GENETIC ALGORITHM IN SOFTWARE TESTING OPTIMIZATION: A REVIEW

Olga Ristić[1], PhD; Marjan Milošević[1], PhD; Maja Radović[1], BSc; Vlade Urošević, PhD;
[1] University of Kragujevac, Faculty of Technical Science, Čačak, SERBIA, olga.ristic@ftn.kg.ac.rs
marjan.milosevic@ftn.kg.ac.rs , maja.radovic@ftn.kg.ac.rs , vlade.urosevic@ftn.kg.ac.rs

Abstract: *The technological progress and software complexity makes difficult to maintain and improve the quality of software. Software testing is one of the main phases in software engineering. There are many testing methods which make the software free from error and provide the complete functional capabilities. Software testing requires a lot of time, work and money, and depends mainly on test case generation, execution and evaluation. Automated testing is a technique used to maximize test coverage, detect more errors, increase test execution, decrease cost as well as improving the quality of software. The use of genetic algorithm can help to improve the test cases. This paper proposed an algorithm which can be applied to both black box and white box testing to get some of the best test cases. The purpose is to implement the genetic algorithms to reduce: the test cases, cost, time and effort to give good quality software.*

Keywords: *software testing, genetic algorithm, test case.*

1. INTRODUCTION

Software systems are becoming more complicated consequently and defects affecting the system components are expanding. It takes up time, employees and increase costs. Software testing takes almost 50% of the cost and time of the development complex software system [1]. Testing is a process of discovering errors. The aim of testing is to eliminate the gap between actual output and expected output. There are various types of software testing techniques. Broadly, testing techniques include functional (black box) and structural (white box) testing. Functional testing is based on functional requirements whereas structural testing is done on the code itself [2]. White box testing in which consider the internal structure of the software. In white box testing, path testing considers control flow and tests each individual path of the control flow graph. Data flow testing, analyze decision to path graph. Data flow testing uses a sequence of variable access to select a path from a control graph. Mainly, these two kind of white box techniques are considered in many scientific papers [3, 4].
In Black box testing, the actual implementation of software is not analyzed. Testing can be done either manually or automatically by using testing tools. It is found that automated software testing is better than manual testing. However, very few test data generation tools are commercially available today. Various techniques have been proposed for generating test data or test cases automatically. Recently, a lot of work is being done for test cases generation using soft computing techniques like fuzzy logic, neural networks, genetic algorithm (GA), genetic programming and evolutionary computation providing keys to the problem areas of software testing [5].
This paper presents an overview of the application of GA in software testing. In one simple example, all GA phases in software testing are displayed. In example is define the maximum value of a given fitness function of several randomly generated values. Using the GA phases is determined by a simple process and the maximum value in each next generation is determined. GA generates an almost optimal number of test cases to perform effective testing.

2. GENETIC ALGORITHM (GA)

GA is a type of evolutionary algorithm and it is a type of meta-heuristic search technique developed by John Holland and works on Darwin's principle of survival of the fittest. GA uses the technique of natural genetics, representing a computer model of biological evolution. GA has the ability to solve a variety of optimization and search problems. Several testing techniques use GA believing that testing may be carried out in a better way using the natural evolutionary process present in them. GA identifies an optimal solution for a problem by applying natural evolutionary techniques to a group of possible solutions referred to as "population" [6]. After each generation, a new generation is formed which is better than the previous generation.

A string of digits called chromosomes are present and each individual of the string is called a gene. Each individual in the population has a fitness value which decides the quality and performance of that individual. If the fitness value is greater, problem-solving will be better. Collection of chromosomes creates a population. The initial population is created randomly and the fitness of the individuals in the population is calculated.
The series of steps involved in GA are (figure 1):

- population initialization,
- selection,
- crossover,
- mutation and
- termination.

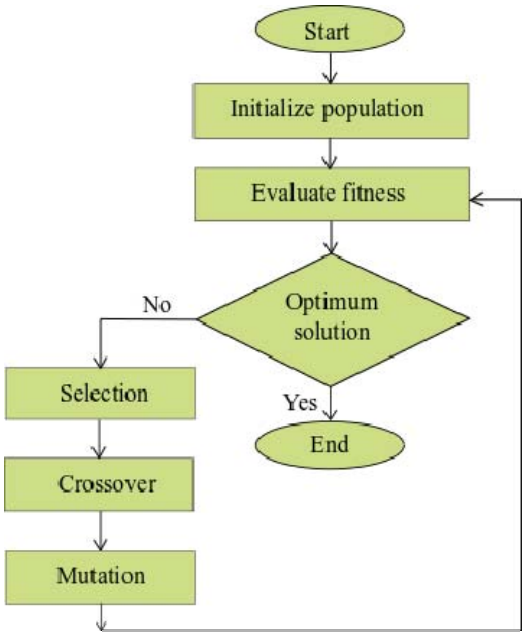

Figure 1: Flow chart of GA phases

The ***selection*** process determines which solutions are allowed to reproduce and which ones deserve to die out. The selection process involves identifying good solutions in a population, making multiple copies of the good solutions and eliminating bad solutions from the population. The solutions are evaluating in the final stage by applying the fitness function. A fitness function represents the quality of the solution.
The ***crossover*** operator is used to create new solutions from the existing solutions by recombination of the information from both parents. Crossover enables the selection of good features from parents to produce the new generation. The genes exchange randomly between the chromosomes of two parents and create new offspring. Crossover can either be 1-point, 2-points, ..., n-points. Figure 2 shows an example of two different crossover operators on two individuals, marked in blue and red [7]. On the left-hand side of figure 2, only one split is performed. In this case, both children contain two original genes come from one parent each. The example on the right side shows that the crossover operator splits both individuals' n times to generate two child individuals. So, three splits are performed, representing an n-point crossover for n = 3. In our example, both techniques generate two children to keep the population size stable. In this case, parents and children are part of the population.

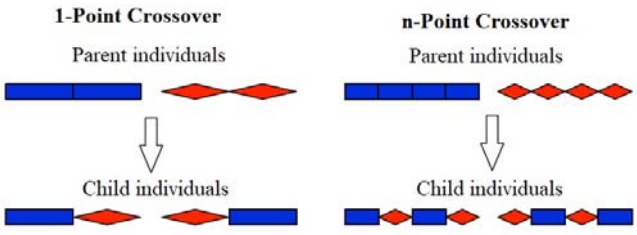

Figure 2: Crossover operator examples

The next genetic operator is a ***mutation*** which introduces new features to a given population or problem to obtain and maintain diversity in the population. Mutation is applied to the offspring to create better quality individuals. Mutation is

defined as the process of altering the genes in the chromosome. A new generation is chosen from the offspring based on the fitness of the individuals. These individuals are considered as parents of the next generation.
The various types of mutation are Uniform mutation, Inversion mutation, Reciprocal mutation, as presented in figure 3 [7]. These examples show for potential mutation operators based on the individuals retrieved by the n-point crossover in figure 2. The example shows how an individual is mutated in three different ways separately. First, one blue gene is replaced by a green gene and that is new information set into the population to keep constant size of individuals. In the second example, two genes switch the positions. Such a mutation is important when the order of genes is vital for an individuals' fitness. The third mutation extends the number of genes using insertion. On the right-hand side of figure 3 a green gene is added. Similar to uniform mutation, this adds new information into the gene code, but can also change the size of individuals, e.g., in selection tasks. Once the individuals have been mutated, one iteration of the GA cycle is complete. Then the next iteration is initiated, starting with the fitness evaluation of individuals. The GA cycle is repeated until a global solution for the problem is obtained (figure 1).

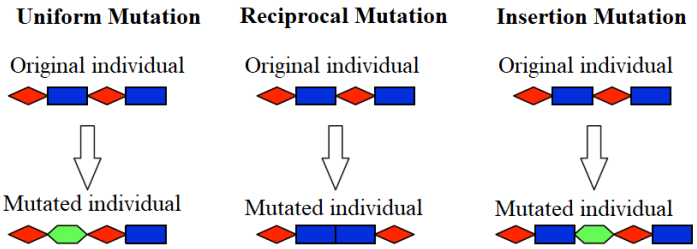

Figure 3: Mutation operator examples

3. SOFTWARE TESTING ACTIVITIES

Software testing is the process of executing a program with the intention of finding bugs. Software testing consumes major resource in term of effort, time in software product's lifecycle. Test cases and test data generation is the key problem in software testing and as well as its automation improves the efficiency and effectiveness and lowers the high cost of software testing. Generation of test data using random, symbolic and dynamic approach is not enough to generate the optimal amount of test data.
The application of GA in software testing appears in research papers [8, 9] that bring new ideas across two domains. GA is used to generate test cases with maximization the test coverage. By examining the most critical paths first, obtain an effective way to approach testing which in turn helps to refine effort and cost estimation in the testing phase [3, 4].
Software testing is a complex process that involves the implementation of black box and white box testing.

3.1 Application of GA in White Box Testing

White Box Testing (WBT) techniques or structural testing checks program code. The types of white box testing techniques are:

- control flow graphs (CFG),
- statement testing,
- branch testing,
- path coverage testing,
- data flow testing,
- loop testing.

A weighted CFG can be used. The number of paths to be tested in the program is directly proportional to the number of loops or branches. So, a large number of paths need to be checked if the program is complex. Hence, this becomes an NP-Complete program. The CFG will handle the entire independent path for new conditions and the statement. All these routes must be travelled at least once. We can initialize different weights to the critical section of the program. More weights will be assigned to the edges which are critical. The fitness function is then evaluated for the paths. They define the relative contribution of the path. The path with loops and branches will be given priority over the sequential order. The crossover will be performed according to the crossover probability. Then, the mutation will occur bit-by-bit. Every bit has equal liability to mutate. During testing, we need to consider the following criteria. They are:

1. All the independent path needs to be covered
2. Testing time should be less than the determined time.

In software testing, generating test case is more important than applying efficient testing techniques. Manual testing will consume more time. Hence, this optimized GA can be used to generate suitable data were covered every independent path in the control flow graph [10, 11].

3.2 Application of GA in Black Box Testing

Black box testing is based on user specifications, so all the program functionalities are tested. We can easily perform functional and regression testing in the program using the GA. In functional testing, we initially develop difference between good test case and a bad one. This is done using a GA. We will take a set of all the test cases, and then we will apply the algorithm. While developing the fitness function, we will separate them. Fitness function for a test case will be assigned according to the priority and importance of that function in that program. If that function is an essential one, then it cannot be removed and then will be given a higher fitness value rather than a function whose existence is optional. All the good test cases will act as the required set of test cases, and all the other test cases would be eliminated. Now, the number of iterations would be performed to generate these test cases with the help of mutation. This will help in finding the test cases which can expose the errors in the program.

4. EXAMPLE OF GA IN TEST GENERATION

Here, with the simple example, we will show usage of GA for calculation of maximum value of function $F(x) = x^2$ in a randomly generated integer value in interval from 0 to 15.

$F(x)$ - is fitness function for calculated numbers.

In this example, x can be represented in the form of 4 bits, since a binary string of 4 bits can be represented with a maximum of 16 different numbers, and GA functions correctly with a binary representation of the generated numbers [12]. First, four binary numbers are generated, for example:

1000, 1001, 0011, 0101.

In order to calculate the fitness function, we first convert the binary records of numbers into decoded records:

1000→8, 0011→3, 1001→9, 0101→5.

By the given function, which calculates the square of a given number, it gets:

8→64, 3→9, 9→81, 5→25.

The display of the generated values, the fitness function, and the probability of selection are given in Table 1.

Table 1: Demonstration of the application of GA in tests for number generation

Number	Binary value	Decimal number N	Fitness value	Probability of selection p(i)	Expected number (N*p(i))
1.	1000	8	64	0,36	1,44
2.	0011	3	9	0,05	0,20
3.	1001	9	81	0,453	1,80
4.	0101	5	25	0,14	0,64
Sum			179	1,00	3,98
Average number			44,75	0,25	0,99
Maximum value			81	0,45	1,80

Based on Table 1, it can be concluded that the number 9 has the highest probability of choice because the value is maximal. Based on the given table, the range of the fitness function solution can be formed and shown in Table 2.

Table 2: Review of the probability range for generated numbers

Binary value	Probability of selection p(i)	Boundary values
1000	0,36	0...0,36
0011	0,05	0,36...0,41
1001	0,45	0,41...0,86
0101	0,14	0,86...1,00

By generating four uniform numbers for probability values in the range 0 to 1, we select numbers for a new generation after considering which range they can belong to (Table 3).

The random number generator determines a pair of strings to be paired. For the first pair of strings, for example, a change in the fourth digit that will cause the following:

1000 ⇨ 1001
0011 0010

For the other pair of strings, the first two digits will be replaced and new offspring will be obtained:

1001 ⇨ 0101
0101 ⇨ 1001

In the next iteration, the fitness functions for a given population are again calculated (Table 4).
It can be concluded that the overall fitness function ranges from 179 to 191 for the given example. In order to find the optimal solution, the previous procedure is repeated several times. In these two iterations, the algorithm found the value x = 9 as one of the possible optimal solutions. In the first iteration, the value of 9 was in position 3, and in the second generation, the maximum values are in positions 1 and 4.

Table 3: Selection of randomly generated values for specified intervals

Random generated probability value	Border values	New population
0,20	0...0,36	1000
0,40	0,36...0,41	0011
0,60	0,41...0,86	1001
0,95	0,86...1,00	0101

Table 4: Calculation of the fitness function for new offspring

Number	New population	Decimal number N	Fitness value	Probability of selection p(i)	Expected number (N*p(i))
1.	1001	9	81	0,425	1,70
2.	0010	2	4	0,020	0,08
3.	0101	5	25	0,130	0,52
4.	1001	9	81	0,425	1,70
Sum			191	1,00	4,00
Average number			47,75	0,25	1,00
Maximum value			81	0,425	1,70

5. CONCLUSION

The GA is an evolutionary algorithm that optimizes the software testing technique by iterating through the five phases. They are inspired by evolutionary biology. These five phases can be used in software testing to get some of the best test cases rather than testing the software as a whole. It reduces the time taken and can also be applied to the black box and white box testing efficiently to get the almost optimal results. The scope of GA cannot be restricted to the domain of software engineering. It can be applied to computer science which includes the operating system and computer networks. Networking integrated with AI can yield better results for the transferring of data by selecting the best path. Software engineering is an emerging field which has evolved over time. New searching techniques can also be established to find the best test cases rather than applying all the test cases or random test case.
This paper demonstrates that genetic algorithms (GA) can be used as a tool to help a software tester search, find, and isolate failures in a software system. The use of GA supports automated testing and helps to identify failures that are most severe and occur for the user. The strategy presented in this paper relies on a technique that helps the developer with more information concerning the faults in the software.
The simple GA approach presented in this paper provides in itself a useful contribution to the selection of test cases and a focused examination of test results. Thus, application of this approach can support reasoning about test results to support quality system assessment and/or debugging activities.

ACKNOWLEDGMENT

The work presented in this paper was funded by grant III46001 and III47003 for the period 2011-2019 by the Ministry of Education, Science, and Technological Development of the Republic of Serbia.

REFERENCES

[1] Surendran A., Samuel P. Evolution or revolution: the critical need in genetic algorithm based testing, Artificial Intelligence Review, 2017, Volume 48, Issue 3: 349–395.

[2] Kudjo P. K., Nii Noi Ocquaye E., Ametepe W. Review of Genetic Algorithm and Application in Software Testing, International Journal of Computer Applications, Volume 160, No 2, February 2017: 1-6.
[3] Bao X., Xiong Z., Zhang N., Qian J., Wu B., ZhangW. Path-oriented test cases generation based adaptive genetic algorithm, PLOS ONE, 12(11), 2017: https://doi.org/10.1371/journal.pone.0187471
[4] Rijwan, K., Amjad, M. Automatic Generation of Test Cases for Data Flow Test Paths Using K-Means Clustering and Generic Algorithm, International Journal of Applied Engineering Research, 11(1), 2016: 473-478.
[5] Mishra D. B., Bilgaiyan S., Mishra R., Abhinna Acharya A. & Mishra S. Review of Random Test Case Generation using Genetic Algorithm, Indian Journal of Science and Technology, Volume 10, Issue 30, 2017: 1-7.
[6] Khari M., Kumar P. An extensive evaluation of search-based software testing: a review, Springer-Verlag GmbH Germany, Soft Computing, 2017: 1–14.
[7] Lachmann R. Black-Box Test Case Selection and Prioritization for Software Variants and Versions. PhD thesis, 2017.
[8] Dasoriya R., Dashoriya R. Use of Optimized Genetic Algorithm for Software Testing, 2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science, 2018: 5 p.
[9] Haraty R. A., Mansour N., Zeitunlian H.. Metaheuristic Algorithm for State-Based Software Testing, Applied Artificial Intelligence, 32(2), 2018: 197-213, DOI: 10.1080/08839514.2018.1451222
[10] Mishra, D. B., Mishra, R., Acharya, A. A., Das, K. N.: Test Data Generation for Mutation Testing Using Genetic Algorithm, Soft Computing for Problem Solving, 2018: 857–867.
[11] Moshizi M. M., Bardsiri A. K. The Application of MetaHeuristic Algorithms in Automatic Software Test Case Generation, International Journal of Mathematical Sciences and Computing (IJMSC), 1(3), 2015: 1-8.
[12] Kumar M., Nandal D. A Review on Analysis Search Based Software Testing using Metaheuristics Techniques, International Journal of Computer Sciences and Engineering 6.10, 2018: 491-497.