Improving Gene Expression Programming

Dr. Najla Akram AL-Saati **Dr. Nidhal Al-Assady**
Software Engineering Dept.
College of Computers and Mathematical Sciences

ABSTRACT

In this work the algorithm of Gene Expression Programming (GEP) is investigated thoroughly and the major deficiencies are pointed out. Five suggestions for enhancements are introduced in this research aiming at solving the major deficiencies that were investigated. These improvements produced higher success rates and avoid the malfunctioning situations found in GEP. These deficiencies or weak points include: choosing the best parameter settings, using different linking functions, gene flattening problem, and illegal operations in genes. Tests were carried out using two symbolic regression problems.

الخلاصة

يتناول هذا البحث اجراء دراسة حول خوارزمية (البرمجة باستخدام التعبير الجيني الوراثي) (Gene Expression Programming) بشكل شامل وموسع حيث تم ابراز اهم المشاكل التي تعاني منها هذه الطريقة. وقد تم في هذا البحث تقديم خمسة مقترحات لتحسين وحل تلك المشاكل حيث تقوم هذه التحسينات بانتاج نسب نجاح عالية وتتلافى الحالات غير الصحيحة المكتشفة في طريقة (GEP) . تتضمن هذه المشاكل او نقاط الضعف: اختيار افضل ترتيب للمعاملات ، استخدام دوال ربط مختلفة ، مشكلة تسطح الجين ، والعمليات غير القانونية في الجينات. تم اجراء الاختبارات باستخدام مسألتين مختلفتين من الانحدار المرمز (symbolic regression).

١ Introduction:

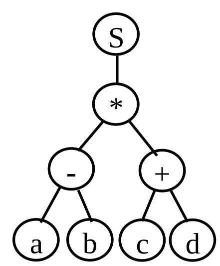
Gene expression programming (GEP) was introduced by Ferreira in 2001 [3]. The great insight of GEP consisted in the invention of chromosomes capable of representing any expression tree. For that a new language (Karva) was created so that the information of GEP chromosomes could be read and expressed. The structural and functional organization of genes always guarantees the production of valid programs, no matter how much or how profoundly the chromosomes are modified.

The phenotype of GEP individuals consists of the same kind of diagram representations used by GP. However, these complex entities are encoded in simpler, linear structures of fixed length (chromosomes). Thus, the main players in GEP are two entities: the chromosomes and the ramified structures or expression trees (ETs), being the latter the expression of the genetic information encoded in the former.

The process of translating the chromosomes to expression trees (ETs) implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship between the symbols and the functions or the terminals they represent. The rules are also very simple: they determine the spatial organization of the functions and terminals in the ETs and the type of interaction between sub-ETs in multigenic systems [4]. Given a GEP individual (genotype) expressed in Karva language, the phenotype can easily be represented by an ET as shown in Figure (1):

```
01234567
S*-+abcd
```

$((a-b)*(c+d))^2$ →



Note :'S' is the square function

Figure (1) Representation of GEP Chromosome

Genes are composed of a head and a tail. The head contains both function (non-terminal) and terminals symbols. The tail contains only terminal symbols. For each problem the head length (h) is chosen by the user. Given the maximum arity n, or the number of arguments for the function with the most arguments, the tail length (t) is evaluated by:

$$t = (n - 1)\, h + 1 \quad \ldots\ldots (1)$$

In this way if n=2 and h= 4, then t=5 and the total length of the gene is 9. So despite their fixed length, GEP genes have the potential to code for ETs of different sizes and shapes, being the simplest composed of only one node (the first element is a terminal) and the biggest composed of as many nodes as the gene length (all head elements are functions of maximum arity).

GEP chromosomes are usually composed of more than one gene of equal length; as in Figure (2). For each problem or run, the number of genes, as well as head length, is a priori chosen. Each gene codes for a sub-ET that interact with one another forming a more complex multi-subunit ET.

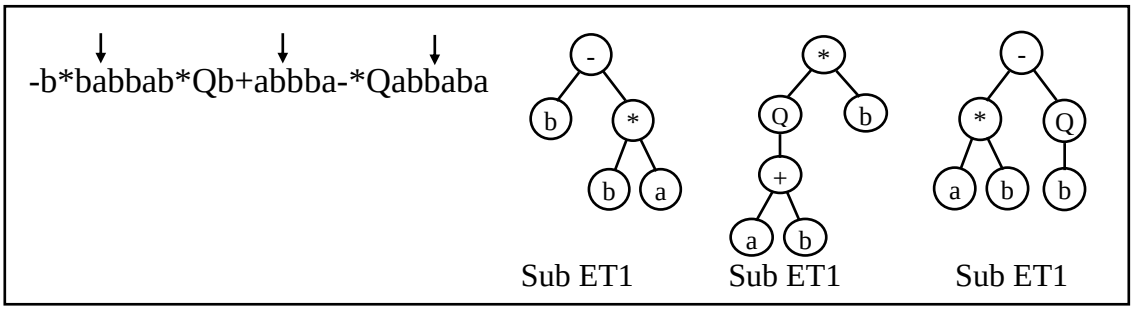


Figure (2) Multigenic Chromosomal Structure in GEP Method

Multigenic chromosome was introduced because it can happen that the first symbol in a gene to be a terminal, and thus a single gene chromosome cannot represent a complex expression. As an indirect consequence, if the first symbol of a gene is a terminal then the rest of the gene is unused.

Breadth-first parsing is used in the translation of tree programs into genes, where usually the gene is not entirely used for phenotypic transcription. If the first symbol in the gene is a terminal, the expression tree consists of a single node. If all symbols in the head are non-terminals the expression tree uses all the symbols of the gene. Genes may be linked by a function symbol in order to obtain a fully functional chromosome. The linking functions for algebraic expressions are addition and multiplication. A single type of function is used for linking multiple genes. If the functions { + , - , * , / } are used as linking operators then the complexity of the problem grows substantially (since the problem of determining how to mix these operators with the genes is as hard as the initial problem).[10]

2 GEP malfunctioning conditions:

GEP method was thoroughly investigated in this work, due to the fact that it is considered to be the most appropriate approach among the various AP methods introduced so far. Carrying out such an investigation has led to the discovery of five main issues that were used to improve the performance of GEP. These are described in the following sections.

2.1 The Choice of the Best Environmental Parameter Settings

This is a problem shared among all EAs; it is the decision of the right parameter setting for an algorithm, which produces the best results possible. When defining an EA there is a great need to choose its components, such as genetic operators, selection

mechanisms for selecting parents, and initial populations. Each of these may have parameters, like: mutation probability, or population size. Values of these parameters greatly determine whether the algorithm will find a near-optimum solution and whether it will find one efficiently. Choosing the right parameters, however, is time-consuming and considerable effort has gone into developing good heuristics for it.[2]

Early attempts put considerable efforts into finding parameter values, which were good for a number of numeric test problems (experimentally determined). Later, meta-algorithms were used to optimize values of these parameters. Eiben, et. al [2], globally distinguished two major forms of setting parameter values: *parameter tuning* (the common approach that amounts to find good values for parameters before the run and then run the algorithm using them) and *parameter control* (remains fixed during the run). They also give arguments that any static set of parameters, having the values fixed during a run, seems to be inappropriate. Whereas Parameter control forms an alternative, it amounts to starting a run with initial parameter values that are changed during the run.

2.2 The Use of Different Linking Functions

Given a set of functions to be used in evolution, one function should be used to link existing genes. This choice varies depending on the function set, the types of functions included in the sets, and the rules to be evolved.

Using one of the linking functions through the entire evolution process is not appropriate nor of any advantage to the system. Attempting to use varied linking functions in one population will only cause the complexity of the problem to grow substantially, while the problem of determining how to mix these operators with the genes is as hard as the initial problem.[10]

2.3 The Problem of Gene Flattening

Another fact noticed about GEP, is gene flattening in chromosomes. Flat genes are genes with heads containing only terminal symbols; they may appear as a product of applying IS insertion of the transposition operator coupled with mutations changing functions to terminals. This problem appears when there is no guarantee for forbidding operators from destructing the functionality of the gene by eliminating functions from the head.

Restricting the operator from inserting the chosen sequence at the beginning of the head is not enough. In the worst case, the first symbol in the existing head might just be a terminal leading to the destruction of any hope in saving the gene though other operators. Repeated occurrence of this event can increase the rate of flat genes in the chromosome. Even when the first symbol in the head is not a terminal, such a process can reduce the efficiency of the gene by increasing terminals in heads, thus producing poorly functioning genes that weaken chromosomes in the population.

2.4 The Problem of Illegal Operations in Genes

Through the process of evaluating a gene, it is very likely to encounter terminals or operands to functions that, when evaluated, gives illegal results like division by zero or square root of negative values. This usually leads to the termination of the evaluation process, and thus excluding the contribution presented by the gene, and the whole chromosome is assigned the worst fitness measure agreed upon. This will certainly cause the loss of significant chances for introducing fit individuals in the population. Chromosomes are assigned poor fitness values due to the existence of illegal operands to functions in only one of its genes; other genes may have valuable fitness measures to offer.

2.5 Improving GEP using Biased Components

Some EC algorithms try to increase efficiency and performance of the evolutionary process by giving a higher rate of occurrence to some elements from the function or terminal set that makes up the contents of genes in the chromosome. This feature was employed in Multi Expression Programming.

In such a procedure, certain components, like addition or multiplication operators, are usually assigned a higher chance of being introduced in the genes of the chromosome than other operators. The idea is about focusing on the terms that are more vital in the construction of a rule, and thus allowing evolution to adapt more rapidly towards forming desired rules or programs.

3 Suggested Solutions

In an attempt to improve the performance of GEP, new characteristics are introduced, the *Multi-population* feature, which is used to ensure better exploitation of the properties possessed by the method. This feature is completely inspired by nature, as many natural environments are found to adopt multi populations as ecosystems that evolve simultaneously and concurrently under some certain resources or environmental circumstances. Some of these situations are shared and are common between such evolving ecosystems, while others are locally exclusive or restricted as they vary from one population to another. This decisiveness usually depends on environmental needs demanded by each individual population, another important issue to rely on when choosing to localize or globalize an aspect relevant to a population, is the overall performance of the resulting system.

Introducing this new feature involves decomposing existing large population into a number of smaller distinct entities each having its own set of parameters, thus forming several diverse environments that evolve independently and simultaneously. In GEP there are some certain settings that must be globally maintained to all populations, while others need to be locally differentiated to overcome certain malfunctioning phenomena. Useful issues that can be viewed using this feature are:

1- **Introducing various environments to enhance evolution**: this is done by dividing the impact of large populations with the same evolutionary features, and use small multiple ones with various environmental features.
2- **Finding parameter sets**: helps to find the appropriate set of parameters applied to a system, instead of trying to find them by hand tuning.
3- **Evaluating Genetic Dynamics**: varying operator's probabilities in a multi-population collection while fixing others and making them global to the whole environment. This is very useful in the study of dynamics.
4- **Evaluating Environmental Settings**: population size, number of generations, chromosomal length and number of genes can each be evaluated using multi-population collections. This enables the study of the impact that these settings have on the behavior of the system.

This feature is used in the following section to improve first and second problems. As for the third and fourth, a monitoring process is added to detect the occurrence of flat genes or illegal operations in the population and are avoided using *emergency mutations*. Considering the idea of component biased assigning, GEP can be improved by giving more weight to one or more solution components. The choice of biasing a certain component among the set is done depending on the type of rule or program to be evolved.

4 Symbolic Regression

4.1 Problem Description

The symbolic regression problem can be stated as finding a function in a symbolic form that fits a given finite sample of data [7]. The advantage of symbolic regression over standard regression methods is that in symbolic regression, the search process works simultaneously on both the model specification problem and the problem of fitting coefficients. Symbolic regression would thus appear to be a particularly valuable tool for the analysis of experimental data where the specification of the strategic function used is often difficult, and may even vary over time.[1]

The system is given a set of input and output pairs, and must determine the function that maps one onto the other. Symbolic regression tries to reconstruct a mathematical function just using a set of data samples. This data can be pairs of independent and dependent variables that are samples of a possibly unknown function.

As an aspect of Data Mining, symbolic regression is inherently computationally extensive because of the lack of a model solution in general.[11] The problem, in its essence, is an optimization problem; a search is conducted for the most fitting individual to the data, in the space of all possible expressions. In his work, Freitas [6]

showed how the requirements of data mining and knowledge discovery influence the design of EAs. In particular, how individual representation, operators and fitness functions have to be adapted for extracting high-level knowledge from data. Data mining is more or less the same as symbolic regression but the emphasis is not on complete description of the data but on extracting salient nuggets of information from potentially large data sources (e.g. databases).[9] GP possesses certain advantages that make it suitable for application in data mining, such as convenient structure for rule generation. Furthermore, it is convenient for process parallelism to improve computational efficiency.[8]

The object of the search is a symbolic description of a model, not just a set of coefficients in a pre-specified model. This sharply contrast with other methods of regression, including feed-forward ANN, where a specific model is assumed and often only the complexity of this model can be varied.[12]

Genetic programming and its variants are in principle capable of expressing functional forms, given a sufficiently expressive function set; they are capable of expressing a linear relationship or a non-linear relationship. With Genetic programming and variants, the object of search is a composition of the input variables, coefficients and primitive functions such that the error of the function with respect to the desired output is minimized.

4.2 Fitness Measure

One important application of GEP is symbolic regression, where the goal is to find an expression that performs well for all fitness cases within a certain error of the correct value. Mathematically, this can be expressed by the equation:

$$f = M - E, \quad \ldots\ldots(2)$$

where M is the range of selection, and E is the absolute error between the number generated by the ET and the target value, as follows:

$$E = |C_{(i,j)} - T_j|, \quad \ldots\ldots(3)$$

where $C_{(i,j)}$ is the value returned by the individual chromosome *i* for fitness case *j* and T_j is the target value for fitness case *j* (for all *j* of the fitness cases). The precision for the absolute error is usually very small, for instance 0.01. For example, for a set of 10 fitness cases and an M = 100, $f\ max$ = 1000 if all the values are within 0.01 of the correct value, as follows:

$$f_i = f_{max} = C_t * M, \quad \ldots\ldots(4)$$

where C_t is the number of total fitness cases. If, for all j, $|C_{(i,j)} - T_j|$, (the precision) less or equal to 0.01, then the precision is equal to zero. So, the fitness measure f_i of an individual program i is given by:

$$f_i = \sum_{j=1}^{C_i} \left(M - \left| C_{(i,j)} - T_j \right| \right) \quad \text{.....(5)}$$

The advantage of this kind of fitness function is that the system can find the optimal solution for itself. [5]

5 Tests and Results

Experiments carried out in this section are implemented using the Symbolic Regression problem. Due to its simplicity and common use in most of the applications, it has almost become a benchmark problem in assessing AP systems. As a standard benchmark problem it is very useful in making comparisons more practicable. Each test applies 100 run of randomly generated populations to evaluate success rates of the approach. In the following tests two equations are used to determine the efficiency of the improvements carried out, they are as indicated in the tables of comparisons:

$Y = a^4 + a^3 + a^2 + a$ .....(6)

$Y=3a^2+2a+1$ .....(7)

Fitness cases (Training set) are chosen as those used by all AP methods proposed so far, this is done to facilitate comparisons. Training cases are given in Tables (1) and (2), parameter settings are given in Table (3).

Table (1) Fitness Cases for First Problem

In	Out
2.81	95.2425
6	1554
7.043	2866.5485
8	4680
10	11110
11.38	18386.0340
12	22620
14	41370
15	54240
20	168420

Table (2) Fitness Cases for Second Problem

In	Out
-4.2605	46.9346
-2.0437	9.4427
-9.8317	271.3236
2.7429	29.0563
0.7328	4.0766
-8.6491	208.1226
-3.6101	32.8783
-1.8999	8.0291
-4.8852	62.8251
7.3998	180.0707

Table (3) Parameter Settings for Tests

Setting	**GEP**
Number of Runs	100
Generation	50
Population	30
Chromosome Length	39
Genes	3 (**h=6**)
Function Set	{+,-,*,/}
Terminal Set	{a}

5.1 Improvement Related to Parameter Setting

Applying Multi-population feature enables the system to use different settings for each population and can therefore reduce the parameter-setting problem discussed in the first subsection. Having *P* Populations each of size *S* with *G* as the number of Generations, the test is done using 3 populations, with settings in Table (4). Results are shown in Table (5).

Table (4) Multi-Population system with Different Parameter Setting

					Transposition			**Recombination**		
	P	**S**	**G**	**Mutation**	**IS**	**RIS**	**GIS**	**One**	**Two**	**Gene**
Improved	1	7	50	0.05	0.1	0.1	0.1	0.2	0.5	0.1
	2	10	-	0.03	0.15	0.15	0.1	0.1	0.7	0.15
	3	13	-	0.1	0.15	0.1	0.15	0.3	0.5	0.1
GEP	1	30	50	0.05	0.1	0.1	0.1	0.2	0.5	0.1

Table (5) Results of applying Multi-population

Evolved Function	**GEP results**	**Improved Results**
$Y = a^4 + a^3 + a^2 + a$	0.81	0.91
$Y = 3a^2 + 2a + 1$	0.83	0.92

5.2 Improvement Related to Linking Function

This is another case that can make use of the multi-population feature in investigating the affect that linking functions have on fitness calculations. *First*, different populations were introduced each having its own local linking function, results showed that the '*', '-', and '/' function were not able to enhance the rate of successful runs, the rate went down for all functions except the '+' function.

Second, different linking functions were applied to link genes. Having 3 genes, the proposal suggests linking first and second genes with one linking function, while linking the result with the third gene by another one. Results showed that this was also not helpful in increasing success rates.

Gained results point out a very normal consequence, as the type of rules evolved in the tests relies heavily on addition; any other linking function will not be appropriate in this case. The function to be evolved is a summation process of multiple terms. It is very clear that the use of the Multi-population feature enabled the study of applying various linking functions to the system, and was able to distinguish the best population that gave best results.

5.3 Improvement Related to Flat Genes

Flat genes are avoided by imposing some monitoring process on the application of the IS operator, so that, when the number of functions in the head is zero, an emergency mutation is forced after that IS operation to ensure the existence of a function in the head of that modified gene. Results are shown in Table (6).

Table (6) Results of Adjusting IS Operator for Eliminating Flat Genes

Evolved Function	GEP results	Improved Results
$Y = a^4 + a^3 + a^2 + a$	0.81	0.90
$Y = 3a^2 + 2a + 1$	0.83	0.92

5.4 Improvement Related to Illegal operations in genes

The problem of illegal operations in genes is treated by adding a very simple mechanism in fitness calculation, when an invalid operation is about to cause the termination of fitness calculation, it is simply mutated in its place to any of the other remaining functions in the function set. Using this mechanism, the gene is saved from complete loss and can be presented again in the population with an appropriate fitness value. The result of applying this idea to GEP is shown in Table (7).

Table (7) Results of Eliminating Illegal Operations

Evolved Function	GEP results	Improved Results
$Y = a^4 + a^3 + a^2 + a$	0.81	0.88
$Y = 3a^2 + 2a + 1$	0.83	0.89

5.5 Improvement Related to Biased Components

Biased GEP was tested through biasing different components and monitoring the effect of that biasing on the process of evolution and the rate of success; results are shown in Table (8).

Table (8) Results of Biased GEP Operations

Evolved Function	Biased Function	GEP Results	Improved Results
$Y = a^4 + a^3 + a^2 + a$	'+'	0.81	0.57
	'-'		0.76
	'*'		**0.90**
	'/'		0.68
$Y = 3a^2 + 2a + 1$	'+'	0.83	**0.91**
	'-'		0.74
	'*'		0.73
	'/'		0.83

For the first case in Table (8), biasing the multiplication operator influenced the rate of success considerably. This is mainly because the rule depends heavily on this function. While in the second case the biasing of the addition operator was more successful than the others, as the evolved rule depends more on addition than multiplication, subtraction or division.

6 Conclusions:

Many linear variants of Genetic programming are presented in the literature, of these; GEP was investigated thoroughly as it possesses the least limitations among other methods. Like any other method, GEP has some points of weakness that reduces its efficiency. These points were investigated and reinforced with five solutions that managed weak points in an efficient manner; weak points included the choice of the best parameter settings for evolution, the use of different linking functions, the problem of gene flattening, and the illegal operations that occur in the genes of the chromosome.

The five enhancement procedures suggested were able to eliminate these problems and increase the efficiency of the method. Enhancement procedures included introducing the *Multi-population* feature, the *Emergency Mutation* feature, and the *Component Biasing* feature. Tests and results showed that success rates improved clearly towards higher values in all cases.

7 References

[1] Duffy, J., and Engle-Warnick, J., (2002), *Using Symbolic Regression to Infer Strategies from Experimental Data*. In S-H Chen, Ed., Evolutionary computation in economics and finance New-York Physica-Verlag.

[2] Eiben, A.E., Hinterding, R., and Michalewicz, Z., (1999), *Parameter Control in Evolutionary Algorithms*, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2, pp: 124-141.

[3] Ferreira, C., (2001), *Gene Expression Programming: A new Adaptive Algorithm for Solving Problems*, in Complex Systems, 13(2),pp:87-129.

[4] Ferreira, C., (2002), *Discovery of the Boolean Functions to the best Density-Classification Rules using Gene Expression programming*, in Lutton, E., Foster, J. A., Miller, J., Ryan, C., and Tettamanzi, A. G. B., Eds., in Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002, Vol. 2278 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany,pp: 51-60.

[5] Ferreira, C., (2002), *Gene Expression Programming in Problem Solving*, in Roy, R., Koppen, M., Ovaska, S., Furuhashi, T., and Huffmann, F., Eds., Soft Computing and Industry - Resent Applications, Springer-Verlag, pp: 635-654.

[6] Freitas, A.A., (2002), *A survey of evolutionary algorithms for data mining and knowledge discovery*, to appear in: Ghosh, A. and Tsutsui, S., Eds.: Advances in Evolutionary Computation, Springer-Verlag.

[7] Hoai, N.X., (2001), *Solving the Symbolic Regression with Tree-Adjunct Grammar Guided Genetic Programming: The Preliminary Results*, In The Proceedings of The 5th Australasia-Japan Joint Workshop on Evolutionary Computation and Intelligent Systems (AJWIES), Dunedin, New Zealand, 19-21st Nov. 2001,pp:1-6.

[8] Keijzer M., (2002), *Scientific Discovery using Genetic Programming*, Ph.D. thesis at the Technical University of Denmark.

[9] Langdon, W.B., (1996), *Genetic Programming and Databases*, Internal Note IN/96/4, 11 February 1996, Short survey, 3p.

[10] Oltean M., Dumitrescu D., (2002), *Multi Expression Programming*, Technical Report: UBB-01-2002, Babes-Bolyai University, Cluj-Napoca, Romania, in Journal of Genetic Programming and Evolvable Machines, Kluwer, second tour of review, 33p.

[11] Salhi, A., Glaser, H., and De Roure, D., (1998), *Parallel Implementation of a Tool for Symbolic Regression*, in Information Processing Letters, Vol. 66, No. 6, pp: 299-307.

[12] Takač, A., (2003), *Genetic Programming in Data Mining: Cellular Approach*, M.Sc. Thesis, Institute of Informatics Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia.