Using Differential Evolution for GEP Constant Creation

Qiongyun Zhang
Department of Computer Science
University of Illinois at Chicago
Chicago, IL, 60607, USA
qzhang@cs.uic.edu

Chi Zhou
Realization Research Center of Motorola Labs
Schaumburg, IL 60196, USA
Chi.Zhou@motorola.com

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

ABSTRACT

Gene Expression Programming (GEP) is a new evolutionary algorithm that incorporates both the idea of simple, linear chromosome of fixed length used in Genetic Algorithms (GAs) and the ramified structure of different sizes and shapes used in Genetic Programming (GP). Same as other genetic programming algorithms, GEP has difficulty finding appropriate numeric constants for terminal nodes in the expression trees. In this work, we describe a new approach of constants generation using *Differential Evolution* (DE), which is a simple real-valued GA that has been proved robust and efficient on parameter optimization problems. Our experimental results on two symbolic regression problems show that the approach significantly improves the performance of the GEP algorithm. The proposed approach can be easily extended to other Genetic Programming variants.

General Terms

Genetic Algorithm, Genetic Programming, Gene Expression Programming, Differential Evolution

1. INTRODUCTION

Gene Expression Programming (GEP) is a new evolutionary algorithm for automatic creation of computer programs, first proposed in [6] by Cândida Ferreira. Unlike traditional GP, in GEP, computer programs are represented as linear strings of fixed length called *chromosomes* (genotype) which subsequently are mapped into *Expression Trees* (ETs) (phenotype) of different sizes and shapes for fitness evaluation. The search space is separated from the solution space, which results in unconstrained search of the genome space while still ensuring validity of the program's output. Due to the linear fixed-length genotype representation, genetic manipulation becomes much easier than that on parse tress in GP. Compared with traditional GP, the evolution of GEP has more flexibility and power in exploring the entire search space. GEP methods have performed well for solving a large variety of problems, including symbolic regression, optimization, time series analysis, classification, logic synthesis and cellular automata, etc. [6]. Zhou, et al. applied a different version of GEP and achieved significantly better results on multi-class classification problems, compared with traditional machine learning methods and GP classifiers. Instead of the original head-tail method in [6], their GEP implementation uses a chromosome validation algorithm to dynamically determine the feasibility of any individual generated, which results in no inherent restrictions in the types of genetic operators applied to the GEP chromosomes, and all genes are treated equally during the evolution [16] [17]. The work presented in this paper is based on this revised version of GEP.

Despite its flexible representation and efficient evolutionary process, GEP still has difficulty discovering suitable function structures, because the genetic operators are more disruptive than traditional tree-based GP, and a good evolved function structure is very likely to be destroyed in the subsequent generations [8]. Different tentative approaches have been suggested, including multi-genetic chromosomes, special genetic operators, and constant creation methods [6]. Our attention was drawn to constant creation methods due to their simplicity and the potential benefits. It is assumed that local search effort for finding better combinations of numeric constants on top of an ordinary GEP process would help improve the fitness value of the final best solution. In this paper, we propose a new constant creation approach for GEP using *Differential Evolution* (DE), and have tested them on two typical symbolic regression problems. Experimental results have demonstrated that the approach can achieve significant improvement on GEP performance, and is able to find optimal constants for a given GEP formula structure.

The rest of the paper is organized as following: Section 2 describes the GEP algorithm and provides a brief review on related work on constant generation. Section 3 briefly reviews the Differential Evolution algorithm , and describes our approach of embedding DE into GEP for constant tune-up. Experiments and results with our new approach are presented and analyzed in Section 4. Section 5 summarizes this research work and ideas for future work.

2. RELATED WORK

2.1 Gene Expression Programming(GEP)

Gene Expression Programming is a genotype/phenotype system that evolves computer programs of different sizes and shapes encoded in linear chromosomes of fixed length [5]. When GEP is used to solve a problem, usually five components are specified: the function set, the terminal set that includes problem-specific variables and pre-selected constants, fitness function, control parameters, and stop condition. A chromosome in GEP is a character string of fixed length,

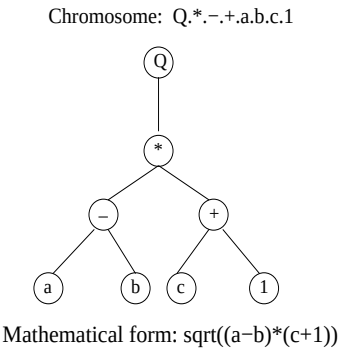

Figure 1: An example of GEP chromosome, the corresponding expression tree and the mathematical form.

which can be any element (called gene) from the function set or the terminal set. Figure 1 shows an example of a chromosome of length 8, using function set {+, -, *, Q} where Q represents the math function *sqrt*, and the terminal set {a, b, c, 1}. This is referred to as the *Karva Notation*, or *K-expression* [4]. A K-expression can then be mapped into an *Expression Tree* (ET) following a breadth-first procedure and be further written in a mathematical form as shown in Figure 1.

A chromosome is valid only when it can map into a legal ET within its length limit. Therefore all the chromosomes randomly generated or reproduced by genetic operators are subject to a validity test, in order to prevent illegal expressions from being introduced into the population [16].

To start, the GEP algorithm initially generates a random population of linear fixed-length chromosomes. Then the chromosomes are represented as ETs, evaluated based on a user defined fitness function, and selected according to fitness to reproduce with modification. The individuals in this newly generated population are subjected to the same development process until a pre-specified number of generations are completed, or a solution has been found. In GEP, the selection procedures are often determined by roulette-wheel sampling with elitism [9] based on individuals fitness, which guarantees the survival and cloning of the best individual to the next generation. Variation in the population is introduced by applying genetic operators, i.e., crossover, mutation and rotation [6], to selected chromosomes, which usually drastically reshape their corresponding ETs. Refer to [17] for detailed description on GEP.

2.2 Related Work on Constant Creation

The problem with constant creation has received quite a bit research and discussion in GP community, as it is difficult for GP to find good numeric constants for the terminal nodes in s-expression trees. This is one of the issues that GP has to overcome in order to achieve greater efficiency for complex applications. Ryan and Keijzer [12] analyzed the density and diversity of constants over generations in GP. They also introduced two simple constant mutation techniques, creep mutation and uniform/random mutation, and experiments with these techniques have shown better search performance of GP.

Some other approaches to constant creation procedure in GP can be categorized as local search algorithms. Several researchers have tried to combine hill climbing, simulated annealing, local gradient search [14] [6] [15] [3], and other stochastic techniques to GP to facilitate finding useful constants for evolving solutions or optimizing extra parameters. Although meaningful improvements have been achieved, these methods are complicated to implement compared with simple mutation techniques. Furthermore, an overly constrained local search method would possibly reduce the power of the free style search inherent in the evolutionary algorithms. A novel view of constant creation by a digit concatenation approach is presented in [10] for Grammatical Evolution (GE). Most recently, a new concept of linear scaling is introduced in [7] to help the GP system concentrate on constructing an expression that has the desired shape. However, this method is suitable for finding significant constants for evolved expressions that are approximately linearly related to their corresponding target values, but it is generally ineffective for identifying other candidates with good function shapes. The idea of embedding a GA process into GP has been proposed and implemented, and experiments have shown this approach is very good at finding models that fit the training data [2] [1].

Since the invention of GEP, constant creation techniques have received attention in the research literature. Ferreira introduced two approaches for symbolic regression in the original GEP [5]. One approach does not include any constants in the terminal set, but relies on the spontaneous emergence of necessary constants through the evolutionary process of GEP. The other approach involves the ability to explicitly manipulate random constants by adding a random constant domain Dc at the end of chromosome. Previously Li et. al. [8] proposed several GEP constant creation methods similar to creep and random mutation as described in [12], but in a greedy fashion. Their experimental results demonstrated that the constant tune-up process for the entire population can significantly improve the average fitness of the best solutions.

3. DIFFERENTIAL EVOLUTION FOR GEP CONSTANT CREATION

3.1 Differential Evolution

First proposed in [13], *Differential Evolution*(DE), is an exceptionally simple vector-based evolutionary algorithm. The algorithm optimizes a system by choosing appropriate system parameters that are represented as a real-valued vector. Despite its simple form, it has been proved to be effective and robust at numerical optimization and is more likely to find a function's true global optimum .

In DE, a population of solution vectors are successively updated by vector operations that performs linear recombina-

tion such as addition, subtraction, and component swapping, until the population converges. It starts with NP randomly generated solution n-dimensional vectors,

$$X_i = (x_{i1}, x_{i2}, ...x_{in}), i = 1, ..., NP,$$

as the initial population. At each generation, two operations *mutation* and *crossover* are applied to each individual vector in the current population, to generate an offspring. Following that, a selection between each individual and its corresponding offspring is performed, based on their objective value computed by user-defined objective function. The one that yields better objective value is select as an individual in the next population, and the other one is eliminated.

For each solution vector, first a *mutant vector*,

$$V_i = (v_{i1}, v_{i2}, ...v_{in}), i = 1, ..., NP,$$

is formed using one of the following schemes [11] [13]:

$$\begin{aligned} V_i &= X_{r1} + F(X_{r2} - X_{r3}) && (1)\\ V_i &= X_{best} + F(X_{r2} - X_{r3}) && (2)\\ V_i &= X_i + F(X_{best} - X_i) + F(X_{r1} - X_{r2}) && (3)\\ V_i &= X_{best} + F(X_{r1} - X_{r2}) + F(X_{r3} - X_{r4}) && (4)\\ V_i &= X_{r1} + F(X_{r2} - X_{r3}) + F(X_{r4} - X_{r5}) && (5) \end{aligned}$$

where X_{best} is the best individual in the current population, and X_{r1},X_{r2},X_{r3},X_{r4}, X_{r5} are mutually distinct randomly chosen vectors from the population. F is a real and constant scaling factor that controls the amplification of the difference between two vectors, and F usually falls in the range of $(0, 2)$ [13]. The scheme used in our work is equation 4.

Following the mutation operation, the crossover operation is applied to each individual X_i and its mutant vector V_i, so as to generate a new vector U_i called *trial vector* which is the offspring of X_i. For each vector component, draw a random number $rand_j$in the range of $[0, 1]$. The trial vector is produced, with

$$U_{ij} = \begin{cases} V_{ij} & \text{if } rand_j \leq CR; \\ X_{ij} & \text{if } rand_j > CR. \end{cases}$$

where CR is a user-defined crossover threshold between 0 and 1. Therefore the trial vector has some components from the mutant vector, and some from the parent vector X_i. To ensure some crossover, one component of U_i is selected at random to be from the mutant vector V_i [13].

The selection between the vector X_i and its offspring Ui, a comparison is performed based on a user-defined objective function *f*. In most cases, the objective function transforms the optimization problem into a minimization task [13]. In other words, the vector that yield a smaller objective function value is considered a better solution.

$$X_i^{G+1} = \begin{cases} U_i & \text{if } f(U_i) < f(X_i); \\ X_i & \text{if } f(U_i) \geq f(X_i). \end{cases}$$

DE has the advantages of simple structure, speed and robustness. Therefore DE has been widely used in optimization problems with real variables and many local optima, and proved to be a very efficient algorithm.

3.2 Using DE for GEP Constant Creation

Given the power of DE on parameter optimization, we could utilized DE to tune up the constant creation process in GEP. A special gene named *Random Number Generator* (RNG) is introduced into the GEP terminal set to replace a list of constant genes, e.g., {1, 2, 3, 5, 7,...}. Different instance of this RNG gene will have different values which are randomly initialized in the range of [0, 1]. The idea behind fusing the DE algorithm into GEP is to extract all random number generators from a GEP chromosome as a parameter vector, and apply a separate DE process to optimize those parameters using the same fitness evaluation function as in GEP. As a result, the GEP learning process is divided into two phases: in the first phase, the GEP algorithm focuses on searching for the best solution structure, while in the second phase, DE focuses on optimizing the constant parameters given the fixed solution formula. This can be implemented as such: for every chromosome in the GEP population, when evaluating the fitness, apply the DE algorithm to search for the best constants in that chromosome without changing the structure of the chromosome. After the DE process is finished, the parameter vector with the best DE fitness will be assigned to the chromosome. Figure 2 illustrats the new GEP algorithm with DE tuning-up the constants.

With the structure of the formula fixed, the application of DE algorithm tunes up the constants. Most importantly, by making use of DE algorithm, the choices of constants are not only limited to prime numbers as in conventional genetic programming algorithms. Instead, real number constants are enabled in a GEP formula.

4. EXPERIMENTS

In order to test if the DE algorithm can help in finding constants in the training process of GEP, we have selected two problems that have have been studied by other researchers with regard to constant creation issue in GP or GEP. The experimental results presented in previous work by Li et al. [9] will be used as benchmark for comparison.

4.1 Experiment Settings

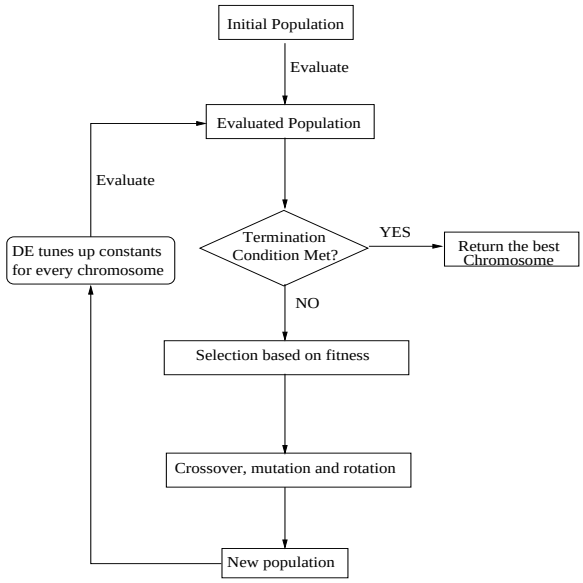

Figure 2: GEP with DE embedded.

The two datasets tested in the experiments are regression problems. One is a simple polynomial with real-number coefficients (6). A set of 21 fitness cases equally spaced along the x axis from -10 to 10 are chosen for this polynomial. The second equation is a V-shaped function which not only has real number coefficients, but also exhibits complex functionality and structure (7). That adds to the difficulty for GEP to obtain a close approximation to the target function.

$$y = x^3 - 0.3x^2 - 0.4x - 0.6 \tag{6}$$

$$y = 4.251a^2 + \ln(a^2) + 7.243e^a \tag{7}$$

For GEP control parameters, same as in [8], we used 100 for the GEP chromosome length, 500 for the GEP population size, 1000 for the maximum number of generations, 0.7 as the crossover probability and 0.02 as the mutation probability. The roulette-wheel selection with elitism is utilized as the selection method based on the fitness function calculated by (8), where $fitness_i$ indicates the fitness function for the ith individual in the population, minR is the best (or minimun) residual error obtained so far, and $ResError_i$ is the individual's residual error. Note that this is the fitness function used for selection, and the fitness of a chromosome is measured by its residual error which is better when smaller.

$$fitness_i = minR/(minR + ResError_i) \tag{8}$$

For the experiments on the two datasets, the terminal set includes the input attributes(the variable x or a as in specific problems). No constants are selected beforehand. Instead, they are represented by $RNGs$(Random Number Generators) that have an initial random real value, and will be optimized by the DE algorithm. Due to our prior knowledge about the two benchmark problems, the function set used for first dataset is $\{+, -, *, /\}$, and $\{+, -, *, /, \log, \exp, power, \sin, \cos\}$ for the V-shaped function problem, where log is the natural logarithm, exp represents e^x and $power(x, y)$ represents x^y. The stopping condition for both datasets are the same: either the residual error of the obtained formula on the dataset is less than 10^{-4} or the number of generations has reached 1000.

As for the parameters of the DE algorithm, we chose 128 as the size of vector population NP, 0.2 as the crossover threshold CR, 0.5 as the amplification factor F, and the number of generations is 500.

Several different experiments have been performed. We start with the experiment where in every generation of GEP, DE is applied to optimize the constants for each individual chromosome in the population. Due to the extra time required by the computational intensive process, this experiment is only used with the polynomial function problem. The second experiment is similar to the first one, except that instead of having DE applied in every generation, we only have DE process turned on for every 5th generation. Also we experimented with DE in every 10th generation. The purpose of the latter experiments is two-fold: one is to decrease the computational cost, and the other is to find out if the DE process would still help in choosing constants even if it's not applied in every generation. This experiment has been performed on both datasets. Taking into account the stochastic behavior of the GEP algorithm, these experiments are repeated 30 times, and the results are averaged.

4.2 Experiment Analysis

We compare the results of the three experiments on the two problems, to get some insight into the effect of different strategies on the combination of DE process with GEP.

In Table [1], the *best residual* is the best(smallest) residual error of a chromosome among all of the final best individuals throughout the 30 runs. *Average of best residuals* is the average of all 30 final best residual errors. *Average tree size* refers to the average size of the expression trees, i.e., the number of nodes in the tree, of the 30 best residuals.

The first set of statistics are obtained from the experiment where the DE process is turned on in every GEP generation. As mentioned above, the first approach where DE is invoked every generation is computationally expensive and takes significant extra time, therefore, this experiment is only done with the polynomial function dataset. The second and second set of statistics correspond to the experiment where the DE process is applied in every 5th and 10th GEP generation respectively. The last set of numbers are the best results on the same dataset, reported in [8], where constant creation is done using non-DE approach. The following observations can be made from Table [1]:

- GEP with DE embedded shows significantly better results in terms of *Best residual*, compared with the results in [8]. Having the DE process applied in every GEP generation provides better results than in every 5th and 10th generation.
- The experiments show better results in terms of *average of best residuals*. But having DE in every generation produces much better result than having DE applied less frequently. The improvement on the polynomial problem is more obvious than that on the V-shaped problem.
- Our approach generates significantly smaller expression trees on the polynomial problem than previous work in [8], with the first experiment slightly outperforming the other two experiment. Expression trees for the V-shaped problem almost have the same average size as those in [8].

The conclusion drawn from the experiment is that having DE embedded into GEP helps produce better approximation of the optimal solution. The more frequent DE is invoked in the evolution procedure, the better results GEP can produce.

4.3 A Closer Look at Constants

Since GEP, like any other genetic programming algorithms, can produce a mathematical formula that is an approximation of the target solution for a regression problem, it provides us with a way to examine the constants tuned up by DE process. To analyze the effect of DE on the choices of constants, for each of the two problems, we pick a good and a bad example of their GEP formulas and take a closer look at them.

The polynomial regression problem is relatively simple and most of its GEP formulas in the two experiments are almost perfect. A good example is shown in (9), which is equivalent to the equation in (10). Its corresponding best residual error is $9.463 * 10^{-5}$. As shown in (10), the constants are almost the same as those in the target function in equation (6), with slight differences less than 10^{-4}. A relatively bad example of GEP formula for the same problem that produces best residual error 0.1309 is shown in equation(11), which is equivalent to (12). The constants in (12) are still very close to those in the target function, with differences less than 10^{-3}.

$$y = (x*(-0.399962-((-1*x)*(x-0.300001))))-0.599972 \tag{9}$$

$$y = x^3 - 0.300001x^2 - 0.399962x - 0.599972 \tag{10}$$

$$y = (x*(0.501598+(((2.180309/x)*(x+0.243787)) +x)-1.855177))*(x-1.126747) \tag{11}$$

$$y = x^3 - 0.300017x^2 - 0.400092x - 0.59878 \tag{12}$$

Comparing equation (9) with equation (11), the difference in their structures are noticeable, but both of them can be converted into exactly the same structure of the target function. After the conversion, the constants in both formulas (10) and (12) are very close to those in the target function, but the real numbers in (12) are less precise than those in (10).

The V-shaped function is much more complicated. As shown in the previous sections, the performance of our experiment on this dataset was far from perfect, although decent improvement was shown. The GEP formula corresponding to the best residual error on this dataset is shown in (13), which is equivalent to the equation in (14). This is a close approximation of the target function in equation (7). Equation (15) is a worse example of the formula for this dataset, whose corresponding best residual error is 0.5360. Notice the structure of equation (15), it is far from the structure of the target function (7), and no matter how we try to convert this equation, its equivalence is nothing like (7).

$$y = 7*(e^x/0.138066)-((\log(x)/(-0.500009)) -e^{\log(x)-(-1.001507*(1.445251+\log(x)))}) \tag{13}$$

$$y = 7.242913e^x + \log(x^{1.999964}) + 4.252168e^{1.447429} \tag{14}$$

$$y = \log(e^{x/0.042217}+(-1.865846+x*(-119.921534))) \tag{15}$$

The above analysis shows that when GEP finds a structure close to that of the target function, DE oftentimes can find proper constants for that arbitrary formula. While GEP fails to find an appropriate formula structure through its evolution, the power of DE can not be made full use of in optimization, even though it might still be able to find the best constants given a bad structure of the formula. Therefore, GEP and DE are dependent on each other in the process of evolution, to find a good approximation. A good GEP formula structure may brings out the power of DE, and proper constants found by DE may simplify the structure of a GEP formula.

5. CONCLUSIONS AND FUTURE WORK

In this work, we have explored a new way of constant creation for GEP using Differential Evolution (DE), which improves the performance the GEP algorithm. The experiment results reported show that the embedded DE process has a very strong capability of finding optimal constants given an arbitrary structure of GEP formula, and finding a close approximation to the target regression function. In turn, that

Dataset	Statistics	Every Gen.	Every 5th Gen.	Every 10th Gen.	Best in [8]
Polynomial	Best residual	$5.901 * 10^{-6}$	$1.0776 * 10^{-5}$	$1.989 * 10^{-5}$	0.157
	Avg. of best residuals	$2.913 * 10^{-4}$	0.005128	0.0223	0.966
	Avg. tree size	22.2	22.8667	23.9333	37.3
V-shaped	Best residual	0.0150	0.0089	$5.1425 * 10^{-4}$	1.038
	Avg. of best residuals	0.1382	0.2572	0.3404	1.863
	Avg. tree size	31.5333	26.5333	28.8667	28.4

Table 1: Comparison of the results of the experiment on both datasets.

improves the performance of GEP algorithm, in terms of residual error. It also may simplify the structure of a GEP formula, by reducing the size of the expression tree. However, the DE process can't optimize constants by itself. To what extend its power can be utilized and it can contribute to finding the optimal solution largely depends on the structure of a GEP formula. If GEP fails to evolve a structure close to the target formula, DE may not be able to find constants that are close to those in the target function, even if it may still find the optimal constants for that arbitrary structure. On average, the DE controlled tuneup process improves the performance of GEP significantly, and is able to find constants close enough to the optimal ones.

In our future work, we plan to further examine the DE embedded GEP algorithm with large scale regression problems. To do this, we need to first address the drawback of this approach, i.e., the extra computation caused by the DE process. Therefore, it will be one of our main focuses to reduce the computational cost. We have experimented with DE not invoked in every generation of GEP, and it still improves the GEP algorithm significantly, although not as good as when DE is invoked in every generation. Another alternative is to apply DE only when the fitness of the GEP population is improved, compared with earlier generation. Currently, DE is applied on every chromosome in a population to tune up their constants, which is very time consuming. One alternative to this is to apply DE on the best individual chromosome or an elite group that contains set of individuals with relatively high fitness values. These variations would save some computation but may still has the advantage.

6. ACKNOWLEDGMENTS

We appreciate the Physical Realization Research Center of Motorola Labs for providing funding for this research work.

7. ADDITIONAL AUTHORS

Additional authors: Weimin Xiao (Physics Realization Research Center of Motorola Labs, Schaumburg, IL, 60196, USA, email: `awx003@motorola.com`).

8. REFERENCES

[1] S. Cagnoni, D. Rivero, and L. Vanneschi. A purely evolutionary memetic algorithm as a first set towards symbiotic coevolution. In *IEEE World Congress on Evolutionary Computation(CEC2005)*, pages 1156–1163, Edinburgh, United Kingdom, September 2005.

[2] H. Cao, L. Kang, Y. Chen, and J. Yu. Evolutionary modeling of systems of ordinary differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 1(4):309–337, 2000.

[3] A. I. Esparcia-Alcazar and K. Sharman. Learning schemes for genetic programming. In *Late Breaking Papers at the Annual Genetic and Evolutionary Computation Conference*, pages 57–65, Stanford University, California, 1997.

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

[5] C. Ferreira. Function finding and the creation of numerical constants in gene expression programming. In *The 7th Online World Conference on Soft Computing in Industrial Applications*, Septempter - October 2002.

[6] C. Ferreira. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Angra do Heroismo, Portugal, 2002.

[7] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming.*, volume 2610 of *LNCS*, pages 70–82, 2003.

[8] X. Li, C. Zhou, P. C. Nelson, and T. M. Tirpak. Investigation of constant creation techniques in the context of gene expression programming. In M. Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, July 2004.

[9] M. Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. MIT Press, 1996.

[10] M. ONeill, I. Dempsey, A. Brabazon, and C. Ryan. Analysis of a digit concatenation approach to constant creation. In *Proceedings of European Conference on Genetic Programming*, volume 2610 of *LNCS*, pages 173–182, Essex, April 2003.

[11] K. E. Parsopoulos, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Vector evaluated differential evolution for multiobjective optimization. In *IEEE Congress on Evolutionary Computation (CEC 2004)*, pages 204–211, Portland, Oregon, USA, 2004.

[12] C. Ryan and M. Keijzer. An analysis of diversity of constants of genetic programming. In *Proceedings of*

European Conference on Genetic Programming, volume 2610 of *LNCS*, pages 404–413, Essex, April 2003.

[13] R. Storn and K. Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[14] N. M. I. the Discovery of Numeric Constants in Genetic Programmin. Matthew evett and thomas fernandez. In *the Third Annual Genetic Programming Conference*, pages 66–71, Madison, Wisconsin, 1998.

[15] A. Topchy and W. F. Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *the Genetic and Evolutionary Computation Conference*, pages 155–162, San Francisco, California, 2001.

[16] C. Zhou, P. C. Nelson, W. Xiao, and T. M. Tirpak. Discovery of classification rules by using gene expression programming. In *Proceedings of the International Conference on Artificial Intelligence (ICAI02)*, pages 1355–1361, Las Vegas, U.S.A., June 2002.

[17] C. Zhou, W. Xiao, P. C. Nelson, and T. M. Tirpak. Evolving accurate and compact classification rules with gene expression programming. *IEEE Transactions on Evolutionary Computation*, 7(6):519–531, 2003.