International Journal of Innovation in Computer Science and Information Technology
Vol.1, NO.3 , P:136 - 143
Received : 08 April 2019
Accepted : 15 June 2019

Automatic Test Data Generation Based on a Modified Genetic Algorithm

Amirhossein Damia
Faculty of Computer Engineering K. N. Toosi University Tehran, Iran
damiaa@email.kntu.ac.ir

Abstract

Software Testing is one of the essential parts of the software development lifecycle and structural testing is one of the most widely used testing principles to test various software. In the structural test, the test data generation is very important. Therefore, the problem becomes a search problem and Search Algorithms can be used. Genetic Algorithm (GA) is one of the widely used algorithms in this field. For the problem that GA suffers from large iteration times and low efficiency in test data generation, this paper proposes a Modified Genetic Algorithm (MGA), in this method, we design the chromosome probability of crossover and mutation which has relationship with chromosome adaptability. Experimental result shows that MGA has faster convergence speed and higher test data generation efficiency compared with traditional GA.

Keywords: Software Testing, Test Data Generation, Search Algorithms, Genetic Algorithm**.**

1. Introduction

Software testing is a main method for improving the quality and increasing the reliability of software now and thereafter the long-term period future. It is a kind of complex, labor-intensive, and time consuming work; it accounts for approximately 50% of the cost of a software system development [1]. Increasing the degree of automation and the efficiency of software testing certainly can reduce the cost of software design, decrease the time period of software development, and increase the quality of software significantly. The critical point of the problem involved in automation of software testing is of particular relevance of automated software test data generation. Test data generation in software testing is the process of identifying a set of program input data, which satisfies a given testing criterion. For solving this difficult problem, structural test data generation technique have been used in the past. Structural test data generation is either static or dynamic [2]; this approach uses the information of the internal structure of programs, while the latter utilizes the runtime information of programs through execution with parametric input values. To measure testing adequacy, i.e. how well a program is tested by a set of test datas, many researchers propose the use of different coverage criteria (e.g. control flow criteria). In addition, testing coverage criteria usually define the termination condition of the testing process. In the static method, there is no need to run the program directly, so there are problems with presentations and pointers. In the dynamic method, the program needs to be run directly and the problems related to the static method have been solved. In recent years, GA have been widely used by researchers to dynamic test data generation. The efficiency of the GA is affected by the crossover rate and the mutation. To solve the problem, a high crossover value and a low mutation value are used using the GA and kept constant. Proper setting of these parameters greatly increases the efficiency of this algorithm. It is often very difficult and time consuming to get such settings. In this paper a MGA based on path coverage criteria is used to test data generation. This method has been proposed that in order to obtain the best value for these parameters, the value of these parameters is dynamically obtained during execution due to the suitability of the population members. The results show that the proposed method is better than the traditional GA with a crossover and mutation rate constant.

2. Genetic Algorithm

GA is proposed by Holland in his book Adaptation in Natural and Artificial Systems in 1975 on the concept of survival of fittest. GA are applied on the variety of problems such as search based, optimization problems and machine learning with AI domain. GA search techniques are rooted in mechanism of assessment and nature genetics. The survival of fittest individuals comes from GA inspiration form the natural search and selection processes. Each individual is represented by chromosome, GA select these chromosome to generate a sequence of populations by using a different selection mechanism [3]. GA have the following basic operations:

- Selection
- Reproduction
- Evolution
- Replacement

The first operation is selection in which individuals are selected for reproduction. So many selection operation already defined, anyone can be applied for selection of individual. These selection operations are Roulette Wheel Selection, Linear Ranking Tournament Selection etc. In reproduction offspring are bred with selected individuals. Crossover and Mutation operations are used in reproduction. In crossover operation two individuals exchange some information (if individuals are represented in binary then some binary numbers) at one point or at multiple points. While in mutation only one bit has been change. The evaluation process checks the fitness on new generated individuals after reproduction. In replacement the new individuals takes the position of old individuals. The old individuals are killed in this process. Figure 1 provides a pseudocode listing of the GA [4].

```
Input: $Population_{size}$, $Problem_{size}$, $P_{crossover}$, $P_{mutation}$
Output: $S_{best}$
Population ← InitializePopulation($Population_{size}$, $Problem_{size}$);
EvaluatePopulation(Population);
$S_{best}$ ← GetBestSolution(Population);
while ¬StopCondition() do
    Parents ← SelectParents(Population, $Population_{size}$);
    Children ← ∅;
    foreach $Parent_1$, $Parent_2$ ∈ Parents do
        $Child_1$, $Child_2$ ← Crossover($Parent_1$, $Parent_2$, $P_{crossover}$);
        Children ← Mutate($Child_1$, $P_{mutation}$);
        Children ← Mutate($Child_2$, $P_{mutation}$);
    end
    EvaluatePopulation(Children);
    $S_{best}$ ← GetBestSolution(Children);
    Population ← Replace(Population, Children);
end
return $S_{best}$;
```

Figure 1. Pseudocode of GA.

This algorithm will stop when fitness meet or until a maximum number of iteration has been taken.

3. Structural Testing Using Genetic Algorithm

Structural Testing checks the internal structure of the program. The tester works with code, loop and condition statement. In some research works, code coverage and path testing are discussed using the GA.

3.1. Path Testing

The aim of path testing is that every possible logical execution path in a program must be exercised at least once. A given test case causes a program to take exactly one logical path. However, one single logical path can be triggered by multiple test datas. If many test datas can cause the execution of one path, but few can cause the execution of another path, searching for test datas that trigger the rare path is like looking for a needle in a haystack. A domain of possible data of a program makes a search space for path testing. Generating data that traverse required paths for path testing adequacy is a search. So, path testing is a search for test data that meets path coverage adequacy. Path testing is the strongest coverage criterion in white structural testing. It poses some challenges: justifying its use in the first place, adequate target paths generation, guiding heuristic, computational complexity, and recognizing infeasible paths. Control flow graph (CFG) uses the path to find all possible paths in the program to find test errors. A CFG is a directed graph G = (V, E), with two distinguished nodes a unique entry $n0$ and a unique exit nk. V is a set of nodes, where each node represents a statement, and a E is a set of of directed edges, where a directed edge $e = (n, m)$ is an ordered pair of adjacent nodes, called tail and head of e, respectively. A path in a CFG is a finite sequence of nodes connected by edges [5].

4. Related work

Recently different techniques have been proposed which are based on GA to test data generation. McMinn [6] and Mantere [7] survey some of the work undertaken in this field. Xanthakis et al. in [8] is presented the first work applying GA to test data generation. In this work GA are employed to test data generation for structures not covered by random search. A path is chosen by the user, and the relevant branch predicates are extracted from the program. The GA is then used to find input data that satisfies all branch predicates at once, with the fitness function summing branch distance values. Pei et. al. [9] presented a new approach focuses on pathwise test data generation. Where the basic operations of pathwise software testing consist of there steps: program CFG construction, path selection, and test data generation and dynamic program execution. This approach manually selects the set of paths limited to 2 loops. The overall suitability by the chromosome, that is the matching degree between the path of practical execution and the ideal required path they set, is termed its fitness. The value of fitness function of a chromosome reflects the path of the program executing on the input values of all variables represented by the chromosome how good it complies with the user selected path.

James andrews et.al have discussed Nighthawk, a system which uses GA to find parameters used for randomized unit testing. Feature subset selection tool is used to access the size and content of the representations which is helpful in reducing the size of representations. This GA achieves 100% of results in only 10% of time [10]. Vivek Kothari, Satish Chandra discussed a modification to the artificial bee colony (ABC) algorithm which reduces its variations by applying genetic operators to the ABC algorithm. In this crossover phase is used to provide better solutions as it helps solutions to persist in the population [11]. Soma Sekhara Babu Lama et.al discussed generation of feasible independent paths. ABC algorithm is used for test data generation where parallel behavior of the bees makes test data generation efficient and faster and path is selected based on the priority of all edge coverage criteria. This technique helps to solve local optima problem [12]. Sanjay Singla et.al presents a technique which is based on GA and particle swarm optimization (PSO) algorithm that is used to automatically test data generation for data flow coverage. A number of programs of different size and complexity are used to analyze performance which shows its coverage ratio is more [13]. Tai-hoon Ki et.al have discussed the application of GA in software testing. This algorithm works on CFG. Assigning weights to edges of CFG, distribution of weights, fitness value is calculated, probability is calculate,; crossover is done and mutation is done. In this GA outperforms the exhaustive search and local search techniques by examining the most critical paths first a more effective way to approach testing is obtained which in turn helps to refine effort and cost estimation in the testing phase[14]. Sapna Varshney et.al proposes a novel approach based on GA to test data generation for a program. Its performance is evaluated based on data flow dependencies of a program by comparing with random testing. Based on the experimental results on a number of C programs, it shows that the proposed approach outperforms random testing in test data generation and optimization [15]. Bueno and Jino proposed an approach that utilizes control and data flow dynamic information. The proposed approach is meant to fulfil path coverage testing. In addition, it also tackles the identification of potentially infeasible program paths by monitoring the progress of the search for required test data. The approach considers a continual population's best fitness improvement as an indication that a feasible path is covered. On the other hand, attempts to generate test data for infeasible paths result, invariably, in a persistent lack of progress [16].

5. Test data generation using MGA

An initial population of individuals, each represented by a randomly generated genotype, is created. The MGA starts to evolve good solutions from this initial population. The three basic genetic operators: selection, crossover and mutation carry out the search. Genotypic strings can be thought of as points in a search space of possible solutions. They specify a phenotype which can be assigned a fitness value. Fitness value affects selection and in this way guides the search.

Test data execution: The resulting set of test cases produced by the MGA needs to be executed in order to compute their fitness and also to check the correctness of the results.

Test coverage computation: The analysis of the program under test is performed manually for the time being, and all the information is stored in files for later use by the MGA. Although this may sound very time consuming, it does not actually affect the execution time of the MGA, since this process takes place only once and prior to the execution of the algorithm.

Program under test: A program for which the test data is to be generated is considered.

5.1. Population Initialization

In this paper, population size 20 was considered. The size of each chromosome of this population was n*m. n is the number of paths in the program under test and m is number of input variables program under test.

5.2. Fitness Function

In this way, we assume that the number of all paths in the related program is known and define the fitness function in equation (1) below, which is normalized to values between $0\ to\ 1$.

$$f = \frac{k}{n} \qquad (1)$$

In equation (1), f is the evaluation function, k is the number of the paths that chromosome has covered; n is the number of all paths in the corresponding programs. The individual who achieves an evaluation value $f\ = 1$ means that the individual has covered all paths in the corresponding programs, and it is the optimal solution to the problems.

5.3. Selection

After computing the fitness of each chromosome in the current population, MGA uses the tournament method to select chromosome from members of the current population that will be parents of the new population.

5.4. Crossover

In this paper, MGA uses the uniforms crossover and crossover rata (pc) for each chromosome according to equation (2), updated at each round of execution.

$$pc_{chromosome(i)} = \begin{cases} .50 + f_{avg} * (1 - f_i) + (f_{max} - f_i) & f_i < f_{avg} \\ .50 + f_{avg} * (1 - f_i) & f_i \geq f_{avg} \end{cases} \qquad (2)$$

In equation (2), $favg$ is the average fitness of chromosomes, $fmax$ maximum the fitness of chromosomes, and fi is the fitness of chromosome i. if chromosome Ai and chromosome Bi are selected for crossover operations, then

$$pc\ =\ max\ (pc\_chromosome\ (Ai)\ ,pc\_chromosome(Bi)).$$

5.5. Mutation

Mutation introduces slight changes into a small proportion of the population and is representative of an evolutionary step. In this paper mutation rata (pm) for each chromosome according to equation (3), updated at each round of execution.

$pm_{chromosome(i)} =$

$$\begin{cases} f_{max} * (1 - f_i) & f_i < f_{avg} \\ f_{max} * (1 - f_i) * f_{avg} & f_i \geq f_{avg} \end{cases} \quad (3)$$

5.6. Stopping condition

Termination conditions in MGA specify the stopping criteria after the desired solution is obtained in few numbers of iterations. Termination condition in MGA can occur because of the following reasons:

- Finite number of generation (in this paper is 50000)
- Optimized solution is obtained.

6. Experimental Setup

Taken an example of triangle classification for implementing of given proposed method. This program is one of the most important programs that other previous works have used for the test [17] [18] [19]. The code for this program is in Python programming language below:

```
print("Input lengths of the triangle sides: ")
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))
if x == y == z:
print("Equilateral triangle")
elif x==y or y==z or z==x:
print("isosceles triangle")
else:
print("Scalene triangle")
```

CFG for this program is given in Figure2. Some evaluation criterions to test the effectiveness of MGA than GA are listed as follows (each algorithm is executed 50 times):

Time: recorded the start and end times for find test data in each execution for each algorithm, and calculated the search time using equation (4).

$$search\ time = end\ time - start\ time. \quad (4)$$

Figure 3 shows the search time for test data generating to satisfy the paths in each execution for each algorithm. Figure 4 shows the average search time for these algorithms.

Number of generations: recorded the number of generations for find test data in each execution for each algorithm.

Figure 5 shows the number of generations for test data generating to satisfy the paths in each execution for each algorithm. Figure 6 shows the average number of generations for these algorithms.

In these experiments, for the GA, the crossover rate .90, mutation rate .15, and population size 20 was considered [20].

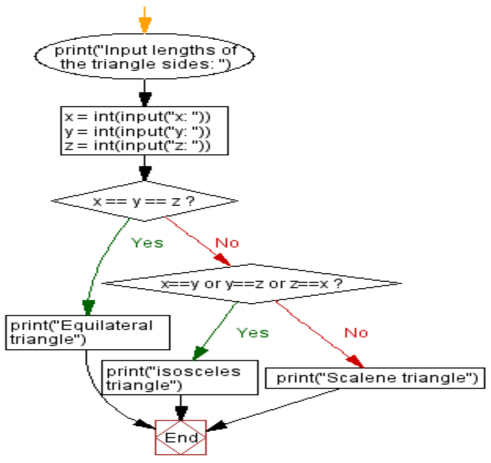
print("Input lengths of the triangle sides: ")
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))
x == y == z ?
Yes
No
x==y or y==z or z==x ?
print("Equilateral triangle")
Yes
No
print("isosceles triangle")
print("Scalene triangle")
End

Figure 2. CFG triangle classification.

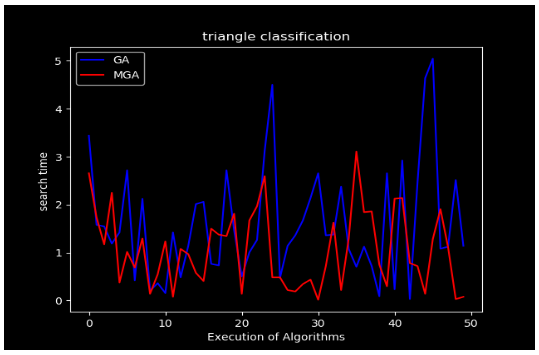

Figure 3 search time.

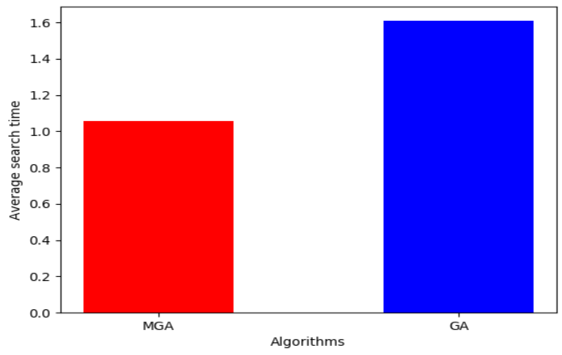

Figure 4 average search time.

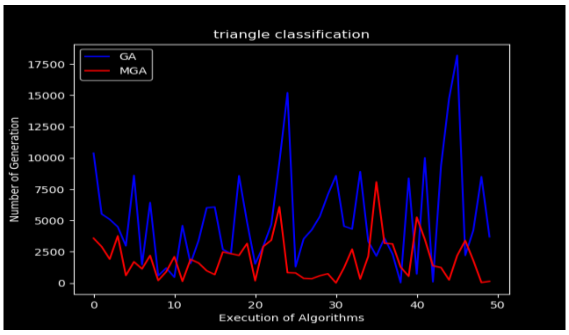

Figure 5 number of generations.

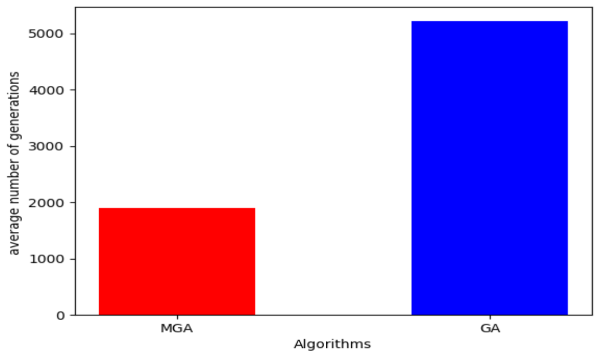

Figure 6 average number of generations.

7. Conclusion

This paper mainly introduces how to apply the genetic algorithm to get the method and technology of software structural test data generation on the basis of the path coverage. This article uses a modified genetic algorithm that calculates the rate of crossover and mutation during the execution of the algorithm. The results suggest that this method is superior to the traditional GA. For future

work, we intend to compare the results of this method with the results of other algorithms such as particle swarm optimization algorithm, bees algorithm, ant colony optimization algorithm, simulated annealing algorithm, hill climbing algorithm and tabu search algorithm.

References

1. [1] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.
2. [2] Lonetti, Francesca, and Eda Marchetti. "Emerging software testing technologies." Advances in Computers. Vol. 108. Elsevier, 2018. 91-143.
3. [3] Holland, J. H. "Adaptation in Natural and Artificial Systems, the University of Michigan Press, Ann Arbor, MI. 1975." (1975).
4. [4] Brownlee, Jason. Clever algorithms: nature-inspired programming recipes. Jason Brownlee, 2011.
5. [5] Ammann, Paul, and Jeff Offutt. Introduction to software testing. Cambridge University Press, 2016.
6. [6] McMinn P. Search-based Software Test Data Generation: A Survey. Journal of Software Testing Verification and Reliability, vol.14, no.2, pp.105-156, June 2004.
7. [7] Mantere T, Alander J T. Evolutionary Software Engineering, A Review. Journal of Applied Soft Computing, vol.5, pp.315-331, 2005.
8. [8] Xanthakis S, Ellis C, Skourlas C, Le Gall A, Kastiskas S, Karapoulios K. Application of genetic algorithms to software testing (Application des algorithms génétiques au test des logiciels). In 5th International Conference on Software Engineering and its Applications, pages 625-636, Toulouse, France, 1992.
9. [9] Pei M,. Goodman E D, Gao Z, Zhong K. Automated Software Test Data Generation Using A Genetic Algorithm. Technical Report GARAGe of Michigan State University June 1994.
10. [10] James H. Andrews, Tim Menzies and Felix C.H. Li, "Genetic Algorithms for Randomized Unit Testing", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, February, 2011.
11. [11] Vivek Kothari, Satish Chandra, "The Application of Genetic Operators in the Artificial Bee Colony Algorithm", IEEE International Conference on Recent Advances and Innovations in Engineering, May, 2014.
12. [12] Soma Sekhara Babu Lama, M L Hari Prasad Rajub, Uday Kiran Mb, Swaraj Chb, Praveen Ranjan Srivastavb, a*, "Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony", International Conference on Communication Technology and System Design, 2011.
13. [13] Sanjay Singla, Dharminder Kumar, H M Rai and Priti Singla1, "A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts", International Journal of Advanced Science and Technology Vol. 37, December, 2011.
14. [14] Praveen Ranjan Srivastava and Tai-hoon Kim, "Developing optimization algorithm using artificial bee colony system" "International Journal of Software Engineering and Its Applications" October 2011.
15. [15] Sapna Varshney, Monica Mehrotra, " Automated software test data generation for data flow dependencies using genetic algorithm", IJARCSE, February,2014
16. [16] Bueno PMS, Jino M. Identification of potentially infeasible program paths by monitoring the search for test data. In: Proceedings of the fifteenth IEEE international conference on automated software engineering (ASE '00), Grenoble, France; 11–15 September 2000. p. 209–18.
17. [17] Zhang, N., Wu, B., & Bao, X. (2015). Automatic generation of test cases based on multi-population genetic algorithm.

Int. J. Multimedia Ubiquitous Eng., 10(6), 113-122.
18. [18] Bao, Xiaoan, et al. "Path-oriented test cases generation based adaptive genetic algorithm." PloS one 12.11 (2017).
19. [19] Mann, Mukesh, Pradeep Tomar, and Om Prakash Sangwan. "Test Data Generation Using Optimization Algorithm: An Empirical Evaluation." Soft Computing: Theories and Applications. Springer, Singapore, 2018. 679-686.
20. [20] Ghiduk, Ahmed S., Mary Jean Harrold, and Moheb R. Girgis. "Using genetic algorithms to aid test-data generation for data-flow coverage." 14th Asia-Pacific Software Engineering Conference (APSEC'07). IEEE, 2007.