A Comparative Study of Linear Encoding in Genetic Programming

Yuttana Suttasupa, Suppat Rungraungsilp, Suwat Pinyopan,
Pravit Wungchusunti, Prabhas Chongstitvatana
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
suppat.r@student.chula.ac.th, and prabhas.c@chula.ac.th

***Abstract*—Genetic Programming is a widely used technique to solve many optimization problems. The original representation of a solution is a tree structure. To improve its search capability there are many proposals for encoding data structure of a solution of Genetic Programming as a linear code. However there are a few work in comparing between these proposals. This work presents a systematic way to compare three popular techniques for linear encoding in Genetic Programming. They are Linear Genetic Programming, Gene Expression Programming and Multi-Expression Programming. Ten problems in Symbolic Expressions are defined and are used as benchmarks to compare the effectiveness of these proposals against the baseline standard Genetic Programming. The metrics of comparison are the Success Rate and the absolute error. The discussion and comparison of the strength and weakness of each method are also presented.**

***Keywords* — Genetic programming, Linear Genetic Programming, Gene Expression Programming, Multi-expression Programming**

1. INTRODUCTION

Genetic programming (GP) [1] is a form of evolutionary algorithm that copy evolution of life. Genetic Programming looks like Genetic Algorithms (GA) particularly the problem solving process. Their differences are that GP searches for appropriate computer programs but GA searches for a solution. In GP, users define an appropriate form of input and output so that the correct form of computer programs will be evolved. Normally, GP stores a form of answers as tree structures that are easy to understand for human. Many works suggest linear encoding of data structure instead of trees. The claims of advantages of linear encoding are numerous. For example, it saves space [2]. The manipulation of the data structure is also faster in linear form than in the original form. In terms of search performance, it is not clear which proposals are better. The examples of these proposals are Linear Genetic Programming (LGP) [3], Gene Expression Programming (GEP) [4, 5], Multi-expression Programming (MEP) [6]. Each algorithm has different pros and cons.

In this paper, we present a comparison of three proposals namely, Linear Genetic Programming (LGP), Gene Expression Programming (GEP) and Multi-expression Programming (MEP). A standard Genetic Programming is used as a reference. A set of benchmark problems are defined using ten problems in symbolic regression. For the comparison, all parameters in the runs, such as the size of population, the number of maximum generation, the genetic operators etc. are the same for all methods. The success rates of solving the problems are recorded. They indicate the performance of each method in finding an answer. The average error in each generation is used as a measure of the speed of convergence of each method.

The structure of this paper is as follows. Section 2 discusses the encoding technique of each method and analyse its strength and weakness. Section 3 explains the numerical experiments. The conclusion and discussion are presented in Section 4.

2. INDIVIDUAL REPRESENTATION

In this section, we compare the individual representation. Each algorithm has different characteristics. GP, LGP and GEP encode a single solution in a chromosome while MEP encodes multiple solutions in a chromosome. All methods use integer and real number in encoding an individual according to problems. The presentation starts with the description of the standard Genetic Programming.

2.1 Genetic programming (GP)

Genetic Programming (GP) [4] is developed from Genetic Algorithm. The main difference between Genetic Programming and Genetic Algorithm is that Genetic Programming represents a solution as a tree instead of a binary string used in Genetic Algorithm.

The steps of Genetic Programming are as follow:

1. Create initial random population. The function set (such as +, -, x, / etc.) becomes the internal node of the tree. The terminal set (the free variables, such as x1, y1 etc.) becomes the external (leave) node.
2. Evaluate the population according to the fitness function. The fitness indicates how well each individual solves the problem.
3. Create the next generation population by evolving its structure as follow:
   - 3.1 Select the better individual in the population and copy them to the next generation using tournament selection.
   - 3.2 Create new trees by crossover method.
   - 3.3 Create new trees by mutation method.
4. Repeat 2 and 3 until the solution is found or the maximum generation is reached.

978-1-4577-2162-5/11/$26.00 ©2011

Pseudo code as follow:

gen = 0
create initial random population
while not terminate
evaluate fitness of all individual
improve selected individual by genetic operation
reproduction
crossover from two selected parents
mutation
gen = gen + 1

2.1.1 Genetic programming strengths

Genetic programming represents individual as a tree that always has a correct form so that all genetic operators generate a valid tree. Therefore there is no wasted effort in producing a ill-formed solution.

2.1.2 Genetic programming weaknesses

Genetic programming encodes a single expression in an individual (other method can represent many expressions). When the population evolve, the complexity of individual will grow, hence it will become less effective.

2.2 Linear genetic programming (LGP)

Linear genetic programming uses genetic operations on linear genomes. The linear genome represents computer programs at low-level language, machine code in some applications. Its characteristics are more similar to imperative language (such as c/c++) thus evolution is a process of unlimited possibilities. The population will be "compiled" into machine code that can be executed directly by the host computer. The basic unit of evolution is native machine code instruction that work on floating-point processor unit (FPU). An individual is composed of instruction blocks. Each block size is 32 bits. Each instruction blocks may have one or more native machine code instructions. The crossover boundary is on each instruction. The mutation operator can have random effect on the instruction by changing its parameter values.

An individual is described by a variable length sequence of simple C-like instructions. Each instruction uses one or two indexed variables (registers) r or constant c from the group of constants that is already defined. The number of registers is generally equal to the number of attributes of the problem. Typically, LPG uses modified steady state algorithm which changes the population one-by-one. The setting of various parameters for linear genetic programming is very important to make the system work properly.

2.2.1 Linear genetic programming strengths

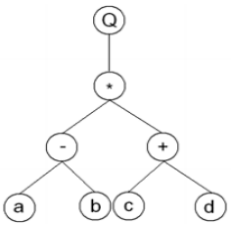
The solutions can be evolved quickly because each solution is 'compiled' into machine codes, usually native to the processor used to run the algorithm. The evaluation of the fitness for each individual is therefore very fast.

2.2.2 Linear genetic programming weaknesses

The success of the method is sensitive to parameter setting. The number of registers used in a chromosome is important in LGP. As a guide line, the number of registers is equal to the number of attributes of the problem. Except for the problem that has only one attribute, the solution requires to be in a complex expression such as quadratic. Therefore it needs to have many registers. In this case, it is necessary to use several supplementary registers to represent the complexity of the expression being discovered.

2.3 Gene expression Programming (GEP)

Gene expression Programming [5, 7] is an evolutionary algorithm that searches for the answers which is computer programs, similar to Genetic Programming. However, in terms of chromosome manipulation, it is more similar to Genetic algorithm (GA) than Genetic Programming. Each individual is stored as an encoding of a tree in linear order breath first traversal of the tree. Each chromosome also has fixed length. The genetic operators then can be borrowed from GA. However, the result of the genetic operation in GEP is different from GP. For example, the recombination (crossover) in GP is the swapping of subtrees but in GEP it is an exchange of substrings of linear code which when interpret as a tree, it is different from swapping the subtrees. The structure of GEP genes are maintained in the form of open reading frames (ORFs). In the molecular biology, an open reading frames is the coding of a cable of gene starts at the start codon (the beginning of a DNA line), followed by amino acid codons and finally concludes with a termination codon. It is clear that gene has more information than ORF. There are sequences above the start codon and sequences under the stop codon. Both sequences are parts that can not be interpreted hence they do not appear as a characteristic of genes but they are extremely important in terms of evolution.



The encoding scheme can be explained using a simple example. An algebraic expression " sqrt((a-b) * (c+d)) " shows as a tree in Fig.1.

Figure 1 The expression tree of the example. "Q" is the square root function. This expression can be encoded in a linear form as:

01234567
Q*-+abcd
Figure 2 The linear encoding of GEP

This expression tree (ET) reads from left to right and from top to bottom. The expression is an ORF which starts from the "Q" (at position 0) and ends with "d" (at position 7), also known as K-expressions.

The method of interpreting K-expressions into expression tree is not difficult. The required information are the type of functions, the number of their arguments and the type of terminals. Starting from the first symbol "Q", place it as the root node. The next symbol will be the children. If

the symbol is a function, the number of its children is known and can be read properly and placed into the right place in the tree. The order of the children is left to right. K-expression looks like a traversal of the tree in breath-first order. Given an arbitrary chromosome in GEP, only some range in the chromosome can have meaningful interpretation. Therefore, the trees representable by GEP will have variable size. The part that is not interpreted in a chromosome has an important role to protect this chromosome from the crossover and mutation operations.

The genes of Gene Expression Programming include head and tail. The head (h) may include both functions and terminals but the tail (t) only contains terminals. The maximum number of arguments in function, n, is given by equation:

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

2.3.1 Gene expression programming strengths

GEP chromosome consists of two parts (head and tail). The head contains specific function symbols. GEP is powerful for encoding syntactically correct computer programs.

2.3.2 Gene expression programming weaknesses

The success rate depends on the selection of operators such as +, -, * and / that are used to form the structure of the genes as well as the number of genes. In a chromosome, if the target expression is short and the head part is big, a large portion of chromosome will be unused. This will result in a waste computational effort in discovering a solution.

2.4 Multi-expression programming (MEP)

In Multi expression Programming (MEP)[8], a gene is represented by substring of variables. The number of gene per chromosome is constant. This number will determine the length of the chromosome. Each gene encoded the terminal and function symbols. The encoding scheme is again, illustrated by a simple example. Given a set of functions, {+, *}, and a set of terminals, {a, b, c, d}, the following expression "(a+b) * (c+d) " can be encoded as:

1: a
2: b
3: + 1, 2
4: c
5: d
6: + 4, 5
7: * 3, 6

At the position 1, 2, 4 and 5, each terminal is encoded as follow:

E1 = a,
E2 = b,
E4 = c,
E5 = d,

Gene at the position 3, 6 and 7 encoded the functions:

E3 = a + b,
E6 = c + d,
E7 = (a + b) * (c + d).

The maximum number of symbols (s) in the chromosome is calculated from the formula:

$$s = (n+1)*(p-1) + 1 \quad (2)$$

where n is number of conditions function includes the maximum number of conditions, p denotes the number of genes.

2.4.1 Multi-expression programming strengths

MEP encodes several expressions in the same string so it has multisolution representation. The multi-expression chromosome has advantages over the single-expression chromosome especially when the form of the target expression is not known. MEP can repeatedly use the same sub-expression in an expression. This method of code reuse is similar to the automatically defined functions (ADFs) in GP [2].

2.4.2 Multi-expression programming weaknesses

Since the representation in MEP is complex therefore the algorithm itself is more complex than other methods. The flexibility of MEP that it can model the data without knowing the association of its attributes comes with a higher cost in complexity when performing decoding of genes.

3. NUMERICAL EXPERIMENTS

The symbolic regression problems are used to compare the effectiveness of the several linear encoding Genetic Programming methods. Ten functions are defined:

$F\_1(x) = x^4 + x^3 + x^2 + x$
$F\_2(x) = \sin(x^4 + x^2)$
$F\_3(x) = \sin(\exp(\sin(\exp(\sin(x)))))$
$F\_4(x) = 3x^3 + 4x^2 + 2$
$F\_5(x) = \log(x^3 + x^2)$
$F\_6(x) = \tan(x)\sin(x^2)$
$F\_7(x) = \frac{1}{x^2} + \sqrt[2]{x}$
$F\_8(x) = \frac{(x^2+x)}{y^2}$
$F\_9(x) = \log(x^2 + y) + x$
$F\_10(x) = e^x + \cos(x^3)$

The parameters of the runs for all problems are similar except the function and terminal symbols set. The function symbols and the terminal symbols are chosen to be appropriated to the problem. The function symbols for each problem are chosen as follow: F_1 = {+, *, /}, F_2 = {+, *, /, sin}, F_3 = {sin, exp}, F_4 = {+, *, /} the operator {-} is not used because it is not required in the target function, F_5 = {+, *, /, log}, F_6 = {*, /, sin, tan}, F_7 = {+, *, /, sqrt}, F_8 = {+, *, /, sqrt}, F_9 = {+, *, /, log}, F_10 = {+, *, /, cos, exp}. Other parameters are: population size 50, recombination probability 0.70, mutation probability 0.10, transposition probability 0.10, the number of maximum generation 250. The length of chromosomes is 30. For each problem, the computer runs are repeat 500 times and the results are averaged.

3.1 Fitness assignment

The fitness assignment in GP is described in the following formula:

$$f_i = \sum_{j=1}^{C_t}(M - |C_{(i,j)} - T_j|) \quad (3)$$

Where M is the range selection $C_{(i,j)}$ is the result from the evaluation of individual chromosome i for the test data j and

T_j is target value. The maximum value of this fitness function is

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

The fitness assignment in LGP is described in the following formula:

$$f = \sum_{j=1}^{N}(|O_j - E_j|) \quad (5)$$

With the same parameters as above, O_j is the value returned by a chromosome for the fitness case j and E_j is the expected value for the fitness case j.

The fitness assignment in GEP is described as follow:

$$f = \sum_{j=1}^{N}(M - |O_j - E_j|) \quad (6)$$

The fitness assignment in MEP is described as follow:

$$f = min_{k=1,L}\{\sum_{j=1}^{N} E_j - O_j^k\} \quad (7)$$

where N is the number of fitness cases, O_j^k is the value returned (for the j whatever fitness case) by the k whatever expression encoded in the current chromosome, L is the number of chromosome genes, and E_j is the expected value for the fitness case j.

3.2 Success rate

The metric used to measure the effectiveness of the methods is "Success Rate". The Success Rate is the probability that a method discovered an expression that match exactly to the benchmark functions. It is the percent of the number of successful runs discovered the correct expression from the total number of runs using the parameters specified.

Function	Success Rate (%)			
	GP	LGP	GEP	MEP
$F(x) = x^4 + x^3 + x^2 + x$	100	50.8	76.4	100
$F(x) = \sin(x^4 + x^2)$	9.09	6.6	5.8	22.4
$F(x) = \sin(\exp(\sin(\exp(\sin(x)))))$	100	99.8	93.4	100
$F(x) = 3x^3 + 4x^2 + 2$	0	1.6	12	25.2
$F(x) = \log(x^3 + x^2)$	66.67	7.6	7	48
$F(x) = \tan(x)\sin(x^2)$	53.33	24.4	25	96.2
$F(x) = \frac{1}{x^2} + \sqrt{x}$	0	2.2	0.6	27
$F(x) = \frac{(x^2 + x)}{y^2}$	0	18.4	30	83
$F(x) = \log(x^2 + y) + x$	0	2.8	5	17.6
$F(x) = e^x + \cos(x^3)$	60	17.4	34.8	20

Table 1 The Success Rate from the experiments. Comparing GP, LGP, GEP and MEP.

3.3 Absolute error

An absolute error measured the accuracy of an individual expression against the target formula. It is computed from the average of the best chromosome in each generation using the following formula.

$$e_j = \frac{1}{n}\sum_{i=1}^{n}|c_i - f_i| \quad (8)$$

Where e_j is the absolute error of the run j, c_i is the target value of the function, f_i is the value of the fitness of an individual chromosome i. The average absolute error is averaged over all e_j. The figure 3 to 6 show the progression of the average absolute error as a function of time (generation) for all methods.

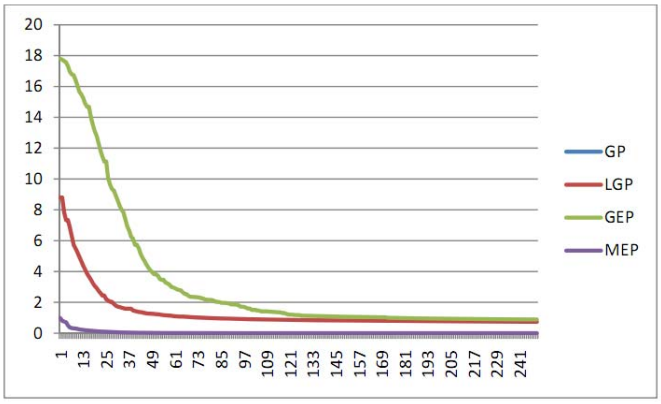


Figure 3 Graph absolute error of function F_7

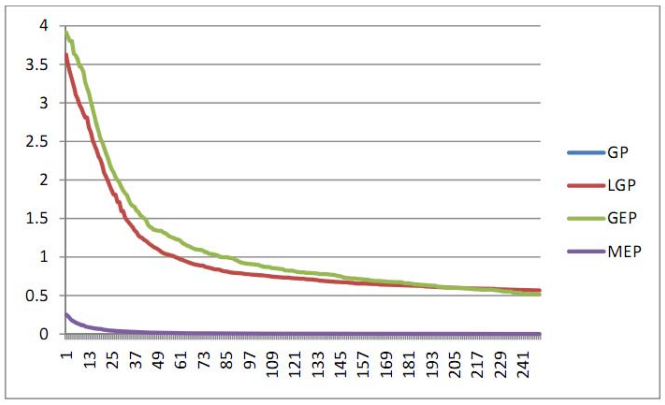


Figure 4 Graph absolute error of function F_8

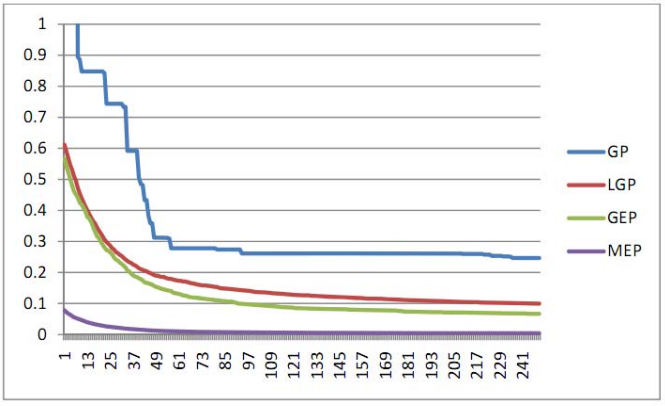


Figure 5 Graph absolute error of function F_9

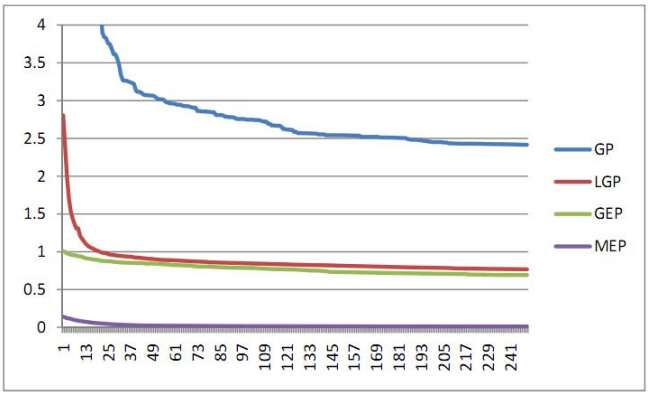


Figure 6 Graph absolute error of function F_10

4. CONCLUSION AND DISCUSSION

From the Success Rate in Table 1, MEP has the highest value for all benchmarks except F_5 and F_10. GEP, LGP and GP are better in some cases. The reason for the effectiveness of MEP can be explained from its ability to represent multi-expression. Because one individual in MEP can has several expression, it has higher chance to discover a solution.

In some cases GP gets approximately the answer similar to MEP. Because GP represents an individual without special encoding, genetic operators work on this representation in a direct way so it always generates a correct form of solutions.

LGP does not work well because the crossover operator can create a lot of "unused" part of chromosome. This results in a larger search space. To improve its effectiveness, perhaps the population size must be larger than other methods.

In terms of convergent rate of the error within 250 generations, MEP converges the fastest in all problems. GP is faster than GEP and LGP for some benchmarks. Comparing GEP and LGP, they have similar convergent rate.

REFERENCES

[1] Abraham, A.; Nedjah, N. & Macedo Mourelle, L. d. Evolutionary Co mputation: from Genetic Algorithms to Genetic Programming Studies in Computational Intelligence (SCI), 2006, 13, 1-20

[2] Rodkaew, Y. and Chongstitvatana, P., "How to reduce the memory requirement in Genetic Programming", Annual National Symposium on Computational Science and Engineering, Bangkok, 2000.

[3] Oltean, M. Evolving Evolutionary Algorithms Using Linear Genetic Programming Evolutionary Computation, 2005, 13, 387-410

[4] Koza, J., Genetic Programming II: Automatic Discovery of Reuseable Programs, MIT Press, 1996.

[5] Ferreira, C. Gene Expression Programming in Problem Solving WSC 6 tutorial, 2001

[6] Oltean, M. & Grosan, C. Evolving Evolutionary Algorithms using M ulti Expression Programming The7th European Conference on Artificial Life, 2003, 651-658

[7] Ferreira, C. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems Complex Systems, 2001, 13, issue 2: 87-129

[8] Mihai, O. and Crina, G., “A Comparison of Serveral Linear Genetic Programming Techniques,” 2003