Improving Gene Expression Programming Performance by Using Differential Evolution

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

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

Weimin Xiao
Physical & Digital Realization
Research Center of Motorola Labs
Schaumburg IL, 60196, USA
awx003@morotola.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 an evolutionary algorithm that incorporates both the idea of a simple, linear chromosome of fixed length used in Genetic Algorithms (GAs) and the tree structure of different sizes and shapes used in Genetic Programming (GP). As with other GP algorithms, GEP has difficulty finding appropriate numeric constants for terminal nodes in the expression trees. In this work, we describe a new approach of constant generation using Differential Evolution (DE), a real-valued GA robust and efficient at parameter optimization. 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 variations.

1. Introduction

Gene Expression Programming (GEP) is an evolutionary algorithm for automatic creation of computer programs, proposed in [5] by Cândida Ferreira. In GEP, computer programs are represented as linear strings of fixed length called *chromosomes* which subsequently are mapped into *Expression Trees* (ETs) of different sizes and shapes for fitness evaluation. Due to the linear fixed-length genotype representation, genetic manipulation becomes much easier than that on parse trees in GP. GEP has performed well for solving a large variety of problems, including symbolic regression, optimization, time series analysis, classification, logic synthesis and cellular automata, etc. [5]. Zhou, et al. [14] applied GEP to multi-class classification problems and achieved significantly better results, compared with traditional GP and machine learning methods.

Despite its flexible representation and efficient evolutionary process, GEP still has difficulty discovering suitable constants like other GP algorithms. In this paper, we describe a new constant creation approach for GEP using *Differential Evolution* (DE), and have tested this approach on two symbolic regression problems. Experimental results have demonstrated that the approach can find optimal constants for a given GEP formula structure, and significantly improves GEP performance. Preliminary results for this work have been reported in [13].

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

2. Related Work

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

character string of fixed length, which consists of any element from the function and terminal sets. A chromosome is mapped into an *Expression Tree* (ET) following a breadth-first procedure and can be further written in a mathematical form. A chromosome is valid only when it can map into a legal ET within its length limit. Therefore all the chromosomes are subject to a validity test, in order to prevent illegal expressions in the population [14]. To start, the algorithm generates a random population of chromosomes. The chromosomes are represented as ETs, evaluated based on a user defined fitness function, and selected for reproduction with modification. The individuals in this newly generated population are subjected to the same development process until the termination condition is satisfied. In GEP, the selection procedures are often determined by roulette-wheel sampling with elitism based on individuals fitness, which guarantees the survival of the best individual to the next generation. Variation in the population is introduced by genetic operators, i.e., crossover, mutation and rotation [5]. Refer to [14] for detailed description on GEP.

Finding good numeric constants for terminal nodes in parse tree is one of the issues that GP has to overcome. Ryan and Keijzer [10] introduced two simple constant mutation techniques, creep mutation and uniform/random mutation. Several researchers have tried to combine hill climbing, simulated annealing, local gradient search [5] [12] [3], and other stochastic techniques to GP to facilitate finding useful constants. A novel view of constant creation by a digit concatenation approach is presented in [8] for Grammatical Evolution (GE). Most recently, a new concept of linear scaling is introduced in [6] to help constructing an expression that has the desired shape. The idea of embedding a GA into GP has also been proposed [2] [1].

Since the invention of GEP, Ferreira has introduced two approaches for symbolic regression in the original GEP [4]. One approach does not include any constants, but relies on the spontaneous emergence of necessary constants through the evolution. The other approach explicitly manipulates constants by adding a random constant domain Dc at the end of chromosome. Previously Li et. al. [7] proposed several constant creation methods similar to creep and random mutation in [10], but in a greedy fashion. Their experimental results demonstrated significant improvement.

3 Differential Evolution for GEP Constant Creation

First proposed in [11], *Differential Evolution*(DE) is a vector-based evolutionary algorithm. The algorithm optimizes a system by choosing appropriate system parameters represented as a real-valued vector.

In DE, a population of solution vectors are successively updated by vector operations until the population converges. It starts with *P* randomly generated n-dimensional vectors $X_i = (x_{i1}, x_{i2}, ...x_{in}), i = 1, ..., P$. At each generation, *mutation* and *crossover* are applied to every individual vector, to generate an offspring. A selection between an individual and its offspring is performed based on their objective value computed by an objective function. The one that yields better objective value is included into the next population, and the other one is eliminated.

For each vector, first a *mutant vector* $V_i = (v_{i1}, v_{i2}, ...v_{in}), i = 1, ..., P$, is formed using one of the following schemes [9] [11]:

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

where X_{best} is the best individual by far, and X_{r1},X_{r2},X_{r3},X_{r4}, X_{r5} are mutually distinct vectors randomly chosen from the population. *F* is a scaling factor that controls the amplification of the difference between two vectors, and *F* usually falls in $(0, 2)$ [11]. Then, crossover is applied to X_i and V_i, to produce an offspring U_i called *trial vector*. 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. The trial vector has components from both X_i and V_i. To ensure minimum crossover, one component of U_i is guaranteed to be from V_i. Then the objective values of X_i and Ui are computed. In most cases, the objective function transforms the optimization problem into a minimization task [11]. Therefore, the vector that yield a smaller objective function value is considered better.

DE has the advantages of simple structure, speed and robustness, and has been widely used in optimization problems with real variables and many local optima. To utilize DE for constants optimization in GEP, a special gene named *Random Number Generator* (RNG) is introduced into the GEP terminal set to replace a list of constant genes. Instances of the RNG gene have random initial values between 0 and 1. All RNGs in a GEP chromosome form a parameter vector, and a separate DE process is applied to optimize those parameters using the same fitness function as in GEP. Each generation is now divided into two phases: in the first phase, GEP focuses on searching for solution structure, and in the second phase, DE focuses on optimizing the constants given a solution structure. Figure 1 illustrates the

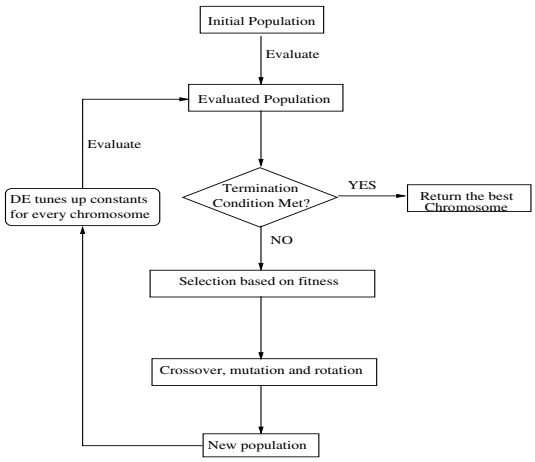


Figure 1. GEP with DE embedded.

	P	*Gen.*	*Crossover*	*Mutation*	*CR*	*F*
GEP	500	1000	0.7	0.02	NA	NA
DE	128	500	NA	NA	0.2	0.5

Table 1. GEP and DE control parameters.

new GEP algorithm with DE embedded. During the population evaluation, for every chromosome, DE searches for the best constants and the optimal numbers found are assigned to the chromosome.

4 Experiments

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 problem is a V-shaped function with real number coefficients, complex functionality and structure (7). The dataset has 20 random fitness cases chosen from [-1, 1] of the variable a.

$$y = x^3 - 0.3x^2 - 0.4x - 0.6 \quad (6)$$

$$y = 4.251a^2 + \ln(a^2) + 7.243e^a \quad (7)$$

Table 1 shows the GEP and DE control parameters used in our experiments.

The GEP terminal set includes the input attributes(the variable x or a as in specific problems). Due to our prior knowledge about the problems, the function set for polynomial dataset is $\{+, -, *, /\}$, and $\{+, -, *, /, \log, \exp, power, \sin, \cos\}$ for the V-shaped function. The GEP termination condition is: either the residual error of the best individual is less than 10^{-4} or the number of generations has reached 1000. The DE algorithm is terminated after 500 DE generations. The roulette-wheel selection with elitism is used as the selection method based on the fitness function calculated by (8), where $fitness_i$ indicates the fitness for the *i*th individual in the population, minR is the best (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.

$$fitness_i = minR/(minR + ResError_i) \quad (8)$$

Three sets of experiments have been performed, starting with the experiment where in every generation of GEP, DE is applied to optimize the constants for each individual chromosome in the population. This experiment is computationally expensive due to the extra time required for DE. In order to decrease the computational cost, two alternatives were tried. One was to turn on DE at a lower frequency instead of at every GEP generation, and the other was to apply DE at every generation, but only to a top portion of the population with lower residual error. The scheme chosen for the DE algorithm is Equation (4). We also experimented the five different DE schemes in Equation (1) – (5), with DE applied at every 10^{th} generation. All these experiments were repeated 30 times. The experimental results in previous work by Li et al. [7] are used for comparison. The three criteria that will be used throughout this section for performance evaluation are defined as follows:

1. **Best Residual**: smallest residual error among the 30 best-of-run;
2. **Average of the Best Residuals**: average error of the 30 best-of-run;
3. **Average Tree Size**: average size of the ETs of the 30 best-of-run.

Table 2a lists the generation-based experiment results and the results in [7]. Table 2b shows the statistics for the population-based experiments. The following observations can be made from Table 2a and 2b:

- Significantly better results in terms of *Best residual*, compared with the results in [7]. Having the DE process applied at smaller intervals or to larger portion of GEP population produces better residuals.
- Smaller *average of best residuals*. The more frequently DE is invoked to larger subset of the population, the better result achieved. The improvement on the polynomial problem is more obvious than that on the V-shaped problem.
- Significantly smaller expression trees on the polynomial problem and comparable average tree size for the V-shaped problem as in [7].

	Statistics	Every Generation	Every 5th Gen.	Every 10th Gen.	Best in [7]
P	Best residual	5.901e-6	1.078e-5	1.989e-5	0.157
	Avg. of best residuals	3.222e-4±3.558e-4	5.135e-3±8.556e-3	2.229e-2±1.768e-2	0.966±0.167
	Avg. tree size	22.200	22.867	23.933	37.300
V	Best residual	1.969e-2	8.866e-3	5.142e-4	1.038
	Avg. of best residuals	1.731e-1±7.061e-2	2.572e-1±8.672e-2	3.404e-1±9.878e-2	1.863±0.127
	Avg. tree size	26	26.533	28.867	28.4

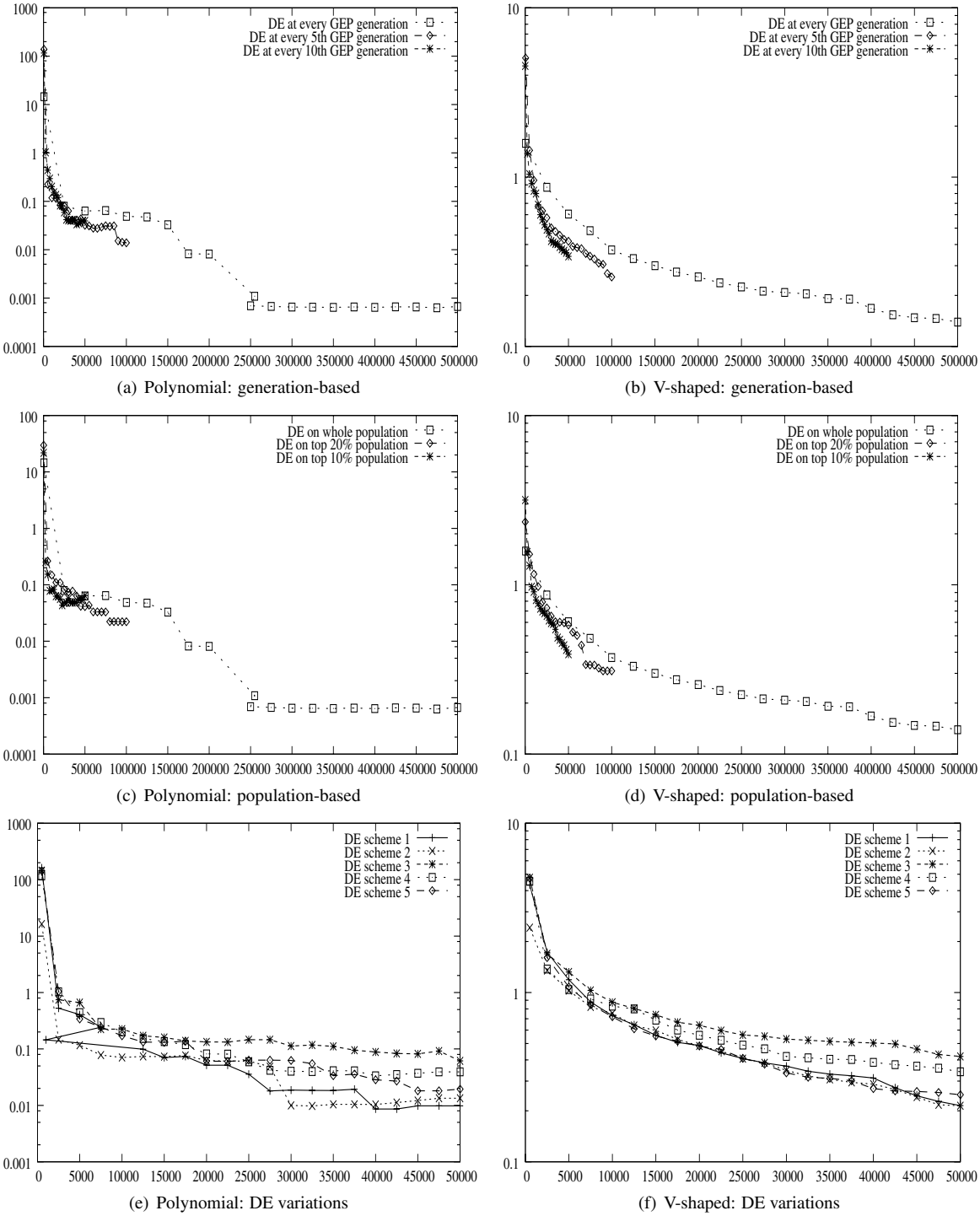
Table 2a. DE applied at different intervals. P - polynomial dataset. V - V-shaped dataset.

	Statistics	Entire population	Top 20% Population	Top 10% Population	Best in [7]
P	Best residual	5.901e-6	2.896e-5	1.523e-5	0.157
	Avg. of best residuals	3.222e-4±3.558e-4	8.909e-3±1.187e-2	2.578e-2±2.186e-2	0.966±0.167
	Avg. tree size	22.2	22.133	23.8	37.3
V	Best residual	1.969e-2	4.593e-3	1.063e-2	1.038
	Avg. of best residuals	1.731e-1±7.061e-2	3.511e-1±9.498e-2	3.89e-1±9.185e-2	1.863±0.127
	Avg. tree size	26	26.667	27.6	28.4

Table 2b. DE applied to subsets of the GEP population.

	statistics	scheme 1	scheme 2	scheme 3	scheme 4	scheme 5
P	Best residual	3.827e-5	4.614e-5	1.763e-5	1.989e-5	2.615e-5
	Avg. best	4.614e-3 ± 8.538e-3	4.518e-3 ± 8.544e-3	2.868e-2 ± 2.436e-2	2.229e-2 ± 1.768e-2	9.099e-3 ± 1.185e-2
	Avg. tree size	20.533	20.333	23.3	23.933	21
V	Best residual	6.076e-3	1.069e-2	3.569e-2	5.142e-4	1.299e-2
	Avg. best	2.057e-1 ± 6.818e-2	2.143e-1 ± 6.151e-2	4.232e-1 ± 8.897e-2	3.404e-1 ± 9.878e-2	2.491e-1 ± 7.354e-2
	Avg. tree size	27.333	27.2	28.033	28.867	28.1

Table 2c. Five different DE schemes.



(a) Polynomial: generation-based

(b) V-shaped: generation-based

(c) Polynomial: population-based

(d) V-shaped: population-based

(e) Polynomial: DE variations

(f) V-shaped: DE variations

Figure 2. Fitness curves for three experiments on two datasets. X-axis represents the number of GEP evaluations, and y-axis is the average of the best residual error. Log scale is used on y-axis.

- The population-based alternative produces more concise GEP formulas, as the average tree size is slightly smaller. However, in terms of *average best residual*, the generation-based approach perform better.

Figure 2(a) and 2(b) are the fitness curves for the generation-based experiments on the two problems respectively, and Fig. 2(c) and 2(d) are for the population-based experiments. The curves in the two figures illustrate how the average residual error decreases when the number of GEP evaluations increases. These curves agree with the patterns presented in Table 2a and 2b.

Both the population-based and the generation-based experiments decrease the computational cost while still having decent performance. For the polynomial function, both experiments reduce the execution of one run from hours to within an hour. For the V-shaped function, both experiments drastically decrease the run time from days to hours.

Table 2c shows the results of the experiments with DE variations. In terms of the *average of best residuals*, schemes 1, 2 and 5 outperform 3 and 4 on both datasets. It is especially obvious on the polynomial dataset. As for *average tree size*, schemes 1 and 2 produce slightly smaller trees. Figure 2(e) and 2(f) are the fitness curves corresponding to the two datasets. Towards the end in both figures, from bottom up, the curves for schemes 3, 4, and 5 are higher than those for 1 and 2. The figures confirm the observations based on Table 2c. Although schemes 1 and 2 work better than others in our experiments, surprisingly, they both have simpler formula and involve less vectors when computing the mutant vector. From our experiments, it is not necessary that more sophisticated schemes will perform better than less complex ones.

4.1 A Closer Look at Constants

For each problem, we pick a good and a bad example formula to take a closer look at the constants tuned up by DE.

The polynomial regression problem is relatively simple and most of its GEP formulas obtained in our experiments are almost perfect. A good example with residual error is $9.463e-5$ is shown in (9), which is equivalent to (10). The constants are almost the same as those in the target function (6). (11) is a relatively worse GEP formula of which the residual error is 0.1309. Its equivalence is (12). The constants in (12) are still very close to those in the target function, although less precise than those in (10). Comparing (9) and (11), the differences in their structures are obvious, but both of them can be converted into exactly the same structure of the target function.

$$y = (x * (-0.399962 - ((-1 * x) * (x - 0.300001)))) - 0.599972 \quad (9)$$

$$y = x^3 - 0.300001x^2 - 0.399962x - 0.599972 \quad (10)$$

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

$$y = x^3 - 0.300017x^2 - 0.400092x - 0.59878 \quad (12)$$

The V-shaped function is much more complicated, and improvement on this dataset was far from perfect. A GEP formula with residual error $5.901e-6$ is shown in (13), which is equivalent to (14). This is a very close approximation of (7). Equation (15) is a worse example, whose corresponding residual error is 0.5360. Notice the structure of (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).

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

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

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

The above analysis shows that when GEP finds a structure close to that of the target function, DE oftentimes can find proper constants. While GEP fails to find an appropriate formula structure, the optimization power of DE is limited, even though it might still find the best constants for the given bad structure. Therefore, GEP and DE are dependent on each other to find a good approximation. A good GEP formula structure 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

We have succeeded in enhancing the performance of GEP, by embedding DE into GEP, to optimize constants in GEP chromosomes. Our approach is novel in that real number constants are introduced into GEP formula and the choices of constants are no longer limited to pre-specified random or prime numbers. By finding appropriate constants for chromosomes, the performance of GEP is improved. In addition, constants need not be represented by combination of pre-selected constants via arithmetic operators anymore. That in turn simplifies the expression tree structure. DE helps improving GEP performance even when it is not turned on at every generation, or to the entire population, and the running time is significantly reduced. We also experimented with five different DE schemes, and found their performances comparable.

To further decrease the computational cost of the algorithm, we plan to parallelize the evaluation of GEP population and the constant optimization of each chromosome. We

also plan to try different selection methods such as tournament selection, as roulette-wheel selection requires fitness comparison between all individuals, which is time consuming. Another work would be modifying our system for classification tasks. For multi-class classification problems, it is again an open question whether multiple binary classifiers/classification rules should be trained combined, or a single multi-class classifier is practical.

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*, 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. 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Q. Zhang, C. Zhou, W. Xiao, P. C. Nelson, and X. Li. Using differential evolution for constant creation in gep. In *Late Breaking Papers at the 2006 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, July 2006.

[14] 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.