*2012 International Conference on Industrial and Intelligent Information (ICIII 2012)*
***IPCSIT vol.31 (2012) © (2012) IACSIT Press, Singapore***

# Santa Fe Trail Problem Solution Using Grammatical Evolution

Hideyuki Sugiura [1], Takao Mizuno [1] and Eisuke Kita [1,2]

[1] Graduate School of Information Science, Nagoya University
[2] Graduate School of System Informatics, Kobe University

**Abstract.** Santa Fe Trail Problem is a well-known Genetic programming exercise in which Artificial Ants search for food pellets according to a programmed set of instructions. In this paper, three improved algorithms of Grammatical Evolution are applied for the problem. The scheme 1 adopts the special roulette selection, instead of the remainder selection which is popular in original Grammatical Evolution. The scheme 2 and 3 use the biased selection probabilities for recursive and terminal rules, respectively. The effectiveness of three schemes is discussed in the example.

**Keywords:** Grammatical Evolution, Santa Fe Trail Problem, Roulette Selection, Biased Selection Probability.

## 1. Introduction

Grammatical evolution (GE) is classified into evolutionary computations [1, 2, 3]. The basic concept 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.

GE can represent the tree structures such as functions and programs by using bit-strings. For that purpose, the translation rules are defined firstly according to the Backus-Naur form (BNF) to translate bit string (genotype) to function or program (phenotype). Except for the translation rules, the algorithm is very similar to GA. The objective of GA, by the way, is to find the optimal or better solution of the optimization problem. In the GA, a population of bit strings of candidate solutions for 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 bit-strings of GA candidate solutions to the tree structure according to the BNF syntax, which is very different from the GA.

In this study, three improved schemes for GE are applied for Santa Fe Trail problem. The scheme 1 is to use the special roulette selection instead of the remainder selection of the original GE. The scheme 2 is to control the selection probability of the recursive rule according to the length of the generated program. The scheme 3 is to control the selection probability of the terminal rule according to the numbers of terminal rules employed in all individuals.

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

## 2. Grammatical Evolution

### 2.1. Original Grammatical Evolution Algorithm

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

1) Define BNF syntax to translate bit-string to program.
2) Define an initial population with randomly generated individuals.

3) Translate bit-string to program according to the BNF syntax.
4) Estimate fitness functions of bit-string.
5) Update the population.
6) Terminate the process if the criterion is satisfied.
7) Go to step 3).

The translation from bit-string to program is as follows.

1) Translate a bit-string to decimal numbers every $n$-bits.
2) Define a leftmost decimal and a leftmost recursive symbol as $\beta$ and$\alpha$, and the number of candidate symbols for $\alpha$ as$n_\alpha$, respectively.
3) Calculate the remainder$\gamma = \beta\%n_\alpha$.
4) Replace the symbol $\alpha$ with the $\gamma$-th symbol of the candidate symbols.
5) If recursive symbols still exist, go to step 2).

In the genetic programming (GP), the programs often grow rapidly in size over time (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. Difficulties of Original GE

1) Remainder Rule Selection

Since, in the original GE, a rule is selected by the remainder$\gamma = \beta\%n_\alpha$, the rule selection is very sensitive to$\beta$. Even when the value of $\beta$ alters by only one, the different rule is selected. This may disturb the development of the better scheme in the population. The following scheme 1 is designed for overcoming this difficulty.

2) Selection Probability of Rules

The original GE selects a rule according to the remainder and therefore, the selection probability for all candidate rules is identical. However, biased selection probability may be better in some problems for improving the convergence speed.

The rules are classified into the recursive and terminal rules. The iterative use of recursive rule makes the program longer and more complicated. On the other hand, the terminal rule terminates the development of the phenotype. Since the roles of the recursive and the terminal rules are different, 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.

## 2.3. Improved GE

### 2.3.1 Scheme 1

The scheme 1 is designed to overcome the difficulty related to the remainder rule selection. For this purpose, 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 probability for all candidate rules is identical. The use of the roulette selection can encourage the development of the better schema.

We will consider a leftmost decimal as$\beta$, a leftmost non-terminal symbol as$\alpha$, and the number of candidate rules for $\alpha$ as$n_\alpha$. The maximum value of $\beta$ is referred to as$\beta_{max}$. The algorithm is as follows.

1) Calculate the parameters$s_\alpha = \beta_{max}/n_\alpha$.
2) If$(k-1)s_\alpha < \beta \leq ks_\alpha$, select $k$-th rule from the candidate rules for $\alpha$ $(1 \leq k \leq n)$.

### 2.3.2 Scheme 2

The selection probability of the recursive rule is controlled according to the length of the generated program. The maximum length of the programs is specified in advance. If the length of the programs is shorter than the maximum length$L_{max}$, the selection probability is increased. If not so, the probability is decreased.

The selection probability of the recursive rule $i$ is calculated as

$$P_i^r = 1 - \frac{L}{L_{max}} \quad (1)$$

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

#### 2.3.3 Scheme 3

Numbers of terminal rules employed in all individuals are counted first and then, the selection probability of the terminal rule $i$ is calculated as

$$P_i^N = \frac{N_j}{\sum_{j=1}^{N^N} N_j} \quad (2)$$

where $N_i$ and $N^N$ denote the total number of the terminal rule $i$ and the total number of the terminal rules, respectively.

Table 1: BNF for Santa Fe Trail Problem

| | |
|---|---|
| (a) | <code>::= <code><code><expr> \| <expr> |
| (b) | <expr> ::= <if_statement> \| <op> |
| (c) | <if_statement> ::= if_food_ahead {<op>} else {<op>} |
| (d) | <op> ::= right \| left \| forward |

## 3. Numerical Examples

### 3.1. Santa Fe Trail Problem

The object of Santa Fe Trail problem is to automatically find the program to control artificial ants which efficiently collect foods in the region [4, 5]. In this study, the region is $32 \times 32$-grid region in which there are 89 foods. The ant behaviour is defined as follows.

I. Ants start from upper-left cell and can move in vertical and horizontal directions alone.

II. Ants move one cell every time-step and cannot get across the walls.

III. Ants have sensor to find food.

IV. Ants have life energy which is expended one unit every time step.

The BNF syntax of this problem is shown in Table 1. The expression “right”, “left” and “forward” indicates that an ant turns to the right, turns to the left and moves forward, respectively. The expression “if_food_ahead{ }else{ }” denotes that the first argument is performed if there exists a food in the front cell of the ant and the second argument is performed if there does not exit a food. The start symbol is “<code>”. Only rule (D) is the recursive rule and the others are terminal ones. The scheme 2 is available for rule (D) alone, and the scheme 3 is for the others.

The fitness is defined as the difference between the numbers of the foods before and after the simulations:

$$Fitness = 89 - F \quad (3)$$

where the value 89 means the number of the foods before the simulation starts and the parameter $F$ is the number of foods which the ant could collect.

The simulation algorithm is as follows.

1) Specify the initial ant energy as 400 and the number of collected foods as 0.
2) Move the ant and decrement the ant energy.
3) If there exists a food there, collect the food and $F = F + 1$.
4) If the energy is equal to 0, estimate the fitness from equation (3).
5) Go to step 2).

The parameters are given as follows; maximum Generation=500, Population size=100, Chromosome length=100, Tournament size=5, Crossover rate=0.9, Mutation rate=0.01, translation bit-size from binary to decimal is every 4bit and maximum length of program $L_{max} = 100$. Moreover, tournament selection, one-

elitist strategy and one-point crossover are employed. Simulation is performed 50 times at different initial population.

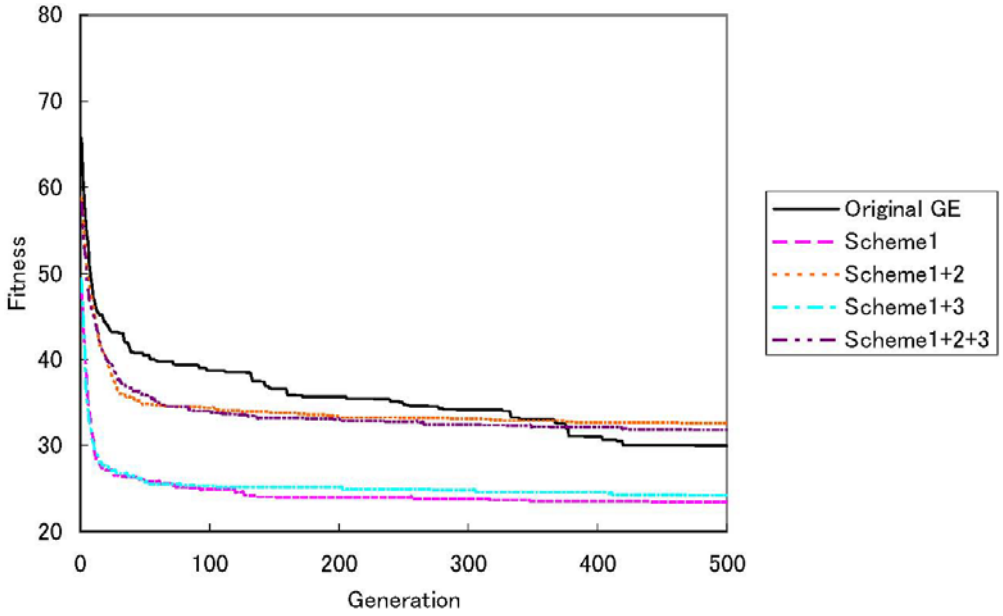


Figure 1: Convergence History of Best Individual Fitness (Crossover Rate = 0.9 Mutation Rate = 0.01)

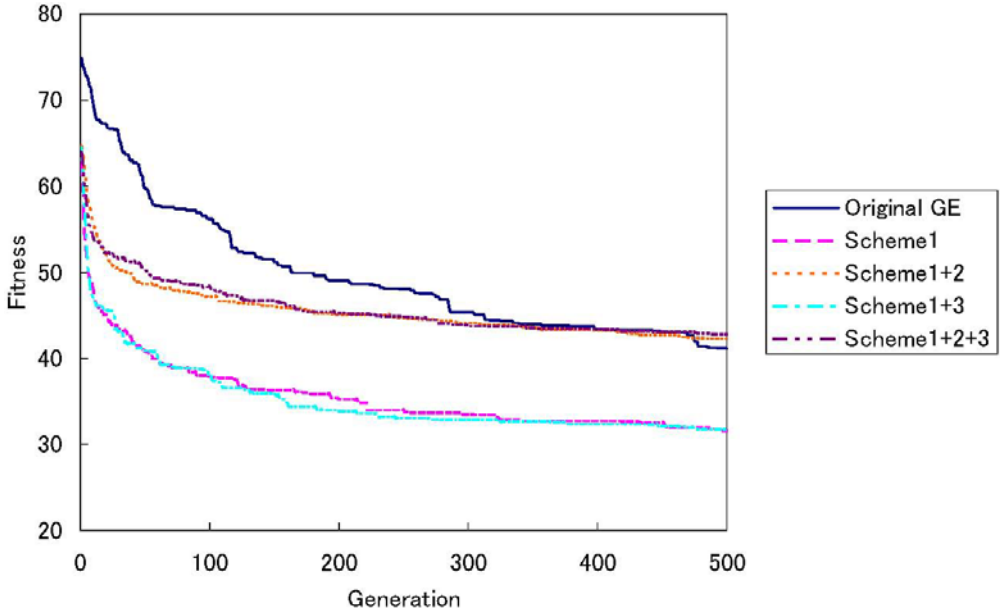


Figure 2: Convergence History of Best Individual Fitness (Crossover Rate = 0.5 Mutation Rate = 0.01)

## 3.2. Results

The convergence history of the fitness of the best individual fitness is shown in Fig.1. The abscissa and the ordinate denote the generation and the average fitness. The scheme 1 can improve the convergence speed. The scheme 1+3 is better than the original GE but lightly worse than the scheme 1. The scheme 1+2 and 1+2+3 show the worse result than the original GE. Therefore, we can conclude that only the scheme 1 is effective for improving the search performance in this problem.

Next, we would like to discuss the effect of the crossover rate. The crossover rate is changed from 0.9 to 0.5. The other parameters are same as in Fig.1. The convergence histories are shown in Fig.2. The comparison of Figs.1 and 2 shows that the crossover rate reduction makes the final best individual worse and also goes down the convergence speed.

Finally, we would like to discuss the effect of the mutation rate. The mutation rate is changed from 0.01 to 0.1. The other parameters are same in Fig.1. The convergence histories are shown in Fig.3. The comparison of Figs.1 and 3 shows the convergence speed of scheme 1 in 3 is slower than that in Fig.1.

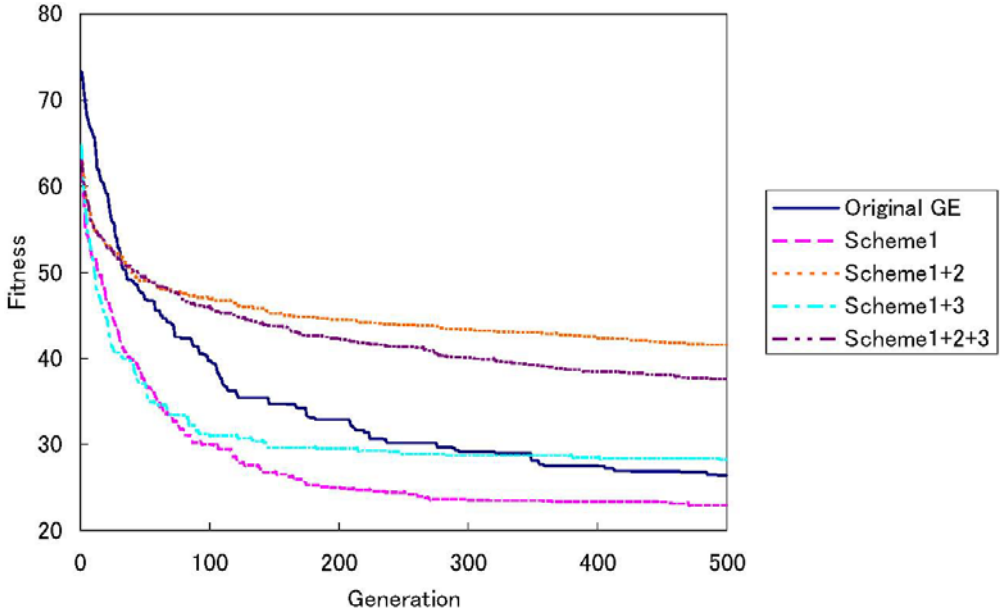


Figure 3: Convergence History of Best Individual Fitness (Crossover Rate = 0.9; Mutation Rate = 0.1)

## 4. Conclusions

Three improved schemes for the grammatical evolution were presented in this study. The scheme 1 uses roulette selection, instead of remainder selection, in order to encourage the development of the better scheme in the chromosome. The scheme 2 and 3 give the biased selection probability for the candidates of recursive and terminal rules, respectively.

In the numerical examples, three schemes were compared in Santa Fe Trail Problem. The results showed that the scheme 1 was very effective and the others were not useful. The above result indicates the effectiveness of roulette selection for the problem to be solved. Finally, the effect of the crossover and mutation rates was discussed. Although the scheme 1 is the most effective in all cases, the convergence speed strongly depends on the crossover and mutation rates. Therefore, we have to design the algorithm which can find the best crossover and mutation rates adaptively.

## 5. References

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

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

[3] M. O'Neill and C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation, Vol.5, No.4, pages 349-358, 2001.

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

[5] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III*. Morgan Kaufmann, 1999.