The Effect of Mutation Operation on GP- based Stream Ciphers Design Algorithm

Wasan Shakr Awad
Department of Information Systems, College of Information Technology, University of Bahrain, Sakheer, Bahrain.

Keywords: Genetic Programming, Simulated Annealing, Stream Ciphers, Mutation Operation.

Abstract: Mutation operation is used to introduce a small perturbation in the population from time to time so as to maintain its diversity. Several mutation operations have been developed for genetic programming. This paper is to study the impact of mutation operation on the performance of genetic programming. We present six types of mutation operations that have been applied in the simulated annealing programming (SAP) algorithm, which is an algorithm used to design stream ciphers using genetic programming and simulated annealing. Experiments performed to study the effectiveness of these operations in solving the underlying problem. It has been shown that mutation operation can affect the performance of genetic programming, especially when it is used to solve complex problems.

1 INTRODUCTION

Evolution is an optimization process, where the aim is to improve the ability of a system to survive in dynamically changing environment. There are a number of evolutionary computation techniques, such as genetic programming (GP). In GP, the population individuals are computer programs represented usually as expression trees (Koza, 1992), and a number of operations are used to update the population individuals, such as crossover and mutation.

In general, the inclusion of mutation step the evolutionary algorithms is very important. The main aim of mutation operation is to keep certain amount of randomness and to introduce a small perturbation in the population from time to time so as to maintain its diversity. Most of the times the mutation operation is applied according to some fixed probabilistic rule. In the past few years mutation operations based on different probability distributions (like Normal, Gaussian, and Cauchy etc) have become popular. Also, in our previous work (Awad, 2011) adaptive mutation has been applied and it has been shown its effectiveness comparable with fixed rate mutation.

Mutation in GP is frequently treated as a secondary operator. However, it has been shown that mutation can significantly improve performance when combined with crossover (Banzhaf , 1996; Poli, 1997). Few of Koza's (Koza, 1992; Koza, 1994) early experiments include mutation. Koza states two reasons for the omission of mutation in the majority of problems. First, it is rare to lose diversity when using a sufficient population size; therefore, mutation is simply not needed in GP. Second, when the crossover operation occurs using endpoints in both trees, the effect is very similar to point mutation. Koza wished to demonstrate that mutation was not necessary and that GP was not performing a simple random search. This has significantly influenced the field, and mutation is often omitted from GP runs. While mutation is not necessary for GP to solve many problems, it can perform as well as crossover based GP in some cases. O'Reily (O'Reily, 1995) argued that mutation in combination with simulated annealing or stochastic iterated hill climbing can perform as well as crossover-based GP in some cases. Nowadays, mutation is widely used in GP, especially in solving complex problems. Koza also advises to use of a low level of mutation (Koza, 1999). In (Luke, 1997), the authors suggested that the situation is complex, and that the relative performance of crossover and mutation depends on both the problem.

Several mutation operations have been developed for GP; some of them are listed bellow (Koza, 1992; Koza, 1994; O'Reily, 1995; Kinnear, 1994; Angeline, 1996):

Shakr Awad W..
The Effect of Mutation Operation on GP- based Stream Ciphers Design Algorithm.
DOI: 10.5220/0004222804450450
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (ICAART-2013), pages 445-450
ISBN: 978-989-8565-39-6

- Subtree mutation replaces a randomly selected subtree with another randomly created subtree.
- Node replacement mutation (also known as point mutation) is similar to bit string mutation in that it randomly changes a point in the individual. A node in the tree is randomly selected and randomly changed.
- Hoist mutation creates a new offspring individual which is copy of a randomly chosen subtree of the parent. Thus, the offspring will be smaller than the parent and will have a different root node.
- Shrink mutation replaces a randomly chosen subtree with a randomly created terminal.
- Permutation mutation selects a random function node in a tree and then randomly permuting its arguments (subtrees).

In addition to these types, two new methods for implementing the mutation operator in GP called Semantic Aware Mutation (SAM) and Semantic Similarity based Mutation (SSM) have been proposed (Nguyen, 2009).

Designing good stream cipher automatically is a complex process, therefore GP has been used in our previous work (Awad, 2011) for designing one class of stream cipher systems, and we have showed that GP can be used successfully to solve this problem. In (Awad, 2011) GP, integrated with simulated annealing, (SA) (Kirkpatrik, 1983) i.e., simulated annealing programming (SAP), has been used successfully to design Linear Feedback Shift Register (LFSR)-based stream cipher systems with the desired properties, such as random keystream, and large period length. However, we still expect some improvements. Thus, this paper is to study the impact of various mutation operations on the effectiveness of SAP algorithm for designing stream ciphers.

Six types of mutation operation are applied in this work, which are:

- GA-Based Terminal Node Mutation
- Random Terminal Node Mutation
- GA-Based Semantic Terminal Node Mutation
- Random Sub-expression Mutation
- GA-based Shrink Mutation
- GA-based Semantic Shrink Mutation

2 SAP ALGORITHM OVERVEIW

SAP is a general automated design approach for designing stream ciphers that satisfy the desired properties. It is an integration of GP and SA in order to work on a population of individuals and to preserve good individuals into the next generation (Yuichiro, 2009; Miki, 2007). The output of SAP is a keystream generator, which is the main component of stream cipher that generates pseudorandom Binary sequence (keystream) of length *size* and fulfills the security and efficiency requirements. Stream ciphers (Forouzan, 2008; Rueppel, 1986; Schneier, 1996; Golomb, 1967) are of great importance in applications, and there are different types of stream ciphers, only LFSR-bassed stream ciphers are considered here, in which, the main component of the keystream generators is LFSR, where LFSR is a shift register with linear feedback function.

The following is the complete SAP algorithm as described in (Awad, 2011):

```
Algorithm: SAP
Input : Keystream period length
  (size).
Output : LFSR-based keystream
  generator.
Begin
  Generate the initial population
  (pop) randomly;
  Evaluate pop;
  Temp ←250.0;   //temperature
  Repeat
   Generate a new population (pop1) by
   applying crossover and mutation;
   Evaluate the fitness of the new
   generated chromosomes of pop1;
   Calculate the averages of fitness
   values for pop and pop1, av1 and av2
   respectively;
   If (av2 > av1) then  replace the old
   population by the  new one, i.e.
   pop ← pop1;
   Else
      Begin
        e ← av1-av2;
        Pr ←  e / Temp;
        Generate a random number (rnd);
        If (exp(-pr) > rnd) then
        pop ← pop1;
      End;
   Temp ← Temp * 0.95;
  Until (Max Number of generations);
  Return the best chromosome of the
  last generation;
End.
```

In SAP algorithm, SA is the technique used for the construction of the keystream generators. The structure under adaptation is the set of GP expressions, and the GA operations are used to update the population of expressions.

In this algorithm, the genetic operations used are 1-point crossover with probability pc=1.0, and terminal node mutation with probability pm=0.1. The mutation operation used to replaces a randomly selected one gene from a LFSR feedback function with a new one. The function library includes the functions: LFSR, XOR, AND, OR. The terminal nodes are strings of characters (which are converted to Binary digits during the process of fitness evaluation) that represent the linear feedback functions of LFSRs.

For example, the following population individual (chromosome):

"& SR adfe SR ddeab" is an expression of two shift registers combined by AND function, and "adfe" and "ddeab" are feedback functions of the LFSRs in the expressions.

3 MUTATION OPERATION DESCRIPTIONS

Different types of mutation operation are applied in SAP algorithm. The descriptions of theses operations are given bellow.

3.1 Terminal Node Mutation

This operation randomly changes a point (terminal node) in the individual. The point is the feedback function of a LFSR. Three variations of this operation are considered in this study, as follows.

I. GA-Based Terminal Node Mutation

Here, Genetic Algorithm (GA) is used to find the best LFSR feedback function to replace a randomly selected terminal node (LFSR feedback function) of an individual. The description of this operation is as follows.

The population chromosomes in GA are represented as fixed length Binary strings. The lengths of these strings are equal to the length of the randomly selected terminal node *S* of SAP individual, where S is the Binary feedback function of a LFSR. Each chromosome in GA population is a mask used to modify S, by Xoring GA chromosome with S. The probability of generating the gene "1" in the GA initial population is 0.2, while the probability of the gene "0" is 0.8.

The fitness value is a measurement of the goodness of the keystream generator, and it is used to control the application of the operations that modify a population. The fitness function used in SAP is also used in GA. After modifying S, the SAP program is executed to generate the keystream. The generated keydtream is then evaluated as described in (Awad, 2011). Eqs. (1), (2), and (3) are used to evaluate the GA chromosomes. Eq. (1) is used for the evaluation of keystream randomness using the frequency and serial tests, in which, *nw* is the frequency of w (where w = 00, 01, 10, or 11) in the generated binary sequence.

$$f1 = |n_0 - n_1| + \sum \left| nw - \frac{size}{4} \right| \quad (1)$$

There is another randomness requirement which is: $(1/2^i * n_r)$ of the runs in the sequence are of length *i*, where n_r is the number of runs in the sequence. Thus, we have eq. (2).

$$f2 = \sum_{i=1}^{M} \left| \left(\frac{1}{2^i} \times n_r \right) - n_i \right| \quad (2)$$

where *M* is maximum run length, and n_i is the desired number of runs of length *i*. Thus, the fitness function used to evaluate the chromosome *x* will be as given by eq. (3), where *wt* is a constant and *size* is the keystream period length:

$$fit(x) = \frac{size}{1 + f1 + f2} + \frac{wt}{length(x)} \quad (3)$$

The parameters used in this work were set based on the experimental results, the parameter value that show the highest performance was chosen to be used in the implementation of the algorithm. Thus, the genetic operations used to update the population are 1-point crossover with probability pc=1.0. The selection strategy, used to select chromosomes for the genetic operations, is the 2- tournament selection. The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations. The run of GA is stopped after a fixed number of generations. The solution is the best chromosome of the last generation.

II. Random Terminal Node Mutation

Using this operation, S is replaced by randomly generated LFSR feed back function.

III. GA-Based Semantic Terminal Node Mutation

GA is used to find the best LFSR feed back function, in term of the big differences in the generated keystreams, to replace a randomly selected terminal node (LFSR feed back function) of an individual. The description of this operation is as follows.

The population chromosomes GA are represented as fixed length Binary strings. The

lengths of these strings are equal to the length of the randomly selected terminal node *S* of SAP individual. Each chromosome in GA population represents the candidate LFSR feedback function to replace S.

The chromosomes of GA population are evaluated based on the big difference in the generated keystreams generated by S and GA individuals. Thus, eq. (4) used to compute the fitness value of the GA chromosome x.

$$fit(x) = \sum_{i=1}^{size} \left| keystream_i - keystream'_i \right| \quad (4)$$

Where $keystream_i$ is the ith bit of the keystream generated by S, and $keystream'_i$ is the ith bit of the keystream generated by the LFSR with the GA chromosome x as feedback function.

The genetic operations used to update the population are 1-point crossover with probability pc=1.0. The selection strategy, used to select chromosomes for the genetic operations, is the 2-tournament selection. The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations. The run of GA is stopped after a fixed number of generations.

3.2 Sub-expression Mutation

By applying this operation, a sub-tree is selected randomly from SAP expression, and then is replaced by a new sub-expression. In this work, different sub-expression mutations have been studied. The following is the description of each one.

I. Random Sub-expression Mutation
Using this operation the sub-expression of SAP expression is replaced by randomly generated expression.

II. GA-based Shrink Mutation
This operation is used to reduce the size of SAP expression, by finding a LFSR that is equivalent to a randomly selected sub-expression. The root node of the sub-expression should be any function other than LFSR. GA is used to find an equivalent LFSR that generates the same keystream generated by the selected sub-expression.

The population chromosomes in GA are represented as variable length Binary strings. Each chromosome in GA population represents the candidate LFSR feedback function.

The solution of GA is an equivalent LFSR that generates the same keystream generated by the sub-expression selected randomly from a SAP expression. Thus, Eq. (6) is used to compute the fitness value of the GA chromosome x.

$$f3 = \sum_{i=1}^{size} \left| keystream_i - keystream'_i \right| \quad (5)$$

$$fit(x) = \frac{size}{1 + f3} \quad (6)$$

Where $keystream_i$ is the ith bit of the keystream generated by the sub-expression, and $keystream'_i$ is the ith bit of the keystream generated by the LFSR with the GA chromosome x as feedback function.

The genetic operations used to update the population are 1-point crossover with probability pc=1.0. The selection strategy, used to select chromosomes for the genetic operations, is the 2-tournament selection. The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations. The run of GA is stopped after a fixed number of generations. The solution is the best chromosome of the last generation.

III. GA-based Semantic Shrink Mutation
It is the similar to previous operation, but GA is used here to find an equivalent LFSR that generated different keystream, thus the fitness function used is given by eq. (4).

4 RESULTS

This section presents the findings and results of the experiments carried out to demonstrate the impact of different types of mutation operations on the effectiveness of SAP algorithm. Therefore, SAP has been implemented with six types of mutation operations mentioned above. The six algorithms have been applied with different mutation rates. The best algorithms parameters used in the experiments are:

- Max number of generations of SAP = 30
- Max number of generations of GA = 10
- Pop size of SAP = 100
- Pop size of GA (if used) = 10

The results of applying the six algorithms are presented in table 1. These results are obtained by running each algorithm 100 times for different values of mutation rates. The results of table 1 represent the average of the fitness values of the best chromosomes in 100 runs. According to the results and by applying Wilcoxon signed-rank test, GA-Based Terminal Node Mutation has been greatly

Table 1: Fitness values averages of 100 runs.

Mutation Rate%	GA-Based Terminal Node Mutation	Random Terminal Node Mutation	Random Sub-expression Mutation	GA-Based Semantic Terminal Node Mutation	GA-based Shrink Mutation	GA-based Semantic Shrink Mutation
0	31.6738					
5	37.3666	30.4139	31.1553	29.4109	24.4812	29.2382
10	43.7615	31.3691	31.4582	33.307	27.5608	29.8604
15	44.2341	31.3691	33.2401	35.4674	26.6744	26.5393
20	44.4663	33.5122	34.5481	35.4996	23.0873	25.3877
30	45.1159	32.8931	35.4401	32.9202	24.6556	23.365
40	47.338	34. 264	35.9812	37.2787	30.7147	25.3943
50	47.452	33.8621	35.5944	38.1934	30.6348	25.9628
60	49.1361	35.3515	36.1849	37.4782	31.3748	26.4893

improved the performance of SAP algorithm; it can evolve keystream generators that generate keystreams of good statistical properties and of large period length. Also, increasing the mutation rate improves the results. That is because; the feedback function of LFSR has great effect on the output the keystream generators. Thus, by using GA to find the best feedback function can highly improve the results.

5 CONCLUSIONS

In this paper, different types of mutation operations have been applied in SAP algorithm in order to study the impact of mutation operation on the algorithm performance. It has been shown that mutation operation can affect the performance of GP, especially when it is used to solve complex problems, such as designing stream ciphers. Six mutation operations have been applied. We found that GA-Based Terminal Node Mutation that replaces the feedback function of LFSR (terminal node) by a new feedback function evolved by GA can highly improve SAP algorithm.

REFERENCES

Angeline, p. j.; 1996. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. *In Proceedings of the First Annual Conference on Genetic Programming*. Stanford University, CA, USA. PP 21-29.

Awad, W. S.; 2011. Designing stream cipher systems using GP. *LNCS,* Vol. 6683, PP. 308-320.

Banzhaf, W., Francone, F. D., and Nordin, N.; 1996. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. *LNCS*, Vol. 1141, PP. 300–309.

Forouzan, B. A.; 2008. *Cryptography and network security*. McGRAW-HILL, New York, USA.

Golomb, S. W.; 1967. *Shift Register Sequence.* Holden-Day, San Francisco, USA.

Kinnear, K. E.; 1994. Fitness landscapes and difficulty in genetic programming. *In Proceedings of the IEEE World Conference on Computational Intelligence*. Vol. 1, PP. 142-147.

Kirkpatrik, S., et al.; 1983. Optimization by simulated annealing. *Science*, 220(4598), 671-680.

Koza, J. R.; 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

Koza, J. R.; 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press.

Koza, J. R., F. H. B., Andre, D., and Keane, M. A.; 1999. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann Press.

Luke, S., and Spector, L.; 1997. A comparison of crossover and mutation in genetic programming. *In Proceedings of the Second Annual Conference on Genetic Programming*. Morgan Kaufmann, PP. 240–248.

Miki, M., Hashimoto, M., and Fujita, Y.; 2007. Program Search with Simulated Annealing. *In Proceeding of the 9th Annual Conference on Genetic and Evolutionary Computation.* PP. 1754 – 1754.

Nguyen Quang Uy, Nguyen Xuan Hoai, and Michael O'Neill; 2009. Semantics based mutation in genetic programming: the case for real-valued symbolic regression. *In Mendel09, 15th International Conference on Soft Computing.* PP. 73-91.

O'Reilly, U.; 1995. *An Analysis of Genetic Programming*. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada.

Poli, R., and Langdon, W. B.; 1997. Genetic programming with one-point crossover. *In Soft omputing in Engineering Design and Manufacturing.* Springer-Verlag, PP. 180–189.

Rueppel, R. A.; 1986. *Analysis and Design of Stream Cipher.* Springer-Verlag.

Schneier, B.; 1996. *Applied cryptography.* John Wiley and Sons.

Yuichiro, U., Mitsunori, and M., Tomoyuki H.; 2009. Simulated Annealing Programming Using Effective Subtrees. *Doshisha Daigaku Rikogaku Kenkyu Hokoku*, 49(4), 205-209.