# GENETIC ALGORITHMS BASED BLACK BOX TESTING

[1]Ketna Khanna, [2]Naresh Chauhan, [3]Vedpal
[1]M.Tech Scholar, [2]Professor, [3]Assistant Professor
[1] Computer Engineering Department,
[1]J. C. Bose University of Science and Technology, YMCA, Faridabad, India

*Abstract:* Software Testing is immensely important task because thorough testing improves the quality of the software and reduces its maintenance cost. This work proposes a novel way of Black Box Testing that uses Rate of Fault Detection and Genetic Algorithms (GA) for the set task. GA is known for finding near optimal solution in very large search space based on the fitness function. Here fitness function has been crafted using rate of fault detection which is a measure that calculates the detection of as many faults as possible in the least amount of time i.e. as quickly as possible. Model has been explained and the results are demonstrated. Comparison of the proposed work with the other existing approach is performed using APFD metric. The work gives encouraging results and would be extending by using various data sets and applying other optimization techniques in future.

***IndexTerms* – Black Box Testing, Genetic Algorithms, Test Case Prioritization.**

## 1. INTRODUCTION

Testing is a process that includes execution of a program with the intent of finding yet-undiscovered errors, as quickly as possible [1]. Testing is necessary to ensure the quality of the system under test (SUT). In order to develop reliable, fault free software, effective testing is required. The study of National Institute of Standards and Technology (NIST) has revealed that use of inadequate infrastructure of testing results in a higher national annual cost of about $22.2 to $59.5 billion for US economy [2]. The increased cost is due to recurrent testing performed by the developers for detecting faults and their causes. A system which is not tested properly is likely to fail thereby costing much in the process of revival. The current scenario of industry majorly includes manual testing where human experts are responsible for designing of manual test cases. The automation tools are limited to the execution of pre-planned tests [3]. In a resource constrained environment, manual testing is very tedious and time consuming. Hence there is a need for automation.

The purpose of testing is to find as much as not yet-discovered faults as possible. There are two general methods of detecting bugs of a system:

i) By reviewing the internal logic and structure of code which is commonly referred as White box testing.
ii) By executing the test cases for various inputs and comparing the computed outputs with the expected outputs which is known as Black box testing.

Here BBT is used as most of the time; developer's team is different from tester's team. So tester might not be aware of internal design and structure of code. Often developers do not share the program code with the testers and only specification is given to them. The absence of code of the system under test forces the testers to perform BBT [1].

In the proposed work, emphasis is given on implementing testing via soft computing techniques. Soft Computing is a combination of various methodologies which are capable of finding the optimal solutions to real world problems [4]. These techniques are used when the conventional techniques fail to provide solution for complex, intractable and ambiguous problems. Its common methodologies include Neural Networks (NN), Genetic Algorithms (GA) and Ant Colony Optimization (ACO) [5]. A NN is a computer system which is inspired by the biological nervous system. It can imitate the working of a human brain and can be used for performing various computational tasks more efficiently as compared to a human brain [6]. GA can be used for finding unique optimal solutions for complex problems [7]. ACO is an algorithm which is capable of finding optimized solutions for complex computational problems using graphs [8].

For performing an effective testing, a single test case might not be sufficient. Hence there is a need to consider combination of test cases [1]. For n number of test cases, the total possible combinations would be $2^n$ i.e. even for 10 test cases, the number of combinations will rise up to $2^{10}$ which is, 1024 [9]. But in a resource constrained environment, it is impossible to explore such a huge search space. GA is capable of generating optimal solutions for a problem where huge search space is involved [10]. Hence GA has been used in the proposed work. GA is adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and natural genetics [11].

In the proposed work, an approach has been presented in which a fitness function is proposed which is dependent on summation of rate of fault detection of each test case and time taken to execute the complete test suite. Chromosomes consist of combination of test cases. The fittest Chromosomes are selected on the basis of Roulette Wheel Selection and then crossover is applied to it. New offspring are generated. Mutation is applied on the offspring and these modified offspring are then added to the population. This process repeats till terminating criteria is satisfied. Otherwise the process halts and the chromosomes present in the last generation are chosen for execution purpose.

The organization of paper is as follows. Section two discusses GA, section three discusses the literature review, section four discusses proposed work, section five the presents results and analysis and section six concludes.

## 2. GENETIC ALGORITHMS

GA is a heuristic search algorithm, which can be used for finding optimal solutions for complex problems [2]. When the classical techniques fail to give good enough solutions to intractable problems, GA can be used.

### 2.1 Parameters of GA

A GA consists of four major parameters:

i) Chromosomes- Chromosomes are also referred as candidate solutions of the given problem. Collection of chromosomes is termed as population. The very first population generated i.e. start pool can be generated either randomly or by using some technique like Greedy Algorithm [2].

ii) Fitness Function- A fitness function is one which is responsible for predicting the quality of a solution [2]. It contains whole information of problem.

iii) Selection Function- It is a function which determines the chromosomes that will take part in the execution process of a GA. One of commonly used function is Roulette Wheel Selection i.e. individual having highest fitness number will be given more priority than others [12].

iv) Crossover Operator-This is used for sharing the genetic material (information) between two parents i.e. chromosomes and new offspring are generated from them [2]. There are three main types of crossovers:
a) One-point Crossover: Here a random site is chosen and then latter parts of both the parents are swapped in order to generate offspring.
b) Multi-point Crossover: In this more than one random site is selected and swapping of parents tails is done.
c) Uniform Crossover: Here each gene is treated individually. It is just like flipping a coin and it is decided that allele of which parent will be used for generating the offspring.

v) Mutation operator: This is generally used to launch diversity in the genetic population [9]. Unlike crossover, it is done on single parent at a time. It is used for eliminating pre-mature convergence. There are three methods for performing mutation
a) Bit flip Mutation: Here bit value of a chromosome is changed based on the rate of mutation.
b) Uniform Mutation: Here one bit is randomly chosen and its value is flipped.
c) Inversion Mutation: A subset of chromosomes is selected and then string is reverted back to produce a genetic defect [10].

### 2.2 Process of GA

The algorithm for process of GA is as follows:-

Step 1- Initialize the start population either randomly or by using Greedy Algorithm.
Step 2-Evaluate fitness values of chromosomes of start pool.
Step 3-While (stopping conditions are not satisfied)
{
do
{
i) Select the chromosomes using Roulette Wheel Selection method for performing modification.
ii) One-point Crossover is applied on the selected chromosomes.
iii) Perform Mutation on the offspring derived from crossover operator.
iv) Insert offspring into population.
}
}
Step 4-Random selection of test cases for execution from the last generation.

There are various conditions which are used to depict the end of the process of genetic algorithm. Most common terminating criteria are as follows [2]:

1) No further improvements in subsequent generations.
2) When the maximum number of specified generations is reached.

## 3. LITERATURE REVIEW

In this section, Literature Review of various research papers has been presented.

Mark Last et. al., focused on reducing the non-effective and increasing the number of effective test cases [2]. A novel approach Fuzzy-based Age extension of Genetic Algorithms (FAexGA) has been introduced. For calculating its effectiveness, the approach has been implemented on complex Boolean expressions and the results are promising. The above paper does not mention the process which was used for modeling test cases into chromosomes. The method for calculation of fitness value for each chromosome is also missing.

Redstone Software discussed the positives and negatives of automated Black Box and White Box testing [13]. The authors stated that although black box testing had many drawbacks in past, but the innovative approaches of black box testing make BBT a good choice.
Khan M. and Khan F., performed comparison between different strategies for testing, namely, White Box, Black Box and Grey Box along with their techniques is performed [14].

Antti Huima has stated that when test cases find more number of faults, the confidence in that system tends to decrease. An anomaly in the relationship between definition of confidence and common thinking about BBT is described [15].
Neil Walkinshaw discussed about various data mining and machine learning techniques which can be used to analyze the selection of test cases in BBT in absence of specification. Various Inference-Driven Techniques like Test-Driven Algebraic Specification Inference, Decision-Tree Based Approaches, MELBA, MINTEST and various Model-Agnostic Techniques are presented [16].
Shahbazi A. et. al. proposed a model which can generate superior test cases on the basis of measuring diversity of tree test set. A model describing distribution of tree sizes is also introduced [17]. Tree test cases produced by proposed method are proved more superior to the test cases generated by other methods. The performance of various tree test case generation methods like FCFS, ARTOO, GA and MOGA are also compared by implementing them on four real world problems where MOGA outperforms.

Frezza S.T., et. al. proposed a technique to create automated test cases on the bases of 'graph data model' [18]. Relationships between design and requirement have been captured and have been used to generate test cases. The work has been evaluated on floating point arithmetic and logical unit examples.
Flores A. and Polo M. proposed a back to back testing technique for testing the replaced components in software. The work has been implemented using testooj tool for java components [19].
Do H. and Rotherme G., have designed and performed two controlled experiments assessing the ability of prioritization techniques so as to improve the rate of fault detection of test case prioritization techniques. Results depict that prioritization can be effective relative to the faults considered [20].

Karl Meinke proposed a model that corrects the technical flaws found in existing probabilistic correctness models. The proposed model is capable of providing solutions to difficult problems which include coverage measurement in BBT [21].
Zhao R. et. al has proposed an approach for automatic generation of test cases from output domain. The model is created via Neural Networks and Genetic Algorithms is used to map output to its corresponding inputs. Results of experiment have concluded that test cases with higher efficiency can be generated from the output domain [22].
Md. Imrul Kayes presented a new metric for assessing rate of fault dependency detection and proposed an algorithm for prioritizing test cases. From experiments it has been concluded that prioritized test cases are more efficient than non-prioritized test cases [23].

Yang L. et. al. proposed a novel approach that is based on semantics and is independent of the syntactic appearance of the system specification. In some cases, semantic based approach is capable of giving good performance as compared to syntax based approach [24].
Shahbazi A. and Miller J., investigates black-box string test case generation techniques. Two objective functions are introduced for generating effective string test cases. Multi Objective Genetic Algorithms (MOGA) is applied to generate superior test cases [25].
Wicker M. et. al., focused on image classifier and have proposeda feature-guided black box approach for testing the safety of deep Neural Networks. The proposed algorithm is found efficient as it guarantees about theoretical safety under certain restrictions. This approach has been applied on a variety of state-of-the-art networks and benchmarks [26].
Sabharwal S. et. al. proposed a technique for optimizing efficiency of static testing by identifying the critical path clusters using genetic algorithm. The testing efficiency is optimized by applying the genetic algorithm on the test data. The approach uses IF model and GA to find the path to be tested first [27].

## 4. PROPOSED WORK

Prior to the execution of test cases, testers need to choose the test cases keeping two points in mind:

1) Good test cases should be generated
2) Prioritize test cases on some basis. Generally, test cases are prioritized on the basis of rate of fault detection.

The proposed technique introduces a new fitness function which may give superior results than existing ones. Here fitness function is combination of sum of rate of fault detection of test cases and total time of execution of complete test suite.
Figure 4.1 presents the process of proposed work.
The complete process is as follows:

- Test cases are executed first so as to find the rate of fault detection for each test case. Rate of fault detection is a measure that calculates the detection of as many faults as possible in the least amount of time i.e. as quickly as possible. It can be calculated as

$$\text{Rate of Fault Detection (VTi)} = \text{Number of faults detected/ Total time of execution} \quad (1)$$

- The process of GA starts by generating an initial start pool by generating the chromosomes randomly. Each candidate chromosome consists of an order in which test cases will be executed.
- The corresponding fitness values of candidate chromosomes are computed. The fitness value of each chromosome can be calculated as

$$\text{Fitness value} = 1/(1+e^{-\lambda k}) \quad (2)$$

Where λ= execution time for test suite and k= summation of rate of fault detection of every test case.

- The chromosomes are then arranged according to the fitness values i.e. fittest chromosomes are put first over others. Roulette Wheel selection method is used for selecting the chromosomes, which would undergo modification.
- Genetic operators like crossover and mutation would be applied to these selected chromosomes. Current population is updated by adding chromosomes with improved fitness values.
- The process is repeated till there is some betterment in the population. The algorithm terminates when any of the terminating conditions is satisfied. In the proposed work, process halts when the maximum number of specified generations (N) is reached.
- Final selection of test cases for execution is done randomly from the pool of chromosomes present in the last generation.
- On generating the initial population, the first chromosome directs the sequence of the execution of test cases to T1, T2, T3, T4, T5, T6, T7, T8, T9 and T10. The total time required for the successful execution for a test case is assumed as 1 unit per fault. As there are 10 faults which are to be detected, the total time required for execution of each test case is 10 units.
- Other chromosomes will also give good results which will not give optimal solutions but they will be near to the optimal solution. The flowchart of the Genetic Algorithms is as follows:

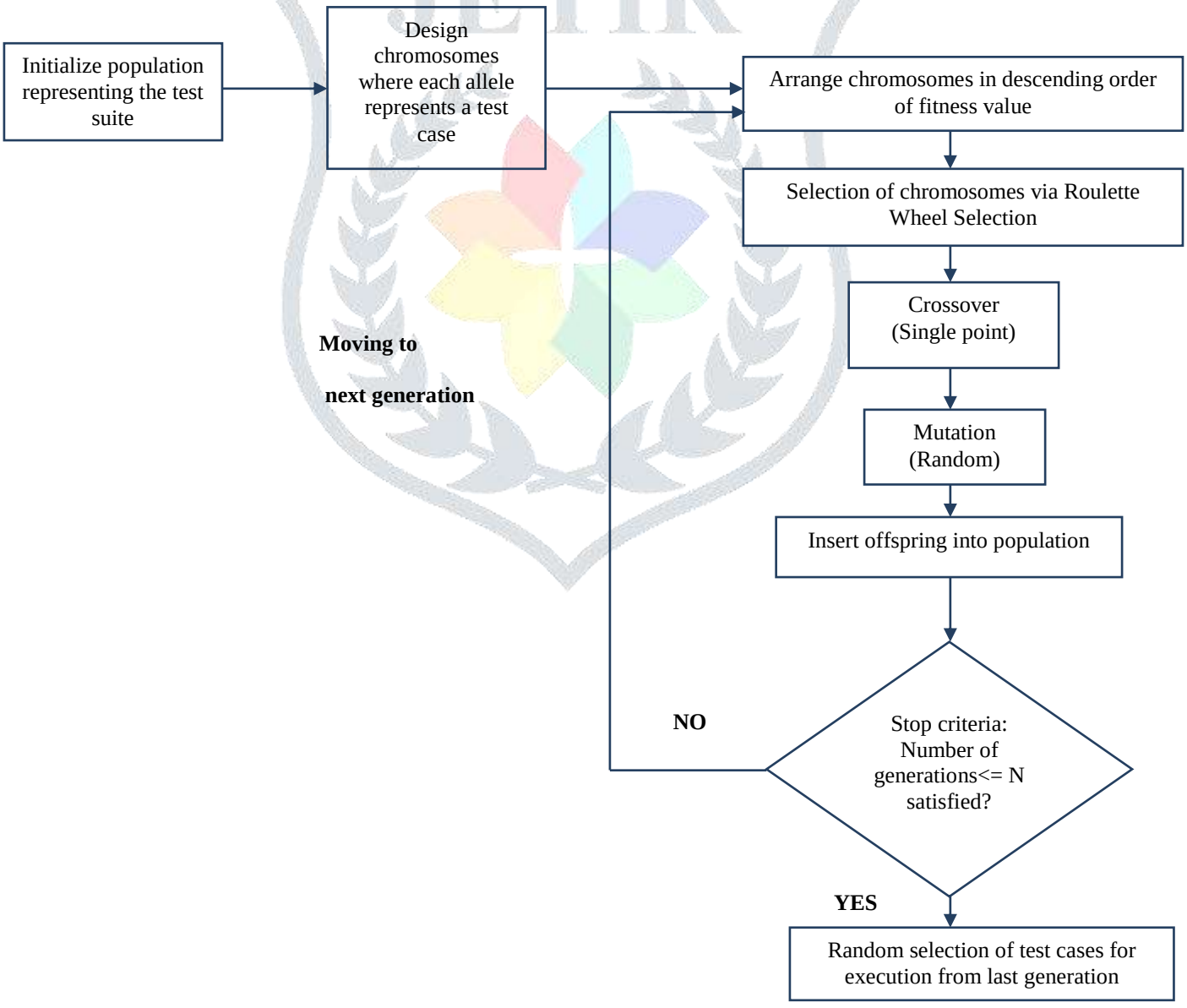


**Figure 4.1: Flowchart of the proposed approach**

## 5. RESULTS AND ANALYSIS

### 5.1 Results of proposed approach

The proposed work has been implemented on the data which is shown in Table 5.1.

#### 5.1.1 Proposed Approach

- The execution time for each test case is assumed to be 10 units. Hence the execution time of a test suite is 100 units.
- Table 5.1 presents the fault matrix and results of proposed approach.
- From the table, value of k can be calculated as summation of rate of fault detection of every test case i.e. k= 2.
- So from the data given in the table, fitness value for this particular chromosome using Eq.2 comes out to be

Fitness value = $1/ (1+e^{-2*100})$

= $1/ (1+e^{-200})$

=1

Similarly, the fitness value for each candidate chromosome can be calculated and the fittest chromosomes are selected for performing crossover and mutation. The set of improved chromosomes having optimized fitness function is the final solution generated by GA. The order of test cases as mentioned in final chromosomes is used in order to find the optimized solution.
The order of test cases of the selected optimal chromosome according to GA is: T10 T2 T6 T9 T1 T8 T3 T4 T5 T7.

**Table 5.1: Fault Matrix and results of proposed approach**

| Test cases / Faults | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | Total number of faults detected | Rate of Fault Detection of test cases(λ) | No of faults detected by test suite | Rate of Fault Detection of test suite |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | X | | | X | | | | | | | 2 | 0.2 | 2 | 0.2 |
| T2 | | | X | | | | | | X | | 2 | 0.2 | 4 | 0.4 |
| T3 | | | | | X | | | | | | 1 | 0.1 | 5 | 0.5 |
| T4 | | X | | | X | | | | X | | 3 | 0.3 | 8 | 0.8 |
| T5 | | X | | | | X | | | | | 2 | 0.2 | 10 | 1 |
| T6 | | | | | | X | | | | X | 2 | 0.2 | 10 | 1 |
| T7 | | X | | | X | | | | | | 2 | 0.2 | 10 | 1 |
| T8 | | | | | X | | | | | | 1 | 0.1 | 10 | 1 |
| T9 | | | | | | | X | X | | | 2 | 0.2 | 10 | 1 |
| T10 | X | X | | | X | | | | | | 3 | 0.3 | 10 | 1 |

Figure 5.1 shows a graph representing the rate of fault detection of test suite using GA with respect to test cases is shown below:

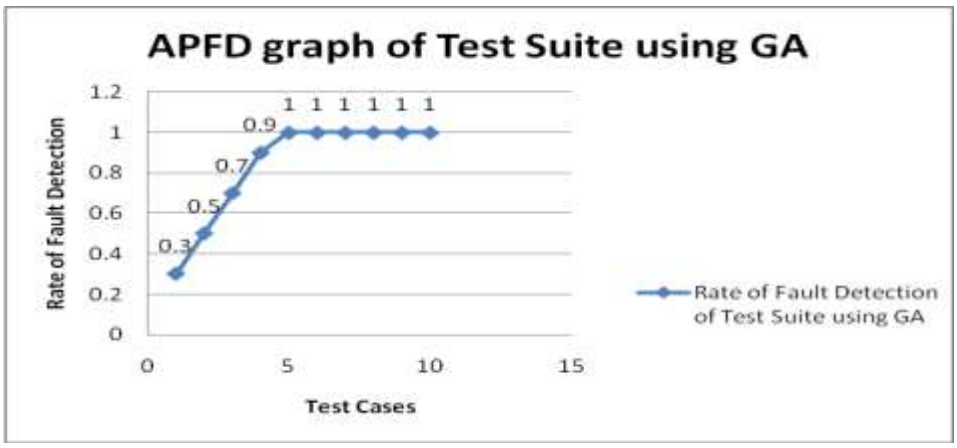


**Figure 5.1: APFD Graph representing rate of fault detection of test suite using proposed approach**

By using the selected order of test cases, a graph calculating the rate of fault detection of the complete test suite is shown above, where X axis represents test cases and Y axis represents rate of fault detection.

### 5.1.2 Approach which uses only Rate of Fault Detection

Rate of fault detection is one of the most popular methods which are used for prioritizing the test cases. Srivastava R. proposed an approach for prioritization of test cases using Rate of Fault Detection [28]. A test case having higher rate of fault detection is given high priority over others. With the help of Eq. 1, rate of fault detection for each test case can be calculated as:

VT1=2/10=0.2 VT2=2/10=0.2 VT3=1/10=0.01 VT4=3/10=0.3 VT5=2/10=0.2

VT6=2/10=0.2 VT7=2/10=0.2 VT8=1/10=0.1 VT9=2/10=0.2 VT10=3/10=0.3

Order of test cases prioritized by using only rate of fault detection is: T4 T10 T7 T5 T6 T1 T2 T9 T8 T3.

A graph representing the rate of fault detection of test suite with respect to test cases using decreasing order of rate of fault detection of individual test case is shown below in Figure 5.2.

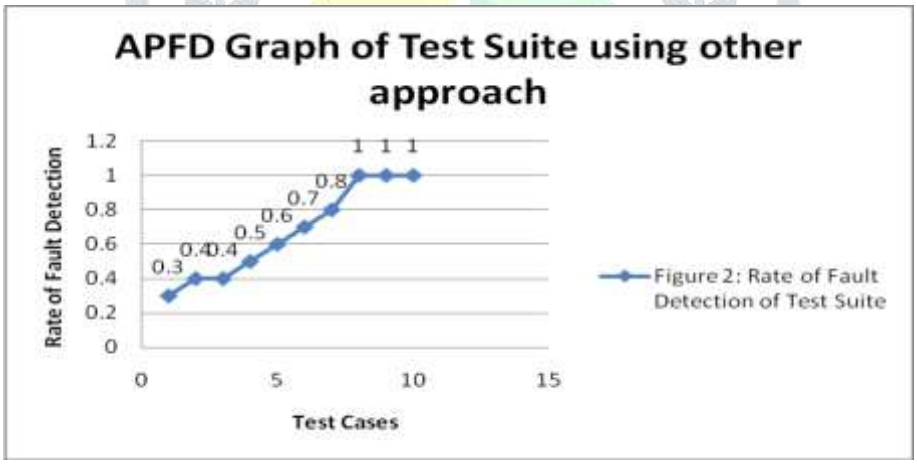


**Figure 5.2: APFD Graph representing rate of fault detection of test suite via rate of fault detection of individual test case.**

Figure 5.2 presents an APFD graph of an approach that uses a single factor i.e. rate of fault detection for prioritizing the test cases.

In order to calculate the level of effectiveness and to compare the results of both methods, APFD metric is used [29]. APFD can be defined as weighted average of the percentage of faults detected during the execution of the test suite [29]. It can be calculated as:

$$APFD=1-[(Tf1+Tf2\ldots.+Tfm)/(n*m)] + 1/2n \quad (3)$$

Where n is the number of test cases and m is the number of faults. (Tf1, Tf2,….,Tfm ) are the position of first test T that exposes the fault. The Number of test cases (n) = 10 and Number of faults (m) = 10.

Applying APFD to the derived result of GA:
APFD = 1 – {(2+ 3 + 1 + 2 + 1 + 3 + 3 + 1 + 3 + 2) / (10*10)} + {1/(2*10)}
= 1 – {21 / 100} + {1 / 20}
= 1 – 0.21 + 0.05
= 0.84
APFD value for test cases prioritized by using rate of fault detection as done in [28] is:
APFD =1– {(2+ 1+ 7+6+1+4+8+8+1+5) / (10*10)} + {1/(2*10)}
=1 – {43 / 100} + {1 / 20}
= 1 – 0.43 + 0.05
= 0.62
From the results, it has been proved that the proposed technique is more efficient than the technique which uses rate of fault detection of each test case for prioritizing the test cases. The proposed approach is capable of finding more number of faults in the least amount of time.

## 6. CONCLUSIONS AND FUTURE SCOPE

BBT is important as it does not require the code. In the literature many techniques of BBT have been suggested. This work proposes a novel technique which is based on rate of fault detection and uses Genetic Algorithms for optimization. The proposed work has been presented and implemented. It may be noted that the proposed work is based on the premise that some combinations of test cases are better than others and these combinations are the ones in which most of the faults are found as soon as possible. For experimental verification and validation, the proposed technique has been applied on a data set. APFD metric is used for performing comparison between the proposed approach and the approach that uses rate of fault detection for prioritization where the former approach is found more efficient. The technique would reduce the time of execution and is promising.

The future work intends to incorporate various machine learning methodologies in the above technique. It may also be stated that the future work intends to apply new variants of GA like diploid Genetic and Neural Network for the estimation of best set of test cases.

## 7. REFERENCES

**[1]** Chauhan, N. 2010. Software Testing: principles and practices. Oxford University Press.
**[2]** Last, M. et. al. 2006. Effective black box testing via genetic algorithms. Springer-Verlag Berlin Heidelberg.
**[3]** Sneha, K. and Gowda, M. 2017. Research on software testing techniques and software automation testing tools.
**[4]** Padhy, N.P. and Simon, S.P. 2015. Soft Computing: With Matlab Programming. Oxford, ISBN: 9780199455423, 0199455422. Edition: 1st Edition.
**[5]**Dorigo, M. 2006. An introduction to Ant Colony Optimization.
**[6]**Jacek, M. Z. 2005. Introduction to Artificial Neural Network.
**[7]**Stender, J. 2004. Introduction to genetic algorithms. IEEE.
**[8]** Agrawal, P. 2011. Ant Colony Optimization : A Technique used for Image Processing. IJCST, Vol. 2, Issue 2.
**[9]** Fischer, M. and Tonjes, R. 2012. Generating test data for black box testing using Genetic Algorithm. Conference: Emerging Technologies & Factory Automation (ETFA). IEEE 17th Conference.
**[10]**HATEM, M. 2018. SOLVING LARGE PROBLEMS WITH HEURISTIC SEARCH: GENERAL-PURPOSE PARALLEL EXTERNAL-MEMORY SEARCH. JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH 62.
**[11]**GOLDBERG, D. 1989. GENETIC ALGORITHMS IN SEARCH, OPTIMIZATION AND MACHINE LEARNING. ADDISON-WESLEY LONGMAN PUBLISHING CO.
**[12]** Malhotra, R. et. al. 2011. Genetic Algorithms: Concepts, Design for Optimization of Process Controllers. www.ccsenet.org/cis, Vol. 4, No. 2.
**[13]** REDSTONE Software. 2008. Black-box vs. White-box Testing: Choosing the Right Approach to Deliver Quality Applications. http://www.cs.unh.edu/~it666/reading_list/Defense/blackbox_vs_whitebox_testing.pdf.
**[14]** Khan, E. and Khan, F. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. International Journal of Advanced Computer Science and Applications, Vol. 3, No.6.
**[15]** Huima, A. 2006. A Note on an Anomaly in Black-Box Testing. Springer- Verlag Berlin Heidelberg.
**[16]** Walkinshaw, N. 2018. Testing Functional Black-Box Programs Without a Specification. Springer International Publishing AG, LNCS 11026: 101–120.
**[17]**Shahbazi, A. 2018. Black-box tree test case generation through diversity. Springer, Volume 25, Issue 3: 531–568
**[18]**Frezza, S.T. et. al. 1996. Linking requirements and design data for automated functional evaluation. Elsevier.
**[19]** Flores, A. and Polo, M. 2009. Testing-based Process for Evaluating Component Replaceability. Elsevier: 101- 115.

**[20]** Do, H. and Rotherme, G. 2006. On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques. IEEE Transactions on Software Engineering, Volume 32 Issue 9: 733-752.

**[21]**MEINKE, K.. 2006. A STOCHASTIC THEORY OF BLACK-BOX SOFTWARE TESTING. SPRINGER-VERLAG BERLIN HEIDELBERG.

**[22]** Zhao, R. and Lv, S. 2007. Neural-Network Based Test Cases Generation Using Genetic Algorithm. 13th IEEE International Symposium on Pacific Rim Dependable Computing : 97-100.

**[23]** Kayes, I. 2011. Test case prioritization for regression testing based on fault dependency. 3rd International Conference on Electronics Computer Technology (ICECT).

**[24]** Yang, L. 2011. Information gain of black-box testing, Formal Aspects of Computing. Springer, Volume 23 Issue 4: 513–539.

**[25]**Shahbazi, A. and Miller, J. 2016. Black-Box String Test Case Generation through a Multi-Objective Optimization. IEEE TRANSACTIONS ON JOURNAL NAME, MANUSCRIPT ID, Volume: 42 , Issue: 4 : 361 - 378.

**[26]** Wicker, M. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks, D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10805:408–426.

**[27]** Sabharwal, S. 2011. Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3: 2.

**[28]** Srivastava, R. 2005-2008. TEST CASE PRIORITIZATION. Journal of Theoretical and Applied Information Technology.

**[29]** Elbaum, S. et. al. 2000. Prioritizing test cases for regression testing.