# A Genetic Algorithm for Fault based Regression Test Case Prioritization

Dr. Arvinder Kaur
Associate Professor,
University School of
Information Technology
G.G.S. Indraprastha
University, Delhi- 110043

Shubhra Goyal
Research Student
University School of
Information Technology,
G.G.S. Indraprastha
University, and Delhi- 110043

## ABSTRACT
Regression testing is the process of validating modified software to detect errors that have been introduced into previously tested code. As the software is modified, the size of the test suite grows and the cost of regression testing increases. In this situation, test case prioritization aims to improve the effectiveness of regression testing by ordering the test cases so that most beneficial test cases are executed first. In this research paper, a new genetic algorithm is introduced that will prioritize regression test suite within a time constrained environment on the basis of total fault coverage. The proposed algorithm has been automated and the results are analysed. The results representing the effectiveness of algorithm are presented with the help of Average Percentage of Faults Detected (APFD).

## Keywords
Genetic Algorithm; Test Case Prioritization; Regression Testing.

## 1. INTRODUCTION
Throughout the software life cycle, most test cases should be written adequately to test the software. As the software is altered, a maintenance activity called regression testing is performed to ensure the validity of the modified software. To test the modified software, new test cases are added to the test suite to test the changed requirements, which increase the size of the test-suite, cost and time constraints. So, to enhance the efficiency of software testing, improvements in the regression testing would help in reducing the cost of the software. To reduce the cost of regression testing, prioritization of test cases becomes essential. Several techniques have been proposed for the above techniques that are described in the next section of related work.

This paper, investigates the use of an evolutionary approach, called Genetic Algorithm for test case prioritization based on total fault coverage in minimum execution time. Software test automation refers to the activities that are required to automate the test case generation, prioritization and execution of the test cases. The proposed algorithm automates the process of prioritization of test suites on the basis of complete fault coverage using genetic algorithm.

## 2. RELATED WORK
This section presents the work done in the area of genetic algorithm and test case prioritization. Genetic algorithms are used in many areas such as cost estimation problem, hardware-software embedded systems, cryptography, data warehousing and data mining. In the cost estimation problem, the size of the software, usually measured in lines of code or function points, is examined in relation to the effort, which is usually measured in person-months [1]. Search algorithms especially genetic algorithms are used in order to find predictive functions for the relation. The initial population is formed from a set of well-formed equations, to which the operators of a genetic algorithm are applied [1]. The main benefit of using a genetic algorithm is it explores solutions solely based on their fitness values and does not constrain the form of the solution. Thus, even complex evaluation functions have the possibility of being found and the final set of equations provided by the genetic algorithm truly have the best predictive values [1].
A genetic algorithm addresses the problem of co synthesizing hardware-software embedded systems [2]. A co-synthesis system determines the hardware and software processing elements (PE) that are needed and the links that are used for a given embedded system. A co-synthesis system must carry out four tasks: allocation, assignment, scheduling, and performance evaluation. The allocation/assignment and scheduling tasks are known to be NP-complete for distributed systems, so the co-synthesis problem is an excellent candidate for search algorithms [2]. Genetic algorithm has been used in the field of networks. They are been used in network intrusion detection system(IDS). In this technique, both temporal and spatial information of network connections in encoding the network connection rules in IDS is used and is more helpful for identification of network anomalous behaviour [3]. Genetic algorithms are used in cryptanalysis. An algorithm is developed for finding the secret key of a block permutation cipher [4]. Genetic algorithms are applied in the field of data warehouse and datamining. An algorithm is developed based on GA, for incremental clustering in data mining and the efficiency of the algorithm is demonstrated. ICGA requires distance function and, therefore, it is applicable to any database containing data from a metric space [5]. It has been used in the field of robotics for robot navigation controller optimation, specially that are based on neural networks [6].
In recent years several researchers have addressed the test case prioritization problem and presented techniques for addressing

it. Test case prioritization techniques reported in [7, 8] orders test cases such that the test cases with highest priority, according to criteria defined by user, are executed first [9]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing, to reach 100% coverage or to ensure that the maximum possible coverage is achieved by some pre–determined cut–off point. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [10]. Several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process. There are several prioritizing techniques such as total statement (or branch) coverage prioritization and additional statement coverage prioritization that can improve the rate of fault detection [11]. Test cases are prioritized according to the criterion of 'increasing cost per additional coverage [8]. Greedy Algorithms are also used and are implemented in a tool named ATAC [15]. A prioritization technique has been presented that is based on the changes that have been made to the program and focused on the objective function of "impacted block coverage"[12]. Other non–coverage based techniques in the literature include fault–exposing–potential (FEP) prioritization [9], history–based test prioritization [13], and the incorporation of varying test costs and fault severities into test case prioritization [11,13]. Five search techniques: two meta–heuristic search techniques (Hill Climbing and Genetic Algorithms), together with three greedy algorithms (Basic Greedy, Additional Greedy and 2–Optimal Greedy) had been studied and proved that Genetic Algorithms performed well in test case prioritization [14]. In our proposed prioritization technique test suite execution time along with coverage information has been used to prioritized.

## 3. THE GENETIC ALGORITHM

Genetic algorithms were invented by John Holland in the 1960s. Genetic algorithms are used to find an optimal solution that satisfies the criteria defined by user to reach the desired goal. To operate with a genetic algorithm, one needs an encoding of the solution, i.e., a representation of the solution in a form that can be interpreted as a chromosome, an initial population, mutation and crossover operators, a fitness function and a selection operator for choosing the survivors for the next generation.

### 3.1 Methodology

In a genetic algorithm, a population of strings (called chromosomes), which encode candidate solutions to an optimization problem, evolves toward better solutions.

Initialization

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.

Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where better solutions are likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions.

Reproduction

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover and mutation. For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the methods of crossover and mutation, a new solution is created which shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. The crossover operator is applied to two chromosomes (the parents), in order to create two new chromosomes (their offspring). For example, if the two parents are [*v1,...,vm*] and [*w1,...,wm*], then crossing the chromosomes after the *k*th gene ( $1 \leq k \leq m$) would produce the offspring: [*v1,...,vk,wk+1,...,wm*] and [*w1,...,wk,vk+1,...,vm*]. Mutations are a way of creating new individuals from the population at hand by administering a minor change to one of the existing individuals by changing alleles in a random locus. For example, we could have a bit string 001100. By mutating this string in its third locus the result would be 000100[16].

Termination

This process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of above criterias[17]

.

## 4. GENETIC ALGORITHM FOR PRIORITIZATION OF TEST CASES

This paper present a new Genetic Algorithm that uses genetic operators, crossover and mutation to prioritize test cases based on maximum fault coverage. This algorithm takes number of test cases as the number of chromosomes and stopping criteria for each chromosome is total fault coverage

## 4.1 Flowchart

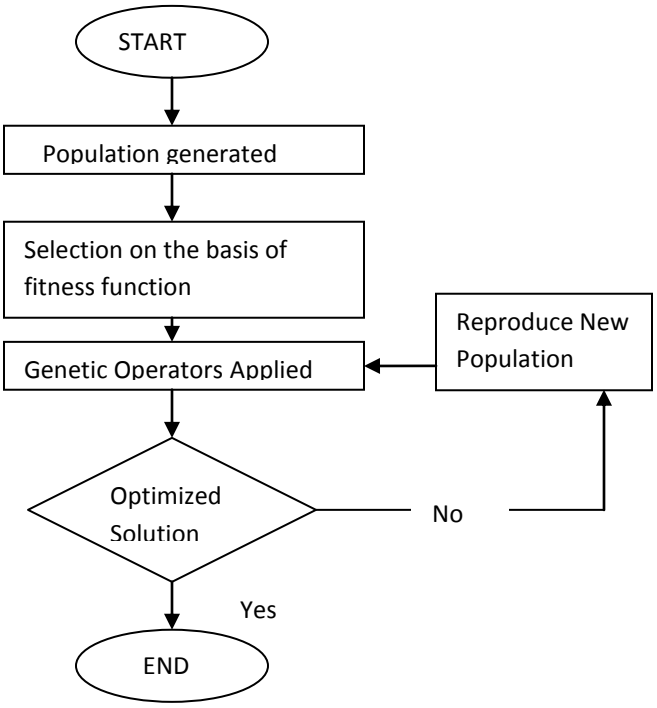


**Fig 1. Flowchart of Genetic Algorithm.**

## 4.2 Algorithm

STEP 1. Initialization of initial population
Generate 'n' number of chromosomes {$c_1$, $c_2$… $c_n$}
Set No. Test Suite= No. of chromosomes (n)

STEP 2. Fitness function criteria
Set fitness function= total fault coverage + minimum time of execution to run the selected test case

STEP 3. Select suitable population on the basis of Fitness Function
SELECT (Best 2 chromosomes)

STEP 4. Genetic Operators Applied
Do for selected Chromosome(s)
While (all faults are covered)
Do crossover
Do mutation
Duplication removed
EndWhile
EndFor

STEP 5. Optimization of solution cheked.
If (solution!= optimised)
Goto STEP 4
Else END.

## 4.3 Algorithm Explained

In GA, the optimal solution is searched on the basis of desired population which further can be replaced with the new set of population. The generation and initialization of test cases (population) is done according to the problem. The fitness criterion chosen is maximum fault covered in minimum execution time to run the test cases. Henceforth, this fitness function will help in selecting suitable population for problem. Further, the genetic operations are performed. First, crossover, which recombines two individuals. Second, mutation, which randomly swaps the individuals. Third, the duplicate individuals are removed. Finally, the solution is checked for optimization. If solution is not optimized, then, the next generation population is reproduced and genetic operations are applied.

## 4.4 Analysis Of Algorithm

The execution time of Genetic Algorithm is bounded by sum of time required to generate the population, applying genetic operators 'n' number of times and checking the optimal solution. We use O-notation to give an upper bound on a function within a constant factor. For the population of size 'n', the population generation requires O(n) operations . All 'n' chromosomes in population will go through crossover and mutation and removal of duplicity till the final test suite is obtained through GA. Since, only two chromosomes are selected it takes a total time of 2[n(O(n)) + 2]. For final result all the 'n' individuals will go for maximum of 3 iterations as 3 point crossover is possible. Hence, taking 3O(n) operation complexity. Therefore, the best running time of proposed algorithm is O(n) + O(n) + 1 + O(n) + 2[n(O(n)) + 2]+ 3O(n), which makes the final running time as $O(n^2)$.

## 4.5 Implementation Of the Proposed Algorithm

Test cases are prioritized on the basis of total fault coverage using Genetic Algorithm. The algorithm is developed and then implemented using Java IDE. The code takes the input of test cases with the faults they cover and execution time to run those test cases and prioritize the test cases based on fitness function and genetic operators' crossover and mutation.

## 4.6 Problem Definition

The code is analyzed using five examples and the results are shown.

Example 1

The problem taken is "college program for admission in courses". The problem specification is available at website http://www.planet-source-code.com.In this example a test suite has been developed which consisted of 40 test cases. For

simplification, to explain the technique, a test suite with 9 test cases is considered in it, covering a total of 5 faults. The regression test suite T contains nine test cases with the default ordering {T1, T2, T3, T4, T5, T6, T7, T8, T9}, the faults covered by each test case and the execution time required by each test case in finding faults are as shown in Table 1. The equal priority is given to the number of faults covered and minimum execution time in selecting test suites.

**Table 1. Test cases with faults covered and execution time for example 1**

| Test case No. | Faults Covered | Execution Time(Units) |
|---|---|---|
| 1 | 1 2 3 5 | 11.5 |
| 2 | 1 2 | 11.5 |
| 3 | 1 3 5 | 12.33 |
| 4 | 1 4 5 | 10.66 |
| 5 | 1 | 15.0 |
| 6 | 1 3 5 | 8.33 |
| 7 | 1 | 15.0 |
| 8 | 1 2 4 | 10.0 |
| 9 | 3 6 | 11.0 |

OUTPUT
Selected chromosomes after applying Genetic Algorithm are: 6, 8 which cover all five faults and total Execution Time to run these two test cases is : 18.33 unit.

Example 2

Another problem specification is for software "Hotel Reservation" which reserves the rooms in hotel and maintains the record. The complete problem specification is available at the website http://www.planet-source-code.com. It contains 40 test cases with 10 faults as shown in the table 2.

OUTPUT
Selected chromosomes after applying Genetic Algorithm are: 5, 1 which cover all ten faults and Execution Time to run these test cases is: 20.7 unit.

Example 3

Another problem specification is for software "Railway Reservation" which reserves and cancels the seats in railways. It contains 26 test cases with 10 faults as shown in the table 3.

OUTPUT
Selected chromosomes after applying Genetic Algorithm are: 2,1,3,4 which cover all ten faults and Execution Time to run these test cases is: 54.0 units

**Table 2. Test cases with faults covered and execution time for example 2**

| Test Case No. | Faults Covered | Execution Time(Units) |
|---|---|---|
| 1 | 1 3 4 5 7 8 9 10 | 10.1 |
| 2 | 1 3 5 7 | 12.8 |
| 3 | 2 3 5 6 8 | 11.28 |
| 4 | 1 5 7 10 | 18.9 |
| 5 | 2 6 | 10.6 |
| 6 | 1 2 3 4 7 8 9 | 20.8 |
| 7 | 1 2 3 6 | 18.66 |
| 8 | 2 3 5 8 10 | 22.6 |
| 9 | 1 5 7 9 | 23.68 |
| 10 | 1 3 4 5 8 | 16.68 |
| 11 | 3 5 6 | 19.9 |
| 12 | 2 3 8 9 | 15.5 |
| 13 | 1 2 4 7 10 | 10.28 |
| 14 | 1 2 | 14.48 |
| 15 | 2 3 8 | 22.68 |
| 16 | 2 3 4 5 6 9 | 17.34 |
| 17 | 1 3 5 | 21.82 |
| 18 | 1 5 8 10 | 26.62 |
| 19 | 2 3 4 5 | 25.28 |
| 20 | 1 2 3 7 8 9 | 18.8 |
| 21 | 1 2 3 | 15.76 |
| 22 | 2 6 8 10 | 19.86 |
| 23 | 1 3 5 | 20.21 |
| 24 | 1 2 3 4 5 8 9 | 21.0 |
| 25 | 1 3 5 7 9 | 13.68 |
| 26 | 2 5 8 9 | 16.28 |
| 27 | 1 2 3 6 9 | 10.19 |
| 28 | 1 2 3 5 8 | 10.28 |
| 29 | 2 3 5 7 | 18.79 |
| 30 | 2 3 4 5 6 8 10 | 27.19 |
| 31 | 1 3 4 5 6 8 10 | 29.86 |
| 32 | 1 3 4 7 9 | 30.8 |
| 33 | 2 7 10 | 32.68 |
| 34 | 1 2 3 6 8 | 19.29 |
| 35 | 1 2 7 | 27.28 |
| 36 | 4 5 9 10 | 18.86 |
| 37 | 1 | 25.57 |
| 38 | 1 2 8 | 23.86 |
| 39 | 1 2 3 6 7 9 10 | 15.78 |
| 40 | 1 2 3 4 5 10 | 30.31 |

**Table 3. Test cases with faults covered and execution time for example 3**

| Test Case No. | Faults Covered | Execution Time (Units) |
|---|---|---|
| 1 | 2 | 8.0 |
| 2 | 5 8 9 10 | 12.0 |
| 3 | 1 3 4 6 9 10 | 16.0 |
| 4 | 1 3 4 5 7 9 10 | 18.0 |
| 5 | 1 3 4 | 16.0 |
| 6 | 1 2 4 | 14.0 |
| 7 | 1 2 4 | 15.0 |
| 8 | 1 2 4 | 14.0 |
| 9 | 1 2 4 | 12.0 |
| 10 | 1 2 4 | 14.0 |
| 11 | 1 2 4 | 14.0 |
| 12 | 1 2 4 | 14.0 |
| 13 | 1 2 3 4 | 13.0 |
| 14 | 1 2 3 4 | 13.0 |
| 15 | 1 2 3 4 | 13.0 |
| 16 | 1 2 3 4 | 13.0 |
| 17 | 1 2 3 4 | 13.0 |
| 18 | 1 2 3 4 | 13.0 |
| 19 | 1 2 3 4 | 13.0 |
| 20 | 1 2 4 10 | 15.0 |
| 21 | 1 2 3 4 | 13.0 |
| 22 | 1 2 4 | 14.0 |
| 23 | 1 2 4 10 | 14.0 |
| 24 | 1 2 4 10 | 14.0 |
| 25 | 1 2 4 10 | 13.0 |
| 26 | 1 2 4 10 | 13.0 |

Example 4
The example taken is the triangle problem. It takes the three sides of the triangle as input and gives the output as scalene, isosceles, equilateral and not a triangle according to the input[18], contains 17 test cases and 6 faults as shown in the table 4.

**Table 4. Test cases with faults covered and execution time for example 4**

| Test Case No. | Faults Covered | Execution Time(Units) |
|---|---|---|
| 1 | 1 3 6 | 5.0 |
| 2 | 1 2 3 6 | 2.0 |
| 3 | 1 2 3 6 | 3.0 |
| 4 | 1 2 3 4 | 4.0 |
| 5 | 1 2 3 4 5 | 5.0 |
| 6 | 1 2 3 6 | 3.0 |
| 7 | 1 2 3 6 | 6.0 |
| 8 | 1 3 6 | 4.0 |
| 9 | 1 2 3 6 | 5.0 |
| 10 | 1 2 3 6 | 3.0 |
| 11 | 1 2 3 4 5 | 8.0 |
| 12 | 1 2 3 6 | 4.0 |
| 13 | 1 2 3 6 | 6.0 |
| 14 | 1 3 6 | 3.0 |
| 15 | 1 2 3 6 | 2.0 |
| 16 | 1 2 3 6 | 3.0 |
| 17 | 1 3 6 | 5.0 |

OUTPUT
Selected chromosomes after applying Genetic Algorithm are: 2, 4 which cover all six faults and Execution Time to run these test cases is: 7.0 units

Example 5

The example taken is the quadratic equation problem which takes the three numbers of the equation as input and gives the output as equal roots, real roots, imaginary roots and not a quadratic equation according to the input[18], contains 19 test cases and 9 faults as shown in the table 5.

**Table 5. Test cases with faults covered and execution For example 5**

| Test Case No. | Faults Covered | Execution Time (Units) |
|---|---|---|
| 1 | 1 2 | 3.0 |
| 2 | 3 8 9 | 5.0 |
| 3 | 1 4 7 9 | 2.0 |
| 4 | 1 4 5 9 | 6.0 |
| 5 | 1 4 6 8 | 3.0 |
| 6 | 1 3 6 9 | 4.0 |
| 7 | 1 2 | 2.0 |
| 8 | 0 | 6.0 |
| 9 | 1 4 5 6 7 | 4.0 |
| 10 | 1 3 5 6 8 | 7.0 |
| 11 | 1 2 5 6 9 | 3.0 |
| 12 | 1 4 5 6 8 | 2.0 |
| 13 | 2 5 | 7.0 |
| 14 | 2 3 | 3.0 |
| 15 | 2 5 7 9 | 5.0 |
| 16 | 1 4 5 7 9 | 6.0 |
| 17 | 1 4 5 6 9 | 3.0 |
| 18 | 1 3 5 6 8 | 4.0 |
| 19 | 2 | 1.0 |

OUTPUT
Selected chromosomes after applying Genetic Algorithm are: 2, 9, 6 which cover all nine faults and Execution Time to run these test cases is: 11.0 units

## 4.7 Analysis of the results

The examples that are prioritized using genetic algorithm are analyzed ten times each and the results are compared and analyzed as shown:

Example 1

The college program test cases is run ten times and the result are recorded and prioritized test cases that cover total faults are (6,8),(4,9),(8,9) the test case (6,8) cover all faults in Minimum execution time of 18.33 units and out of ten the result (6,8) is six times and (4,9) and (8,9) are two times each. Genetic Algorithm gives the optimum result 60% times.

Example 2

The hotel reservation test cases are run ten times and the results are recorded and the prioritized test cases that cover total faults are: (1, 7), (1, 3) and (5, 1). the test cases (5,1) cover total faults in minimum execution time of 20.7 units and out of ten, the result (5,1) is six times, (1,7) and (1, 3) are two times each. Genetic Algorithm gives the optimum result 60% times

Example 3

The railway reservation test cases are run ten times and the results are recorded and the prioritized test cases that cover total faults are: (6,2,3,4), (10,2,3,4), (2,1,3,4) and (3,9,4,2). The test cases (2,1,3,4) cover total faults in minimum execution time of 54.00 units and out of ten, the result (2,1,3,4) is five times, (6,2,3,4), and (10,2,3,4) are two times each and (3,9,4,2) is one time. Genetic Algorithm gives the optimum result 50% times.

Example 4

The triangle problem test cases are run ten times and the results are recorded and the prioritized test cases that cover total faults are: (5,10), (2,5), (8,5) and (9, 5). the test cases (2,5) cover total faults in minimum execution time of 7.00 units and out of ten, the result (2,5) is six times, (8,5) is two times and (5,10) and (9,5) is one time each. Genetic Algorithm gives the optimum result 60% times.

Example 5

The quadratic equation problem test cases are run ten times and the results are recorded and the prioritized test cases that cover total faults are: (3,10,7) and (2,7,9). Both the test cases cover total faults in minimum execution time of 11.00 units and out of ten, the result (3,10,7) is six times and (2,7,9) is four times. Since, both the test cases cover total faults in minimum execution time, therefore, Genetic Algorithm gives the optimum result 100% times.

Five examples had been run for ten times each using Genetic algorithm and the percentage of optimum results obtained are summarised in table 6.

**Table 6. Summary table of all the examples**

| Example No. | No. of test cases | No. of faults | Optimal GA order | No. of runs | Optimum Runs | efficiency |
|---|---|---|---|---|---|---|
| 1 | 9 | 5 | 6,8 | 10 | 6 | 60% |
| 2 | 40 | 10 | 5,1 | 10 | 6 | 60% |
| 3 | 26 | 10 | 2,1,3,4 | 10 | 5 | 50% |
| 4 | 17 | 6 | 2,5 | 10 | 6 | 60% |
| 5 | 19 | 9 | 3,10,7 | 10 | 10 | 100% |

$$\textbf{Efficiency=} \frac{numberofoptimumruns}{total\ numberofruns} * 100$$

## 4.8 Comparison

In this section, the fault based testing is compared with respect to the following prioritization: No order,Random Order, Reverse order, optimal order of the test cases. These approaches are compared by calculating Average Percentage of Faults Detected (APFD) [8] which is given by the equation 1 and figure 2-6 plots the APFD.

$$\text{APFD}= \quad 1-\frac{TF1+TF2+.....TFm}{nm}+1/2n \quad (1)$$

Where, T - The test suite under evaluation;
m - The number of faults contained in the program under test P;
n - The total number of test cases;
$TF_i$ - The position of the first test in T that exposes $i^{th}$ fault.

| Technique | Example 1 APFD % | Example 2 APFD % | Example 3 APFD % | Example 4 APFD % | Example 5 APFD % |
|---|---|---|---|---|---|
| No Order | 77 | 46 | 92.32 | 89.2 | 88 |
| Random Order | 58 | 62 | 78.5 | 84.3 | 84.5 |
| Reverse Order | 71 | 62 | 52.72 | 84.3 | 90.4 |
| Optimal Order | 80 | 62 | 92.32 | 96.1 | 91 |
| GA Order | 80 | 62 | 93.5 | 95.1 | 93.8 |

**Table 7. Representing APFD values**

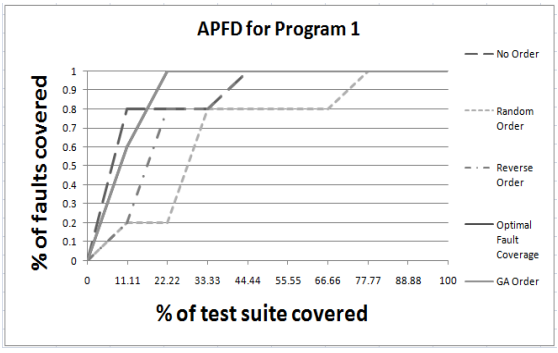


**Fig. 2. APFD for example 1 of total fault coverage**

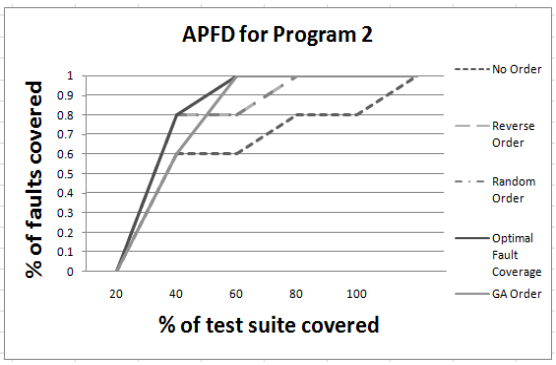


**Fig. 3. APFD for example 2 of total fault coverage**

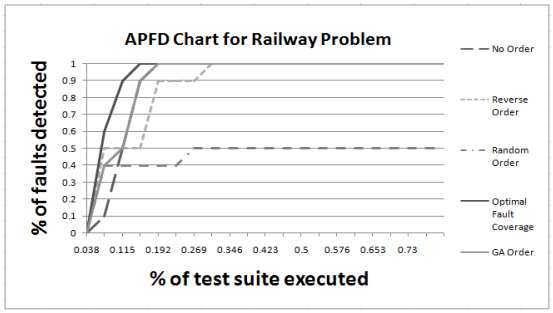


**Fig. 4. APFD for example 3 of total fault coverage**

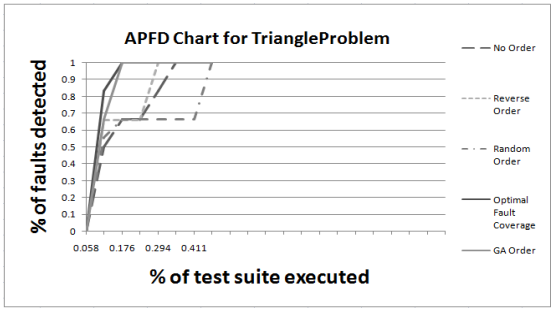


**Fig. 5. APFD for example 4 of total fault coverage**

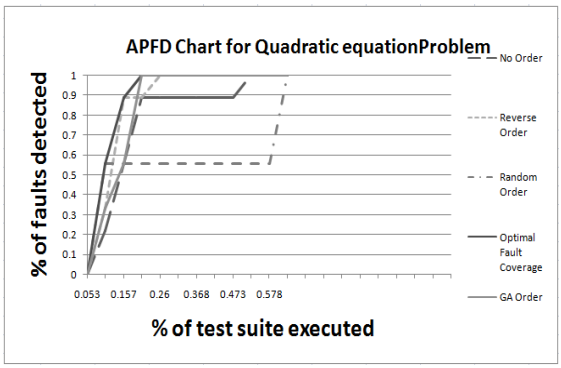


**Fig. 6. APFD for example 5 of total fault coverage**

## 5. CONCLUSION

The algorithm has been proposed to prioritize test cases using Genetic Algorithm. Here, total fault coverage with in time constrained environment on different examples is used for prioritization of test cases and their finite solution is obtained. Through Genetic Algorithm technique, an approach has been identified to find a suitable population, which was further formulated by GA operations to make it more flexible and efficient. The elaborations of results are shown with the help of APFD metrics. The APFD has been calculated to evaluate the usefulness of the proposed algorithm. The algorithm has been automated and are analysed for various examples.

The algorithms implemented are used only for an integer input. In future, it has to be developed for string input variable so we can generate and prioritize test cases for any input value.

# 6. REFERENCES

[1] Clarke. J., Dolado, J. J., Harman, M., Hierons, R. M., Jones, B. and M. Shepperd, Reformulating, "Software Engineering as a Search Problem," *IEEE Proceedings - Software*, vol.150, No.3, 2003, pp. 161-175.

[2] Dick, R.P. and Jha, N.K. "MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.17, No. 10, Oct. 1998, pp. 920-935.

[3] li, W. "using genetic algorithm for network intrusion detection", Masters project report, 2002.

[4] gorodilov, A. and morozenko, V. "Genetic algorithm for finding key's length and cryptanalysis of the permutation cipher," *International journal of information theories and applications*, vol. *15*, 2008.

[5] kamble, A. "Incremental clustering in data mining using genetic algorithm," *International journal of computer theory and engineering*, vol.2, no.3, June 2010, pp. 1793-8201.

[6] Cernic, S., Jezierski, E., Britos, P., Rossi, B. and García Martínez, R. "Genetic algorithm applied to robot navigation controller".

[7] Rothermel, G., Untch, R., Chu, C. and Harrold, M. J. "Test case prioritization: An empirical study", *In Proceedings ICSM 1999*, Sept. 1999, pp. 179–188.

[8]Wong, W. E., Horgan, J. R., London, S. and Agrawal, H. "A study of effective regression testing in practice", *In Proceedings of the Eighth International Symposium on Software Reliability Engineering*, November 1997 , pp. 230– 238

[9] Rothermel, G., Untch, R., Chu, C. and Harrold, M. J. "Prioritizing test cases for regression testing", *IEEE Transactions on Software Engineering*, vol. 27, No.10, October 2001, pp.929–948.

[10]MicrosoftCorporation.Regressiontesting.http://msdn.microsoft.com/library/default.asp?url=/library/enus/vsent7/html/vxconregressiontesting.asp.

[11] Elbaum, S., Malishevsky, A. G. and Rothermel, G. "Test case prioritization: A family of empirical studies", *IEEE Transactions on Software Engineering*, vol.28, No. 2, 2002 , pp.159–182.

[12] Srivastava, A. and Thiagarajan, J. "Effectively prioritizing tests in development Environment, *in Proceedings of the ACM SIGSOFT international symposium on Software testing and analysis*, USA, 2002, pp. 97–106.

[13] Kim, J.M. and Porter, A. "A history-based test prioritization technique for regression testing in resource Constrained environments", *In Proceedings of the 24th International Conference on Software Engineering*, 2002 , pp. 119–129.

[14] Li, Z., Harman, M. and Hierons, R. M. "Search Algorithms for Regression Test Case Prioritization", *IEEE Transactions on software Engineering*, vol.33, No.4, April 2007, pp. 225-237.

[15]Roubtsov,V. "Emma a free java code coverage tool", http://emma. sourceforge.net/index.html, March.2005.

[16]Mitchell, M. "*An Introduction to Genetic Algorithms*", MIT Press, USA, 1996.

[17]. Rothermel, G., Untch, R., Chu, C. and Harrold, M. J. "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, October 2001, vol. 27, no. 10, pp. 929-948.

[18].Aggarwal, K.K. and Singh, Y. "A book on software engineering", New Age International (P) Ltd.; Publishers, 4835/24, Ansari Road, Daryaganj, New Delhi, 2001.