Indian Journal of Science and Technology, *Vol 10(30), DOI: 10.17485/ijst/2017/v10i30/107654, August 2017*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Review of Random Test Case Generation using Genetic Algorithm

Deepti Bala Mishra[1], Saurabh Bilgaiyan[1], Rajashree Mishra[2], Arup Abhinna Acharya[1] and Samaresh Mishra[1]

[1]School of Computer Engineering, KIIT University, Bhubaneswar – 751024, Odisha, India; dbm2980@gmail.com, saurabhbilgaiyan01@gmail.com, aacharyafcs@kiit.ac.in, smishrafcs@kiit.ac.in
[2]School of Applied Sciences, KIIT University, Bhubaneswar – 751024, Odisha, India; rmishrafma@kiit.ac.in

Abstract

Background/Objectives: This research paper presents how Genetic algorithm is efficiently used in random test case generation during functional software testing. **Methods/Statistical Analysis:** Different hybridized Genetic Algorithms are used to generate test data automatically and optimized those test cases to solve many complex problem related to software testing. **Findings:** Genetic Algorithms are successfully used in software testing with increasing number of test case generation and provides a means of an automatic test case generator. **Applications/Improvements:** This study gives us a brief idea to implement Genetic Algorithms in software testing for optimum results and also it can be used with the neural networks and fuzzy systems for different types of testing to improve the performance.

Keywords: Black-Box Testing, Fitness Function, Genetic Algorithm (GA), Neural Network, Software Testing, Test case Generation

1. Introduction

The quality of software can be highly improved by testing the software so that it can satisfy the specific requirements of the customer. Effective test cases, prioritization and optimization of test cases are some issues in software testing, which is to be solved. For this purpose Cost, effort and time of the software testing are very important factors. Software industry suffers with a heavy loss of $500 billion due to reduced in software quality. So for high quality software testing is required, which is a process to test the runtime quality and quantity of the software and the tested software can only meet the specific requirements of the user[1]. In Software Development Life Cycle(SDLC), testing is a very important and expensive task. Testing is done after designing and coding the software, shown in Figure 1.

Basically testing techniques are of two types functional testing and structural testing. Functional requirements are needed in functional testing, which is known as black box testing and structural testing is based on internal coding, known as white box testing. The combination of the above two basic techniques is known as Gray box testing[2]. Software failure causes by different faults and those faults can be detected by software testing, so that the software will work properly.

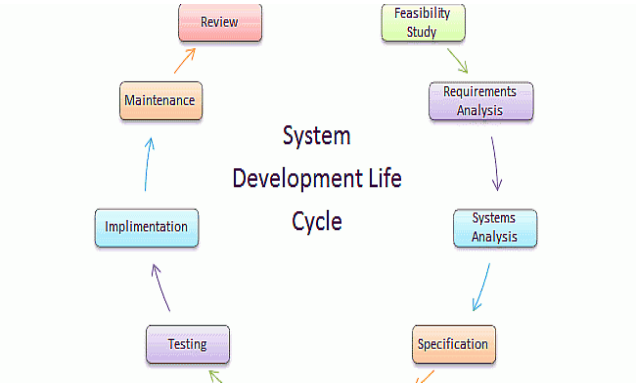

Figure 1. Phases of SDLC.

** Author for correspondence*

Software testing is the process of verification and validation to make sure that the software covers all the desired business and technical needs[3, 4]. It is a very expensive process as it needs about 50% cost of total software life cycle. Different testing tools can be used for manual or automatic testing. Manual software testing suffers from the drawbacks such as operation speed, high investment of cost, more time consuming, resources unavailability, inefficient and inaccurate test checking. These drawbacks can be overcome by automated the testing process[5, 6]. During software testing a maximum of 50% resources are consumed of the total software development resources. If the testing is performed using automated testing then it will lead to reduce in software development cost by a significant margin[7-9].

A test case is a data which acts as input for the software testing and it consists of unique identifier, requirement references from a software specification, a series of steps, events, preconditions, input, output, expected result and actual result[10,11]. Genetic algorithm, neural networks, fuzzy logic, etc. are some of the software techniques which are applied successfully in different engineering and applied sciences field. Among all evolutionary search techniques, Genetic Algorithm can give an optimal problem to the task of test case generation and solution can easily found[12-14]. In the field of software engineering many optimization problems has been solved by applying Genetic Algorithm as parallelism and search space operations are the important characteristics[11,15,16] and for testing purpose this can be used to generate test plans automatically.

This paper presents a survey of how GA is efficiently used to generate test cases for software testing. Further the paper is partitioned into 4 sections. Section 1 presents the Introduction, section 2 contains a brief description about Genetic Algorithm and the working flow of GA, section 3 presents related work in the field of test case generation using Genetic Algorithm for functional testing, and section 4, gives a short conclusion followed by our future work.

2. Genetic Algorithm

Genetic algorithm is a natural genetic based search technique, which is developed by John Holland in 1970. It is used to solve many complex and real life problems by applying function optimization methods. In software testing so many problems related to test cases are solved by producing high quality test data automatically[17-19].

GA has emerged as a practical, robust optimization technique and search method and it is inspired by the way nature evolves species using natural selection of the fittest individuals. It is a best way to solve a set of problems with less information[11, 20].

The solution to a specific problem can be solved by a population of chromosomes, the strings of binary digits and each digit is called a gene and the population can be created randomly.

Basically four different types of operators are used in the process of GA such as selection, crossover, and mutation and elitism (optional) [11, 18].

Selection: The use of selection operator is to select the best parents for performing other GA operations. Usually the selection is done on the basis of fitness value of the individuals, which is obtained from the fitness function. Fitness function can be defined as a specific function depending upon the criteria which returns a number indicating the acceptability of the program. This function is used in the selection process to determine the optimum point and the variants survive to the next iteration[21,22]. Selection methods are of six different types such as roulette wheel, stochastic universal sampling, linear rank, exponential rank, binary tournament and truncation.

Crossover: After selecting the better individuals by using selection operator, the crossover is applied to the chromosomes. In this operation two individuals swaps genes or sequence of bits between them when they satisfy the probability factor of the operator. For binary encoding different types of crossover operators are used like one point, two point, uniform and arithmetic. Mutation is performed after crossover if the mutation probability is true for the given iteration.

Mutation: It is used to maintained genetic diversity in the population by altering chromosomes to introduce new good traits. Basically six types of mutation operators are used in Genetic algorithm such as Bit string, flip bit, boundary, uniform, non uniform and Gaussian.

Elitism: Elitism process involves copying a small proportion of the fittest candidates into the next generation, which are related to the best solution found.

2.1 Process of Genetic Algorithm

The basic process of Genetic algorithms mainly involves creating an initial set of random solutions (population)

and evaluating them[3,8,11,12], by using the GA operators shown in Figure 2. After the better solutions are identified (parents) they are again used to generate new solutions (children). These values can be used to replace with other population. This new generation (population) is then reevaluated and the process for generating new values continues until a final solution is found based on a specified condition of the fitness function[15, 23]. Finally, the optimum data is obtained by function optimization technique[23].

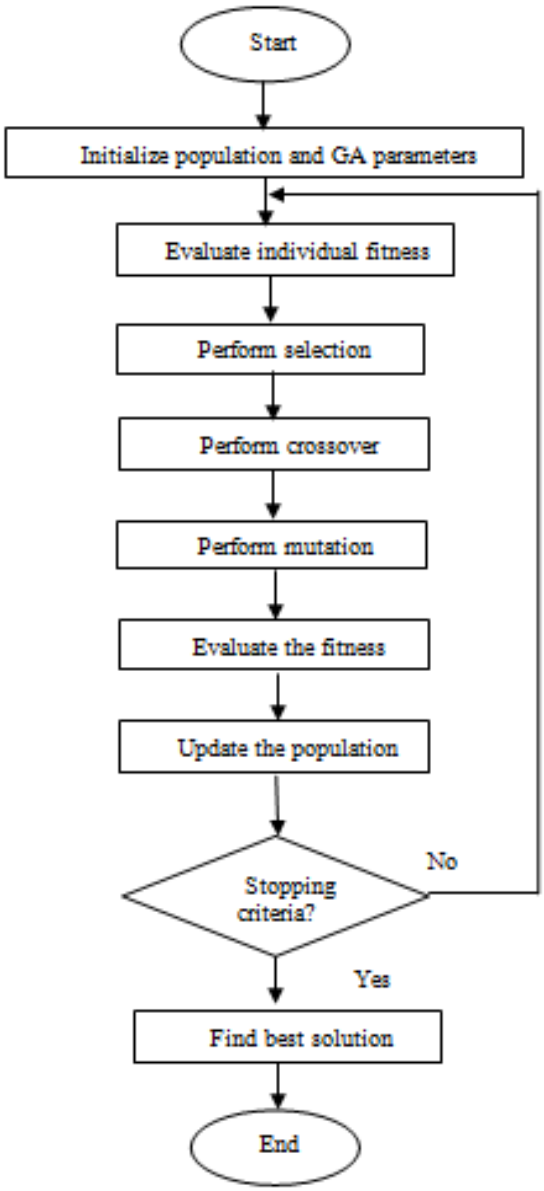

Figure 2. Flow of Genetic Algorithm.

2.2 Generating Random Test Cases by using Dev-C++ code:

```
int main( )
{
        clock_t c;
        time_t t0;
        char *s;
        c=clock( );
        t0=time(NULL);
        s=ctime(&t0);
        cout<< " current time=" <<ctime(&t0)<< endl;
        unsigned seed =time(NULL);
        srand(seed);
        int i,j,k;
        double array_1[120][25][20];
                for(i=1;i<100;i++)
        {
  for (j=1;j<5;j++)
        {
                for (k=0;k<4;k++)
                {
                        array_1[i][j][k]=rand()%2;

                }
                }
                }
        cout<<"Randomly generated Strings are:-
"<<endl<<endl<<"----
        ------------"<<endl;
        for(i=1;i<100;i++)
        {
  for (j=1;j<5;j++)
        {               for (k=0;k<4;k++)
                                cout<<array_1[i][j]
[k];
                }
        cout<<endl;
}
        return (0);
}
```

The above code will generate 100 numbers of string populations randomly and then the parameters such as cross over rate and mutation rates are determined in order to build an efficient function to find our chromosome.

Initially we take 100 numbers of populations for finding the optimal solution for a specific problem, we can increase the number of generation and population until the best solution is found. Using genetic algorithm, new populations are produced which will go for the cross over and mutation by defining a fitness function to a specific problem.

3. Related Works

This section provides a short description about the different hybridized GA techniques applied for software testing.

In[14], the authors proposed a hybrid fuzzy based GA with age extension (FAexGA), to generate test cases for mutation testing. They found a very minimal set of test cases. The faults in test cases are exposed by the use of mutated versions of the original method. The proposed method uses a FLC (Fuzzy Logic Controller) for obtaining the probability of crossover. The probability of crossover differs according to the age intervals allocated during lifetime. The age and lifetime of chromosomes (parents) are defined by the FLC state variables. The truth value for obtaining Young-age, Middle-age and Old-age are shown in Table 1. Where,

Age ∈ [Young-age, Middle-age, Old-age]
Crossover Probability ∈ [Low, Medium, High]

Table 1. Fuzzy Rule for cross over probability [14]

		Parent 1		
		Young	**Middle age**	**Old**
Parent 2	Young	Low	Medium	High
	Middle age	Medium	High	Medium
	Old	Low	Medium	Low

In their work an effective set of test cases are generated for a Boolean expression of 100 Boolean attributes by using three logical operators AND, OR, and NOT. An external application generates the correct expression randomly and one simple function is evaluated for each test case to generate an erroneous expression. Here 100-bit lengthen binary strings of one dimensional are generated as chromosomes and the function value is calculated by using the equation (1). They concluded that the Fuzzy based GA algorithm (FAexGA) can found error in very short time with distinct number of solutions.

$$F(T) = \begin{cases} 1, & if\ Eval_{correct} \neq Eval_{erroneous} \\ 0, & otherwise \end{cases} \quad (1)$$

In[20], the authors has shown the use of GA method to generate better test plans for functionality testing. The method is applied in an unbiased manner to avoid the expert's interference. They used the fitness function as shown in equation (2).

$$f_p = \sum_{i=1}^{k-1} t\left(l_i \rightarrow l_{i+1}\right) \quad (2)$$

Where $p = l_1, l_2, \ldots, l_k$ is a test plan or sequence of operations and t is a transition function for converting one operation l_i to the next operation l_{i+1} in a sequence. The sequence is considered as better if the fitness value is high.

The GA and neural network for the functional testing of the software under test in[24], and the authors applied the improved Genetic algorithm to the function model, created using neural network. The following fitness function, shown in equation (3) is used.

$$f = \begin{cases} \dfrac{1}{|c-g|}, & c \neq g \\ f_{max}, & |c-g| \leq 10^{-8} \end{cases} \quad (3)$$

Here c represents the actual output and g represents the required output of the software under testing. When the resultant (fitness) value of the proposed algorithm crosses the maximum point of possible outcome, then the algorithm terminates its execution and the current individual is termed as the best test inputs for the corresponding outputs. The authors found that proposed GA can generate better test cases with high efficiency.

A Genetic Algorithm based technique was developed to generate test data from Unified Modeling Language (UML) state machine diagrams in[7]. The test data can be generated and evaluated before coding by finding the total number of coverage. They have taken the sequence of triggers as a chromosome in the UML diagram and the fitness value of each test data is the number of fired transitions. They have discussed four case studies as Coffee vending Machine, Student enrollment system, Class management system and a telephone system and have done experiments their different properties with their proposed tool. In their experiments they have taken the best solutions which cover maximum number of

transitions. They found the developed system works in a very good manner for the cases where the final state is not present and their fitness function for a chromosome is shown in equation (4).

$$aW+bX+cY+Z \tag{4}$$

In equation (4) a, b and c represents constant values and if there will be no guard condition for selected transition then a=0, W, X, Y and Z denotes the number of states in test case, the recently covered transitions, the number of states reached and the number of path coverage for the test cases respectively.

In[25], authors state that GA can be applied for optimizations and improve the performance to obtain the local optimal solution of a specific problem. The applied a new algorithm called as Genetic-Particle Swarm Mixed Algorithm (GPSMA) to automatically generate software test data. The proposed technique uses the update mode in each individual to replace the mutation process in the algorithm which is based on population division. The proposed algorithm can generate and search specific test data in a domain to satisfy the test condition and it uses the Triangle Classify Program to address the problems of test data generation.

$$\partial i = \frac{T_{ji}}{\sum_{j=1}^{n} T_{ji}} \tag{5}$$

They introduced a fitness function shown in equation (5), where *i*, represents the number of generation, *j* is the sub population and ∂i is the impact factor of mutation, called as the "excellent rate of production", in the equation (4), T_{ij} is the total quantity of optimal individuals.

In[15], a method called as US-RDG was proposed to generate test cases automatically. They combined the User Session data with Request Dependence Graph (RDG) for web application and applied their approach for gray box testing. Their simulated results show that when the test suit is very small then US-RDG effects better than the traditional user session-based testing by higher path coverage and fault detection rate. They used the conception as transition relation in the form of "page → request → page". The relationship between pages and requests are presented by the transition relations. Transition relations in the specific application can be extracted from structural analysis by RDG and finally by mixing different user sessions the test cases are generated to cover maximum of transition relations. They found the performance of US-RDG is very well in test case generation for web application. They used the fitness function of a chromosome as shown in equation (6).

$$\textbf{Fitness} = (\alpha * |CDTR| + |CLTR|) / (\alpha * |DTR| + |LTR|) \tag{6}$$

Where |CDTR| and |CLTR| represents the number of data and link dependence transition relations respectively. The fitness value will become 1 when all the data and link of a specific application are covered by a chromosome and to indicate the coefficient of the data dependence transition relation, the authors introduced a factor named as α. They have taken α as much as greater than 1 to increase the proportion of the data dependence transition relation and assigned 1 to the coefficient of the link dependence in transition relation. They found the chromosome, with a bigger fitness value, covers more data dependence transition relations.

In[26], authors used a multi objective optimization in black box string test case generation for random testing and adaptive random testing. The authors examined many string distance functions and hence they introduce two objectives for effective string test cases such as the length distribution and the diversity control of the test cases within a test set. They used one diversity- based fitness function to generate optimized test sets to reveal faults more effectively and it is shown in equation (7).

$$F_D = \sum_{i=1}^{test\ set\ size} dist\left(t_i, \beta\left(t_i, test\ \ set\right)\right) \tag{7}$$

In the above function *ti* denotes the ith test case and β is its nearest test case in the test set and the summation is performed between the two distances test cases. They found the higher value of the fitness, the more diverse distribution of test cases.

After an extensive study of different testing techniques, we came to learn GA parameter is efficiently used for generating test cases in functional software testing.

Table 2 represents the value points of GA parameter, corresponding methods used and the results found in different types of software testing discussed in our work.

Table 2. The GA parameters and the advantages of Hybridized GA found in different types of Black box software testing

Authors	P_s	NOI	S_m	C_m &C_r	M_m &M_r	E_m	Results
[14]	100	200	Random	One point, 0.9	Flip, 0.01	Binary	FAexGA is efficient for finding error in a very short time with distinct solutions.
[20]	150	50	Random	Point cross over, 0.8	Flip, 0.01	Binary	Test cases show highest inconsistency of application
[24]	100	500	Roulette Wheel	-, 0.8	-, 0.15	Real	Test cases are generated with high efficiency.
[7]	10		Random	Single point, 0.5	Random, 0.5	Integer	Test data can be successfully generated from the state chart diagrams and GA works very good in the absence of final state
[25]	100	10	Roulette Wheel	Single point, 0.75	Flip, 0.02	Binary	GPSMA is a better method for test data generation and it uses the excellent rate of production to interact between sub populations.
[15]	100	20	Roulette wheel	Two point,	-, 0.25	Integer	US-RDG generate good test cases for web application.
[26]	100	200	Rank selection	Single point, 0.6	-, 0.01	Binary	MOGA produced superior failure detection performance than GA.

(P_s-Population Size, NOI-Number of Iteration, S_m-Selection method, C_r-Cross over rate, C_m-Cross over method, M_r-Mutation rate, M_m-Mutation method, E_m- Encoding method)

4. Conclusion and Future Work

In this paper, authors studied how different types of hybridized Genetic Algorithms are helped in software testing field by generating automatic test cases randomly. Genetic algorithm can also be used with the neural networks and fuzzy systems for performing different types of testing to improve the performance.

In future it is planned to design a new hybridized algorithm by taking a better fitness function which will help us to evaluate the efficiency of test cases and further we can increase the efficiency of the test result by changing the input parameters, by increasing the number of generations and obtain the optimum values for different types of software testing. It is also planned to applied hybridized GA with other soft computing techniques such as neural network for optimized test case generation in web-based application software.

5. References

1. Jones C, Bonsignour O. The economics of software quality. Addison-Wesley Professional. 2011 July; 19.
2. Acharya S, Pandya V. Bridge between Black Box and White Box–Gray Box Testing Technique. International Journal of Electronics and Computer Science Engineering. 2012; 2(1):175–85.
3. Chauhan N. Software Testing: Principles and Practices, Oxford University Press. 2010.
4. Jogersen PC. Software testing: A craftsman approach. 3rd edition, CRC presses. 2008.
5. Srivastava PR, Ramachandran V, Kumar M, Talukder G, Tiwari V, Sharma P. Generation of test data using meta heuristic approach. TENCON 2008-2008 IEEE Region 10 Conference. 2008 November; 19: 1–6.
6. Michael CC, McGraw GE, Schatz MA, Walton CC. Genetic algorithms for dynamic test data generation. Proceedings of 12th IEEE International Conference IEEE on Automated Software Engineering, 1997. 1997 November 1, p. 307-08. Crossref
7. Doungsa-ard C, Dahal K, Hossain A, Suwannasart T. Test data generation from UML state machine diagrams using gas. International Conference on Software Engineering Advances (ICSEA 2007). 2007 August 25; p. 47–47. Crossref
8. Srivastava PR, Kim TH. Application of genetic algorithm in software testing. International Journal of software Engineering and its Applications. 2009 October; 3(4):87–96.
9. Berndt DJ, Watkins A. High volume software testing using genetic algorithms. Proceedings of the 38th Annual Hawaii International Conference on System Sciences IEEE. 2005 January 3; p. 318b. Crossref
10. Dixit S, Tomar P. Automated test data generation using computational intelligence. 2015: 4th International Conference on IEEE Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions). 2015 September 2; p. 1–4. Crossref
11. Sharma A, Patani R, Aggarwal A. Software testing using genetic algorithms.

12. Ahmed MA, Ali F. Multiple-path testing for cross site scripting using genetic algorithms. Journal of Systems Architecture. 2016 March 31; 64:50–62. Crossref
13. Yang S, Man T, Xu J, Zeng F, Li K. RGA: A lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation. Information and Software Technology. 2016 August 31; 76: 19–30. Crossref
14. Last M, Eyal S. A fuzzy-based lifetime extension of genetic algorithms. Fuzzy sets and systems. 2005 January 1; 149(1): 131–47. Crossref
15. Peng X, Lu L. A new approach for session-based test case generation by GA. 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN), IEEE. 2011 May 27; p. 91–96. Crossref
16. Pinto GH, Vergilio SR. A multi-objective genetic algorithm to test data generation. 2010 22nd IEEE International Conference on Tools with Artificial Intelligence IEEE. 2010 October 27; 1: 129–34. Crossref
17. Ribeiro JC, Zenha-Rela MA, de Vega FF. Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software. Information and Software Technology. 2009 November 30; 51(11): 1534–48. Crossref
18. Goldberg DE. Genetic algorithms. Pearson Education India. 2006.
19. Wappler S, Lammermann F. Using evolutionary algorithms for the unit testing of object-oriented software. Proceedings of the 7th annual conference on Genetic and evolutionary computation ACM. 2005 June 25; p. 1053–60. Crossref
20. Emanuelle F, Menezes R, Braga M. Using Genetic algorithms for test plans for functional testing. 44th ACM SE proceeding. 2006; p. 140–5.
21. Mathur AP. Foundations of Software Testing, 2/e. Pearson Education India; 2008.
22. Rauf A, Anwar S, Jaffer MA, Shahid AA. Automated GUI test coverage analysis using GA. 2010 Seventh International Conference on IEEE Information Technology: New Generations (ITNG). 2010 April 12; p. 1057–62. Crossref
23. Andalib A, Babamir SM. A new approach for test case generation by discrete particle swarm optimization algorithm. 2014 22nd Iranian Conference on Electrical Engineering (ICEE) IEEE. 2014 May 20; p. 1180–85. Crossref
24. Zhao R, Lv S. Neural-network based test cases generation using genetic algorithm. 13th Pacific Rim International Symposium on IEEE Dependable Computing, 2007. PRDC 2007. 2007 December 17; p. 97–100. Crossref
25. Li K, Zhang Z, Kou J. Breeding software test data with genetic-particle swarm mixed algorithm. Journal of computers. 2010 January 2; 5(2): 258–65. Crossref
26. Shahbazi A, Miller J. Black-Box String Test Case Generation through a Multi-Objective Optimization. IEEE Transactions on Software Engineering. 2016 April 1; 42(4): 361–78. Crossref