

# A New Method for Finding Constant Terms in the Context of Gene Expression Programming\*

Yanchao Liu<sup>1</sup>, Liang Gao<sup>1</sup>, Yan Dong<sup>2</sup>, Baolin Pan<sup>1</sup>

<sup>1</sup>Department of Industrial & Manufacturing System Engineering,  
Huazhong University of Science & Technology,  
430074, Wuhan, China  
gaoliang@mail.hust.edu.cn

<sup>2</sup>Department of Electronics & Information Engineering,  
Huazhong University of Science & Technology,  
430074, Wuhan, China

**Abstract.** Most of the potential function expressions subjected to function finding contain constant terms. Gene Expression Programming (GEP)'s effectiveness of function finding has been severely limited due to a lack of constant creating ability. Though this issue has been realized and discussed by many GEP researchers, an efficient method has never been established. This paper presents a new method to find functions with constant terms in the context of GEP. Experiments show its performance is outstanding: for simple integer constant terms, it surpasses original GEP up to 1-2 orders of magnitude in terms of efficiency, and for fractional constant terms, it provides an exclusive solution which no existing methods could ever achieve.

## 1 Introduction

First introduced by Candida Ferreira in 2001 [1] and developed subsequently by researchers around the world, Gene Expression Programming (GEP) has proven to be an efficient technique in the realm of data mining, especially in the field of function discovery, sequence induction and time series prediction.

GEP utilizes a genotype/phenotype representation system, which inherits both the evolutionary simplicity in Genetic Algorithm (GA) and the expressional power in Genetic Programming. Furthermore, the mapping mechanism between GEP genotype and phenotype ensures an unconstrained evolutionary environment in which all emergent chromosomes can be expressed into structurally valid individuals, in other words, potential solutions to the problem. Separation of genotype and phenotype endows GEP with the flexibility and power to explore the entire search space, thus the ability to discover complex-structured functions. However, as a data mining technique, GEP still lacks satisfactory capability to deal with numerical constants embedded in target functions both in the form of constant terms and in the form of constant coefficients.

Ferreira proposed in her first paper on GEP [1] a constant creation method by introducing an extra symbol in the terminal set to represent a constant that is picked out from a set of constants randomly created beforehand. In her later work, however, through experiments Ferreira disconfirmed the necessity and efficiency of that method by drawing the conclusion that evolutionary algorithms perform considerably worse if numerical constants are explicitly used [2]. This conclusion makes sense, since both the values and positions of the possibly-existing constants are unknown, it is intuitively impossible to eliminate the two uncertainties in one feasible algorithm. Other papers about constant creation in GEP, such as [3], which are based on Ferreira's idea of building a constant pool, will not be discussed here.

In this paper, we propose a new method to discover functions with constant terms in the context of GEP. In Section 2, we first take a brief overview to GEP's basic concept and terminology in terms of its function finding application. In Section 3, we present our simple and effective new method and in Section 4 we conduct experiments and compare the differences in performance between the original GEP and our new method to show the superiority of our method in finding constant terms. Section 5 presents some conclusion and ideas for future work.

---

\* This paper is supported by the National Basic Research Program of China (973 Program), No.2004CB719405 and the National Natural Science Foundation of China, No. 50305008.

## 2 Brief Overview of Gene Expression Programming

When using GEP to solve a problem, generally five components need to be specified. They are: the function set, terminal set (including problem-specific variable names and pre-selected constants), fitness function, GEP control parameters, and stop condition. Some details are given below:

### 2.1 The GEP Chromosomes, Expression Trees and the Mapping Mechanism

Each chromosome in GEP is a character string of fixed-length, which is composed of the element from the function set and the terminal set. For example, given the function set  $\{+, -, *, /, \text{sqrt}\}$  and the terminal set  $\{a, b\}$ , a chromosome of length thirteen is as follows:

$$*.-a./.*.\text{sqrt}.a.b.a.a.b.a.b \tag{1}$$

where “.” is used to separate individual building elements; sqrt denotes the square-root function; and a, b are variable names. The above is referred to as Karva notation, or K-expression[4].

A K-expression can be mapped into an expression tree (ET) following a width-first procedure. A branch of the ET stops growing when the last node in this branch is a terminal. For example, the ET shown in Fig. 1 corresponds to the sample chromosome (1), and can be interpreted in a mathematical form as (2).

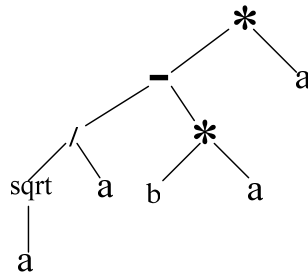


Fig. 1. Example of GEP Expression Tree

$$\left(\frac{\sqrt{a}}{a} - b \times a\right) \times a \tag{2}$$

### 2.2 The Description of the GEP Algorithm

The GEP algorithm begins with the random generation of linear fixed-length chromosomes for individuals of the initial population. The chromosomes are represented as ETs, and the fitness of each individual is evaluated based on a pre-defined fitness functions. The individuals are then selected by fitness to reproduce with modification. The individuals of this new generation are, in their run, subject to the same developmental process, i.e., expression as chromosomes, confrontation in the selection environment, and reproduction with modification. This process is repeated for a pre-specified number of generations or until a solution has been found. In GEP, individuals are often selected and copied into the next generation based on their fitness, as determined by roulette-wheel sampling with elitism [5], which guarantees the survival and cloning of the best individual to the next generation. Variation in the population is introduced by applying one or more genetic operators to selected chromosomes, including:

Crossover, two parent chromosomes are randomly chosen and paired to exchange some elements between them. There are two kinds of crossover: one-point and two-point crossover, working in the same fashion as in the canonical GAs [6].

Mutation, an element symbol in a chromosome is randomly changed to another.

Insertion, a random piece of element sequence in a chromosome is picked out and inserted in this very chromosome at a random place.

Note that all of these operations upon the coding sequence of a chromosome usually drastically reshape the corresponding ET.

### 3 A New Method for Finding Constant Terms

Constant Term universally exists in all kinds of functions, such as term  $-1.234$  in the multivariate function expression  $xy + x - y - 1.234$ , and term  $+55$  in the univariate function expression  $3x^3 + 2x^2 + x + 55$ , which is commonly seen in sequence induction problems. An effective method to discover functions with constant terms in the context of GEP has never been proposed in the existing literature. Here we present a rather simple yet highly effective procedure to handle this problem, thus enabling GEP to discover functions with constant terms.

Directly derived from GA and GP, fitness evaluators in original GEP are designed to measure the closeness between the calculated value (value returned by the potential function) and the target value (value given in the data set), the closer the fitter. This kind of fitness functions are especially represented by the ones Ferreira

proposed in her first work [1] on GEP, as  $f_i = \sum_{j=1}^{C_t} (M - |C_{(i,j)} - T_{(j)}|)$  or  $f_i = \sum_{j=1}^{C_t} (M - \left| \frac{C_{(i,j)} - T_{(j)}}{T_{(j)}} \cdot 100 \right|)$ , where

$f_i$  is the fitness of an individual program  $i$ ,  $M$  is the range of selection,  $C_{(i,j)}$  the value returned by the individual chromosome  $i$  for fitness case  $j$  (out of  $C_t$  fitness cases), and  $T_{(j)}$  is the target value for fitness case  $j$ . In this case, a perfect fit would be  $C_{(i,j)} = T_{(j)}$  and  $f_i = f_{max} = C_t \cdot M$ .

Note that such kind of fitness functions permit little room for a constant term to survive, as it pursues an absolute equality between the calculated value  $C_{(i,j)}$  and the target value  $T_{(j)}$ , while the potential function found by GEP is impossible to include a constant term in itself but the target function may well include one.

To solve this problem, we redesigned the fitness function as follows:

$$f_i = \sqrt{\frac{\sum_{j=1}^{C_t} (d_j - \bar{d})^2}{C_t}} \quad (3)$$

$$\text{where } d_j = T_{(j)} - C_{(i,j)} \text{ and } \bar{d} = \frac{\sum_{j=1}^{C_t} (T_{(j)} - C_{(i,j)})}{C_t}.$$

In the formulas above,  $f_i$  denotes the fitness of an individual chromosome  $i$ ,  $C_{(i,j)}$  the value returned by the individual chromosome  $i$  for fitness case  $j$  (out of  $C_t$  fitness cases), and  $T_{(j)}$  the target value for fitness case  $j$ .

Note that  $d_j$  is actually the distance between the target value and the calculated value for fitness case  $j$ , and  $\bar{d}$  is the average of those distances for all  $C_t$  fitness cases, and  $f_i$  the standard deviation of those distances. Note also that  $\bar{d}$  is just the constant term in the function. And the smaller the fitness value, the fitter the individual; and a perfect fit would be  $f_i = 0$ .

Apart from the ability to discover constant terms, this fitness evaluation system also reveals an inherent noise tolerance that handles inaccurate data in a special way, as will be discussed in Section 4.

### 4 Experiments

In order to justify the advantage of our fitness evaluation system as compared to the original one in constant term discovery ability, we experiment on the two systems with three artificially structured function finding problems, the first one with integer constant term, the second with fractional constant term and the third one without a constant term.

The functions used in this experiment are shown in (4), (5) and (6). Since original GEP possesses its own way of representing simple constant terms, such as representing 1 by  $(x/x)$  and 2 by  $(x+x)/x$  and 0.5 by  $x/(x+x)$ , etc., (4) is especially designed to compare this primitive ability with our new way in term of effi-

ciency. And (5) is mainly for demonstrating the exclusive power of our new method in finding fractional constant terms that no existing methods ever possess. At last, the experiment on (6) is also important here as on one hand, by contrast with the experiment on (4), it shows the clumsy nature of the original GEP's primitive ability to find simple constants, and on the other hand, it indicates a special trait of our new fitness evaluation system.

$$z = xy + x^2 - y^2 + 1 \quad (4)$$

$$z = xy + x^2 - y^2 + 1.234 \quad (5)$$

$$z = xy + x^2 - y^2 \quad (6)$$

For the fairness of the comparison, we implement the common parts (all parts except the fitness function) of the two methods (original GEP's primitive ability to find constants and our new evaluation system) using the same C++ code and run the respective programs on the same computer with the configuration as: Pentium 2.66GHz CPU, 512MB memory and Windows XP professional sp2. We adopt uni-gene chromosomes and the evolutionary environment is set as follows:

**Table 1.** Evolutionary configurations for the experiments

Mutation Rate	0.12
One-point Crossover Rate	0.5
Insertion Rate	0.3
Head Length	10
Number of Chromosomes	30
Operators	+ - * /

We assume that when the fitness function achieves a value less than 0.0001 (though the fitness values in the two methods take quite different meanings), the optimal solution is achieved, in other words, the target expression is found. In the experiment for finding expression (4), we run each method 10 times, and compare their performances by listing the number of generations evolved and the computing time of each run, as shown in Table 2. It is obvious that our new method evolves far fewer generations before the target expression  $z = xy + x^2 - y^2 + 1$  is found than the primitive method of GEP. In terms of the number of generations evolved, we can say the new method's ability to discover simple integer constant terms surpasses that of the original GEP in roughly 1 to 2 orders of magnitudes.

As for expression (5), the original GEP has no way to deal with, and our new method solves it as easily as ever, the expression (after reduction)  $z = xy + x^2 - y^2 + 1.234$  found, and the performance highly satisfactory, as shown in Table 3.

In the experiment on (6), we set the threshold fitness value to 0.001. Each run the original GEP find the expression exactly as it is with a fitness value of 0.000067, but the new method's results keep  $z = xy + x^2 - y^2 + 0.000012$  (after reduction) with a fitness value of 0.000125. This is comprehensible. Since the data set more or less suffers round-off error, we regard that the inaccuracy in the result is natural, as the new method is designed to seek minimum of the standard deviation of the residuals instead of the average of the residuals. In fact, this reveals an inborn ability of the method to deal with noisy data, which is very useful in mining real world experimental data. The comparison of the two methods' performances is shown in Table 4.

Note from the comparison between Table 2 and Table 4 that for the original GEP, finding an expression with simple integer constant term is no easy task, much more difficult than finding the same expression without constant term. This is a telling demonstration of the inefficient nature of GEP's primitive way of constructing constants. In contrast, the existence of a constant term and the type of the constant term (integer or fractional) have no effect on the performance of our new method.

**Table 2.** Comparison of performance parameters at optimal points for finding (4)

Run	Primitive method		New method	
	Generations	CPU time (s)	Generations	CPU time (s)
1	27938	26.547	1062	1.172
2	53146	49.859	2184	2.156
3	10169	9.609	599	0.547
4	73352	69.875	289	0.282
5	119701	113.015	132	0.140
6	3649	3.375	1012	1.000
7	10837	10.141	243	0.234
8	83760	79.203	231	0.219
9	88000	82.968	212	0.204
10	7972	7.437	1122	1.094

**Table 3.** Performance parameters at optimal points for finding (5) using the new method

Run	1	2	3	4	5	6	7	8	9	10
Generations	43	71	271	710	759	388	167	284	54	294
CPU time	0.031	0.078	0.266	0.687	0.735	0.359	0.157	0.282	0.063	0.297

**Table 4.** Comparison of performance parameters at optimal points for finding (6)

Run	Primitive method		New method	
	Generations	CPU time (s)	Generations	CPU time (s)
1	816	0.844	446	0.437
2	1470	1.500	10	0.015
3	1458	1.531	15	0.031
4	234	0.234	1010	0.969
5	181	0.203	552	0.500
6	2421	2.515	342	0.344
7	2044	2.125	6	0.016
8	767	0.781	361	0.375
9	351	0.391	727	0.687
10	205	0.203	53	0.062

## 5 Conclusions and Future Work

This paper has presented a new fitness evaluation system for GEP, thus provides a method to discover functions with constant terms. The positive experimental results prove it a very effective and efficient method. Our work has also opened a new perspective in the development of GEP and future research in this direction will mainly focus on: 1. through the combination of GEP and other data mining techniques, find a way to discover function expressions with constant coefficients; 2. investigate more into the mining of noisy data in the context of GEP; 3. through fitness functions redesign, explore new applications of GEP.

## References

1. Ferreira, C.: Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. Vol.13. Complex Systems (2001) 87-129
2. Ferreira, C.: Function Finding and the Creation of Numerical Constants in Gene Expression Programming. In: Proceedings of the 7th Online World Conference on Soft Computing in Industrial Applications (2002)

3. Xin L., Chi Z., Peter C.: Nelson, and Thomas M. Tirpak, Investigation of Constant Creation Techniques in the Context of Gene Expression Programming. In: M. Keijzer, ed., Late Breaking Paper at Genetic and Evolutionary Computation Conference, GECCO-2004, Washington, USA (2004)
4. Ferreira, C.: Gene Expression Programming and the Evolution of Computer Programs. In: Leandro N. de Castro and Fernando J. Von Zuben, eds., Recent Developments in Biologically Inspired Computing, Idea Group Publishing (2004) 82-103
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Pub (1989)
6. Mitchell, M.: An Introduction to Genetic Algorithms (Complex Adaptive Systems). MIT Press (1996)