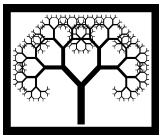
Paper 48



Proceedings of the Eighth International Conference
on Engineering Computational Technology,
B.H.V. Topping, (Editor),
Civil-Comp Press, Stirlingshire, Scotland

Application of Grammatical Evolution to the Determination of Constitutive Equations

E. Kita, T. Kuroda, H. Sugiura, Y. Zuo and Y. Wakita
Graduate School of Information Sciences
Nagoya University, Japan

Abstract

Grammatical evolution (GE), which is a kind of evolutionary algorithm, can find an executable program or program fragment that will achieve a good fitness value for the given objective function to be minimized. The search algorithm of GE is very similar to the genetic algorithm (GA) except for the translation rules from genotype (bit-string) to phenotype (function or program).

In this study, an original GE and three improved algorithms are applied for determining the constitutive relationship in the plane strain state. The numerical results show the convergence speed of the improved algorithm is faster than the original GE.

Keywords: grammatical evolution, constitutive relationship, plane strain state.

1 Introduction

Genetic Algorithms (GA) [1] and Genetic Programming (GP) [2] are very popular evolutionary computation techniques. Their objectives, however, are a little different. While the aim of traditional GA is to find the optimal or better solution of the optimization problem, GP is designed for finding the function representation or executable program or program fragment. In this study, Grammatical Evolution (GE) is applied for finding the relationship between physical quantities, i.e. Hooke's law.

Grammatical evolution (GE), which is also a kind of evolutionary computation technique, was firstly presented by Ryan, Collins and O'Neill [3, 4]. It is related to the idea of genetic programming in that the objective is to find an executable program or program fragment that will achieve a good fitness value for the given objective function to be minimized. The interesting feature of the GE is to represent functions and programs by the help of binary numbers (bit-strings), instead of tree structure in GP. In the GE, the translation rules from genotype (bit string) to phenotype (function

or program) are defined according to the Backus-Naur form (BNF). Except for the use of translation rules, the algorithm is very similar to traditional GA. In the GA, a population of abstract representations (chromosomes or genotype) of potential solutions (individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions by using the genetic operators such as crossover, selection, mutation, and so on. Traditionally, solutions in GA are represented in binary as strings of 0s and 1s. The GE translates the strings of GA potential solutions to functions and programs according to the BNF syntax.

In this study, after introduction of the original GE, three improved algorithms are presented. An original GE has two difficulties. One is related to the rule selection scheme and another is to the selection probability of candidate symbols. In the original GE, the rules are selected by the remainder. The scheme 1 adopts the special roulette selection, instead of the remainder selection. In the scheme 2 and 3, the biased selection probability is adopted to the recursive and terminal rules. In the numerical example, the GE is applied for finding the constitutive relationship between stress and strain in the plain strain state.

The remaining part of this paper is organized as follows. In section 2, the original GE algorithm and its simple example is explained. In section 3, the improved algorithms are explained. The numerical examples are shown in section 4 and the results are summarized again in section 5.

2 Grammatical Evolution

2.1 Original Algorithm

The algorithm of an original Grammatical Evolution (GE) is simply summarized as follows.

1. A BNF syntax is defined to translate genotype (bit-string) to phenotype (function or program).
2. An initial population is defined with randomly generated individuals.
3. Genotypes are translated to function according to the BNF syntax.
4. Fitness functions of genotypes are estimated.
5. Genetic algorithms update the population.
6. The process is terminated if the criterion is satisfied.
7. Go to step 3.

The translation from genotype to phenotype is as follows.

1. A bit-string is translated to a decimal number every n-bits.
2. A leftmost decimal, a leftmost recursive (nonterminal) symbol, and the number of candidate symbols for α are defined as β, α, and n_α, respectively.
3. The remainder is calculated as $\gamma = \beta \% n_\alpha$.

(A)	<expr> ::= <expr><expr><op>	(A0)
	| <x>	(A1)
(B)	<op> ::= +	(B0)
	| −	(B1)
	| *	(B2)
	| /	(B3)
(C)	<x> ::= x	(C0)

Table 1: BNF Syntax of Function Identification Problem

4. The symbol α is replaced with the γ-th symbol of the candidate symbols.
5. If nonterminal symbols exist, go to step 2.

In the genetic programming (GP), the programs rapidly grow in size over time. This difficult is called as "bloat". For overcoming the difficulty, the maximum size of the programs is restricted in advance. The similar idea is applied to the GE. The maximum size of the programs is restricted to $L_{\max}$.

2.2 Application of Original GE to Function Identification Problem

The function identification problem is very popular problem for GP and GE. GE and GP are applied for the function identification problem in order to confirm their features.

The objective of the function identification problem is

> to find an approximate function $\bar{f}$
> when discrete data $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ is given,

where the parameter n denotes the total number of the discrete data sets. When an exact function f is given, the discrete data are referred to as $y_i = f(x_i)$.

We will consider that the exact function f is given as

$$f(x) = x^4 + x^3 + x^2 + x. \tag{1}$$

The discrete data are generated by estimating equation (1) at $x = -1, -0.9, -0.8, \cdots, 0.9, 1$, $(n = 21)$.

The fitness is estimated from f and $\bar{f}$ as follows:

$$fitness = \sqrt{\frac{1}{21}\sum_{i=1}^{21}[f(x_i) - \bar{f}(x_i)]^2} \tag{2}$$

where f and $\bar{f}$ denote the exact function and the function predicted in GE, respectively. The fitness values are estimated as the value averaged over 50 runs.

Generation	1000
Population size	100
Chromosome	100
Tournament size	5
Crossover rate	0.5
Mutation rate	0.1
Translation bit-size	4bit
Maximum size	$L_{\max} = 100$

Table 2: GE Parameters

Generation	1000
Population size	100
Crossover rate	0.9

Table 3: GP Parameters

A GE syntax in BNF is shown in Table 1. The start symbol is <expr>. The use of the rule (A) replaces the symbol <expr> with the candidate symbol (A0) <expr><expr><op> or (A1) <x>. The use of the rule (B) replaces the symbol <op> with the candidate symbols (B0) +, (B1) −, (B2) $\star$, or (B3) /. The use of the rule (C) replaces directly the symbol <x> with x.

The parameters of GE and GP simulations are shown in Table 2 and 3, respectively. Tournament selection, one-elitist strategy and one-point crossover are employed for both GE and GP. The mutation operator is applied for GE alone.

The convergence history of the best fitness value is shown in Fig.1. The abscissa and the ordinate denote the number of generation and fitness value, respectively. The convergence speed of GE is slower than that of GP. Finally, GE can find a better solution than GP.

3 Improved Grammatical Evolution

3.1 Difficulties of Original GE

The algorithm of the original GE can be improved mainly from the following points. One is the rule selection algorithm and the other is the selection probability of the rules and the candidate symbols.

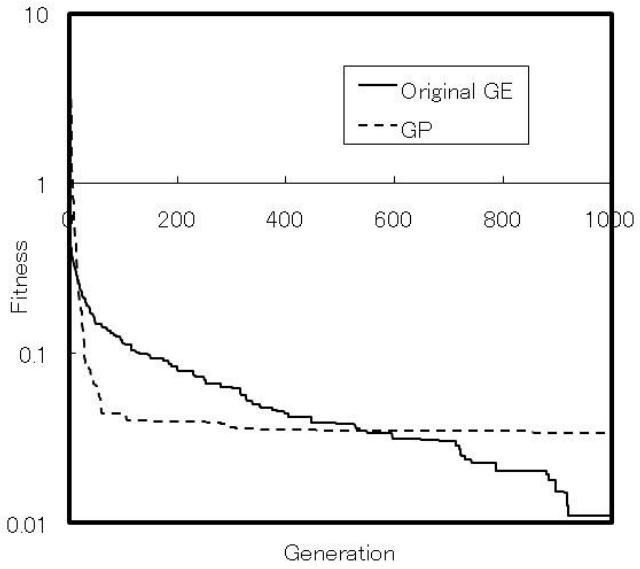


Figure 1: Result of Function Identification Problem

3.1.1 Rule Selection

A leftmost decimal number, a leftmost recursive (non-terminal) symbol, and the number of candidate symbols for α are referred to as β, α, and n_α, respectively. Since a symbol is selected by the remainder $\gamma = \beta \% n_\alpha$, the symbol selection is very sensitive to the variation of the decimal number β. Even when the value of β alters by only one, the selected symbol is changed. This may disturb the development of the better scheme included in the bit-strings. The scheme 1 is designed for overcoming this difficulty.

3.1.2 Selection Probability of Symbols

The original GE selects symbols according to the remainder and therefore, the selection probability for all candidate symbols is identical. For example, in Table 1, the rule <expr> is translated to (A0) <expr><expr><op> or (A1) <x>. The selection probabilities of the symbol (A0) and the symbol (A1) are identical; 50% for each symbol. A biased selection probability for symbols may be better in some problems for improving the convergence speed.

The rules are classified into the recursive (non-terminal) and terminal rules. For example, in Table 1, the symbol (A) is recursive rule and the others are terminal rules. The iterative use of recursive rule makes the phenotype (function or program) longer and more complicated. On the other hand, the terminal rule terminates the development of the phenotype. Since the recursive and the terminal rules have different functions for the algorithm, it is appropriate that the different selection probability is taken for each rule. The following scheme 2 and 3 are designed to control the selection

probability of the recursive and the terminal rules, respectively.

3.2 Improvement of Original GE

3.2.1 Scheme 1

In the original GE, the symbols are selected according to the remainder of the decimal number with the respect to the total number of candidate symbols. The scheme 1 adopts the special roulette selection, instead of the remainder selection. The roulette selection is popular selection algorithm in GA. In the scheme 1, the roulette selection probabilities for all candidates symbols are identical. The objective of the scheme 1 is to encourage the development of the better schemata.

We will consider a leftmost decimal as β, a leftmost nonterminal symbol as α, and the number of candidate symbols for α as n_α. The algorithm is as follows.

1. Calculate the parameter $s_\alpha = \beta / n_\alpha$.
2. Generate a uniform random number p $(0 < p \leq \beta)$.
3. If $(k-1)s_\alpha \leq p < ks_\alpha$, select k-th symbol from the candidate symbols for α $(1 \leq k \leq n)$.

3.2.2 Scheme 2

In scheme 2, the selection probability of the recursive rule is controlled according to the depth of the tree structure. The maximum length of the programs is specified in advance. If the length of the programs is shorter than the maximum depth $L_{\max}$, the selection probability is increased. If not so, the probability is decreased.

The selection probability of the recursive rule i is calculated as

$$P_i^r = 1 - \frac{L}{L_{\max}} \tag{3}$$

where L and $L_{\max}$ denote the depth and the maximum depth of the programs.

3.2.3 Scheme 3

In scheme 3, the selection probabilities of the candidate symbols in the terminal rules are controlled according to the total number of the candidate symbols included in all individuals in the population.

The total numbers of the candidate symbols in the terminal rules in all individuals are counted first. It is assumed that a terminal rule has N^N candidate symbols and that i-th candidate symbol occurs N_i times in all individuals. The selection probability P_i^N of the i-th candidate symbol is calculated as

$$P_i^N = \frac{N_i}{\sum_{j=1}^{N^N} N_j}. \tag{4}$$

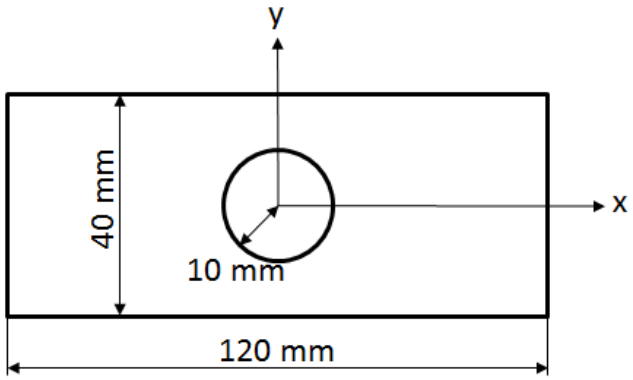


Figure 2: Plate with a circular hole

4 Numerical Example

4.1 Plane Strain Problem

In the plane strain state, the strain components with respect to z-axis are negligible.

The stress and strain vectors at an arbitrary point in the object domain, $\{\vec{\sigma}\}$ and $\{\vec{\varepsilon}\}$ are defined as

$$\{\vec{\sigma}\} = \{\sigma_x, \sigma_y, \tau_{xy}\}^T \tag{5}$$
$$\{\vec{\varepsilon}\} = \{\varepsilon_x, \varepsilon_y, \gamma_{xy}\}^T, \tag{6}$$

where the subscripts denote the x, y and z-axes.

The constitutive relationship, i.e., Hooke's law is defined as

$$\{\vec{\sigma}\} = [D]\{\vec{\varepsilon}\}, \tag{7}$$

where the stiffness matrix $[D]$ is given as

$$[D] = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{(1-\nu)} & 0 \\ \frac{\nu}{(1-\nu)} & 1 & 0 \\ 0 & 0 & \frac{(1-2\nu)}{2(1-\nu)} \end{bmatrix} \tag{8}$$

4.2 Problem Setting

The use of the GE determines the stiffness matrix $[D]$ in equation (7) from the pairs of the stress vector $\{\vec{\sigma}\}$ and the strain vector $\{\vec{\varepsilon}\}$.

The pairs of the stress and the strain vectors are estimated in the one-dimensional tensile problem of the plate with a hole (Fig.2). The pairs of the stress and the strain

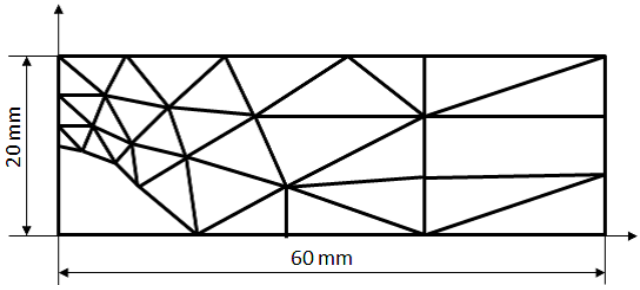


Figure 3: Finite element discretization

vectors for GE simulation are calculated by using the finite element method [5]. Triangle finite element discretization of the quarter part of the object domain is shown in Fig.3. Total numbers of the elements and the nodes are 34 and 27, respectively.

The stiffness matrix determined by GE is defined as

$$[\bar{D}] = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \tag{9}$$

Since the stress and the strain components in the z-axis direction are independent of the other components, the components D_{11}, D_{12}, D_{21} and D_{22} in equation (9) is determined by GE.

BNF syntax is listed in Table 4. The simulation parameters are shown in Table 5. The fitness is defined by the least square errors as

$$fitness = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[\{D_{11}\varepsilon_x + D_{12}\varepsilon_y - \sigma_x\}^2 + \{D_{21}\varepsilon_x + D_{22}\varepsilon_y - \sigma_y\}^2]}. \tag{10}$$

Fifty simulations are performed and the average values are estimated.

4.3 Result

The stiffness matrix determined by Original GE is as follows.

$$[\bar{D}] = E \begin{bmatrix} \nu + 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix} \tag{11}$$

The stiffness matrices in equations (8) and (11) are very similar.

The convergence histories of the fitness of the best individuals are shown in Fig.4. The figure is plotted with the generation as the horizontal axis and the fitness as the vertical axis, respectively.

The convergence properties of the original GE and the scheme 1 and 1+3 are similar. The results by the use of scheme 1+2 and 1+2+3 show the faster convergence

(A)	<expr> ::= <expr><op><expr>	(A0)
	| <var>	(A1)
(B)	<var> ::= <const>	(B0)
	| <num>	(B1)
(C)	<op> ::= +	(C0)
	| −	(C1)
	| *	(C2)
	| /	(C3)
(D)	<const> ::= E	(D0)
	| ν	(D1)
(E)	<num> ::= 1	(E0)
	| 2	(E1)

Table 4: Translation rules

Max. generation	500
Number of individuals	100
Chromosome length	100
Selection	Tournament
Tournament size	5
Number of elite individuals	1
Crossover	One-point
Crossover rate	0.9
Mutation rate	0.1
Radix conversion	Every 4 bit
Max. length of sentence	$MaxN = 100$

Table 5: Parameters

property than them by the others. Therefore, the scheme 2 is very effective for this problem and the combinational use of the scheme 2 and 3 is the most promising among them.

5 Conclusion

Grammatical evolution (GE) is one of the evolutionary computations, which can represent tree structures such as functions and programs by the binary number. The use of the BNF syntax transforms binary numbers to functions or programs. After the original GE algorithm was described, three improved algorithms were explained for improving the convergence property of the original GE. The improved algorithms were named as scheme 1, scheme 2, and scheme 3, respectively.

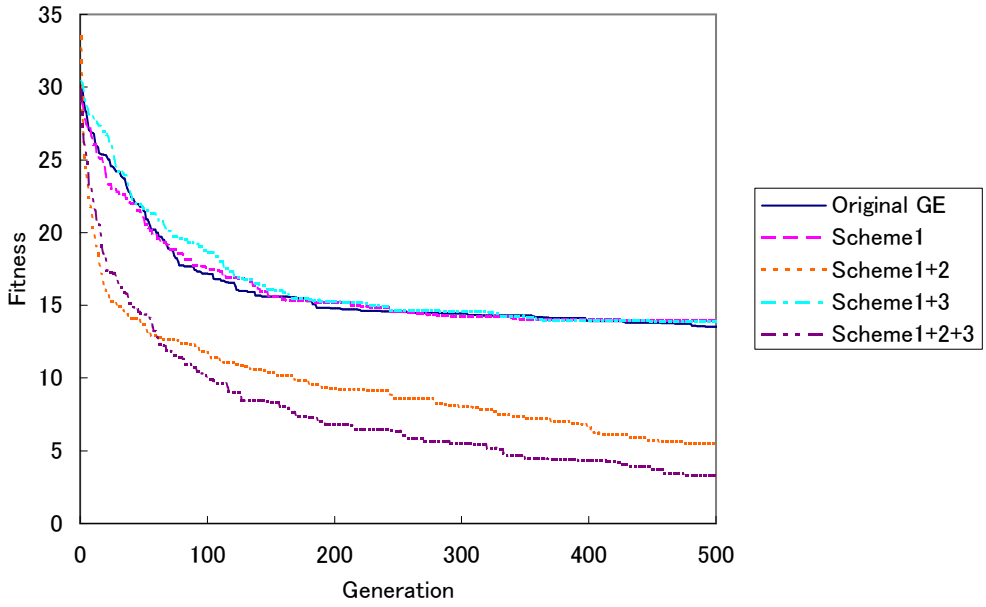


Figure 4: Comparison of convergence speed

GEs were applied for determining the constitutive equation between stress and strain components in the plane strain state. GE could find the similar stiffness equation as Hooke's law, which is determined from the experiments. Comparing the convergence properties of the schemes showed that the scheme 1+2+3 is the fastest and the scheme 1+2 is the second-fastest. Therefore, we can conclude that the scheme 3 is effective for this problem.

References

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1 edition, 1989.

[2] J. R. Koza, editor. *Genetic Programming II*. The MIT Press, 1994.

[3] C.Ryan, J.J.Collins, and M.O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of 1st European Workshop on Genetic Programming*, pp. 83–95. Springer-Verlag, 1998.

[4] C.Ryan and M.O'Neill. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Springer-Verlag, 2003.

[5] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method*. McGraw-Hill Ltd., 4 edition, 1991.