_1/(c_3 - c_0)) - x\big)\Big) * \Big(c_1 - x\Big)\Bigg) - c_3$$

$$\text{where } c_0 = 0.349331, c_1 = -0.049276, c_2 = 0.017816, c_3 = 0.505355, c_4 = 0.361927 \quad (19)$$

$$= x^3 - 0.300055 * x^2 - 0.400035 * x - 0.598397 \quad (20)$$

Shifting our attention to the next benchmark function, the best solution discovered for Equation 13 is presented in Equation 21. Unmodified, the original solution has a size of 23 and a training fitness of 3.245×10^{-3}. Initially it appears that this function does not even closely resemble that of the target function. But upon evaluation on the testing set, a fitness of 3.436×10^{-3} is achieved. This solution is a remarkably good approximation given its structure and composition. A simplified and possibly more understandable version of this solution is presented in Equation 22.

$$f_3(x) \approx \sqrt{\sin(3.899029 * 10^{84})} * \ln(x + x) * \sin(x * x) * \Big(0.045098 - \exp\Big(\sqrt{x}/\sin\big(\cos(2.960171 * 10^{72})\big)\Big)\Big) \quad (21)$$

$$= \Big(0.998604 * \ln(2 * x) * \sin\big(x^2\big)\Big) * \Big(0.045098 - \exp\big(1.193832 * \sqrt{x}\big)\Big) \quad (22)$$

Utilizing otherwise unnecessary operators like exp, ln, and sqrt, DE-PGEP is still able to evolve good 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. For those interested, this solution was obtained using $(13, \alpha_3, \beta_3)$.

When considering a wider window, Equation 13 becomes less predictable. But surprisingly, even the unsimplified expression in Equation 23 is very similar in structure to the target function. More interestingly though, the simplified version in Equation 24 is very different from that of Equation 22 which was trained with the exact same control parameters, but on a narrower interval. That is, Equation 24 was discovered with $(13, \alpha_4, \beta_3)$ and not $(13, \alpha_3, \beta_3)$.

$$f_3(x) \approx -0.300998 * x * \sin\Big(x/\cos\big(\ln(-5.645193)\big)\Big) \quad (23)$$

$$= 0.300998 * x * \sin(6.276440 * x) \quad (24)$$

5. CONCLUSIONS & FUTURE WORK

A significantly different approach to *Symbolic Regression* (SR) with a seamlessly integrated constant creation mechanism has been proposed. Abandoning the discrete *genotype* used by *Prefix Gene Expression Programming* and adopting a continuous genotype, 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 be evolved simultaneously. Impressive performance was consistently obtained on four experimental benchmarks, selected solutions were critiqued, and various aspects of *redundant representations* and *neutral mutations* were briefly introduced and related to the proposed method called *Differential Evolution-Prefix Gene Expression Programming* (DE-PGEP). Future work will primarily focus on applying DE-PGEP to real-world problems and exploring more of the consequences, implications, and advantages of redundancy and neutral mutations.

6. REFERENCES

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

[2] Cândida Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

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

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

[5] 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.

[6] 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.

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

[8] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Pratical Approach to Global Optimization*. Springer-Verlag, 2005.

[9] Qiongyun Zhang, Chi Zhou, Weimin Xiao, and Peter C. Nelson. Improving Gene Expression Programming Performance by Using Differential Evolution. In *Proceedings of the 6th International Conference on Machine Learning and Applications (ICMLA'07)*, pages 31–37, Cincinnati, OH, USA, 2007. IEEE Computer Society.

[10] Stefano Cagnoni, Daniel Rivero, and Leonardo Vanneschi. A purely evolutionary memetic algorithm as a first step towards symbiotic coevolution. In *Proceedings of the IEEE Congress on Evolutionary Computation, (CEC 2005)*, pages 1156–1163, Edinburgh, UK, September 2005. IEEE.

[11] 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, San Francisco, CA, USA, July 2001. Morgan Kaufmann.

[12] Gunther R. Raidl. A Hybrid GP Approach for Numerically Robust Symbolic Regression. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 323–328, Madison, WI, USA, July 1998. Morgan Kaufmann.

[13] Thomas Fernandez and Matthew P. Evett. Numeric Mutation as an Improvement to Symbolic Regression in Genetic Programming. In *Evolutionary Programming VII, 7th International Conference, (EP98)*, pages 251–260. Springer, 1998.

[14] Conor Ryan and Maarten Keijzer. An Analysis of Diversity of Constants of Genetic Programming. In *Genetic Programming, 6th European Conference, EuroGP 2003*, pages 404–413. Springer, 2003.

[15] 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, WA, USA, July 2004.

[16] Michael O'Neill, Ian Dempsey, Anthony Brabazon, and Conor Ryan. Analysis of a Digit Concatenation Approach to Constant Creation. In *Genetic Programming, 6th European Conference, EuroGP 2003*, pages 173–182. Springer, 2003.

[17] Maarten Keijzer. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In *Genetic Programming, 6th European Conference, EuroGP 2003*, pages 70–82. Springer, 2003.

[18] Michael O'Neill and Anthony Brabazon. Grammatical Differential Evolution. In *Proceedings of the 2006 International Conference on Artificial Intelligence (ICAI 2006)*, pages 231–236. CSREA Press, 2006.

[19] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer-Verlag Berlin Heidelberg, second edition, 2006.

[20] Rob Shipman, Mark Shackleton, and Inman Harvey. The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal*, 18(4):103–111, 2000.