Test Case Optimization for Enhancing System Software Quality using Genetic Algorithm

M. Bharathi , V. Sangeetha

Abstract: *Software Product Lines (SPLs) embraces an enormous capacity of feature mixtures which cause challenges for evaluating software programs. Testsuite optimization plays major role to develope the quality of SPLs. In combinatorial testing (CT), pair wise fault coverage maximization and test case reduction accomplishes a substantial role for shrinking the testing cost of software programs. Many research works have been developed and designed for CT using different test suite reduction techniques. However Fuzzy clustering and TSRSO techniques do not provide a finest solution for test suite optimization problem. For that, Genetic Algorithm (GA) Technique is recommended and designed for test suite reduction in CT. Metaheuristic genetic algorithm delivers optimum solution in an effective manner. GA chooses and consolidates the testcases in a testsuite based on some principles such that maximum faults covered with minimum execution time. In Proposed GA, finest individuals are nominated for reproduction in order to create descendants of the succeeding generation. In addition, GA is a superior type of evolutionary algorithms generate finest solutions to optimization problems using selection, crossover and mutation operators. Consequently, GA is applied for resolving test suite reduction problem in CT.*

Keywords: *Software Product Lines, Combinatorial Testing, Test Suite Optimization, Test Cases, Fault coverage.*

I. INTRODUCTION

CT picks software package to be verified founded on the couples of feature mixtures. Hence, each mixture of feature couples of structure must shielded by atleast single test case in test suite. CT diminishes the quantity of feature couple mixtures desired to investigate software structure excellence as matched to comprehensive testing. Also, test suite optimization acts essential role to diminish the testing rate of software package. Henceforth, this examination effort focus on test suite optimization for powerfully refining the demonstration of CT.

Fuzzy clustering method was offered to diminish the interval necessitated for testing over shortening the amount of test cases. However, test suite optimization exhibition was not energetic for accomplishing greater coverage ratio [1]. A test-suite reduction model based on the subtraction operation (TSRSO) was used to reduce quantity of test cases by means of a row renovation process of matrix. Nevertheless, testing rate for software package was extended [2].

Revised Manuscript Received on October 30, 2019.

***** Correspondence Author
M.Bharathi*, Department of Computer Science, Periyar University College of Arts and Science, Pennagaram, Dharmapuri, TamilNadu, India. Email: abinovharshiniset@gmail.com
Dr.V.Sangeetha, Department of Computer Science, Periyar University **College** of Arts and Science, Pappireddipatti, Dharmapuri, TamilNadu, India. Email: sangee759@gmail.com

Test Suite minimization is an approach, plays an effective role in CT to expand pairwise analysis and to shrink the amount of test cases without corrupting their factors. Hence, GA is suggested in this investigation process to gain test suite diminution for fulfilling CT and thereby quality of software system keeps improving. In GA, best chromosomes are elected for recreation of descendants in the succeeding generation. Moreover, GA is a superior type of Evolutionary Algorithms (EA) that generate finest solutions to optimization problems by means of special operators of GA called selection, crossover and mutation.

This assessment work is framed as follows. In Section 2 we summarize the correlated works. Section 3 describes how to implement Genetic Algorithm for test suite optimization problem. In Section 4 we demonstrate the experimental settings and section 5 we explain evaluation of results and discussions. In section 6 we present conclusions and its future work.

II. RELATED WORKS

Bestoun S. Ahmed proposed cuckoo search (CS) algorithm which minimize the number of test cases in configuration-aware structural testing. But, higher computational time was required for test suite optimization problem [3].Andrea Calvagna, Angelo Gargantini, have proposed A Formal Logic Approach which did not offer better coverage rate and test cost [4]. Xiaofang Qi, Jun He, Peng Wang, Huayang Zhou proposed a variable strength reachability testing strategy for performing combinatorial testing in a concurrent program. However, this strategy did not solve test suite optimization [5].Abhishek Pandey, Soumya Banerjee was proposed firefly algorithm for Test Suite Optimization problem. However, test suite optimization was not achieved [6]. Manju Khari, Prabhat Kumar proposed a Cuckoo Search Algorithm for test suite optimization. But, optimization process was required high execution time [7]. Shilpi Singh, Raj Shree developed an approach for optimizing test cases in a testsuite. However, proposed approach did not solve pairwise coverage rate of test suite [8].
Luciano et al. introduced a hybrid particle swarm optimization approach for optimizing the test cases with multi-objectives. But, problem of test suite optimization was not solved effectively [9].

We proposed metaheuristic genetic algorithm to resolve the above stated existing issues.

III. IMPLEMENTATION STEPS OF GENETIC ALGORITHM

The following Figure1 exhibits the process of proposed genetic algorithm for test suite contraction. Primarily GA initializes the population using contribution of test cases in test suite. Afterwards, an objective function is evaluated and if objective function achieves threshold (T) then test case is nominated as best test cases for optimization problem oppositely a fitness function of all the test cases is evaluated depend on more objectives called fault coverage and execution rate of every test case. And moreover selection, crossover, mutation process is performed to make new population. The mentioned procedure is repeated till entire finest test cases are nominated for CT.

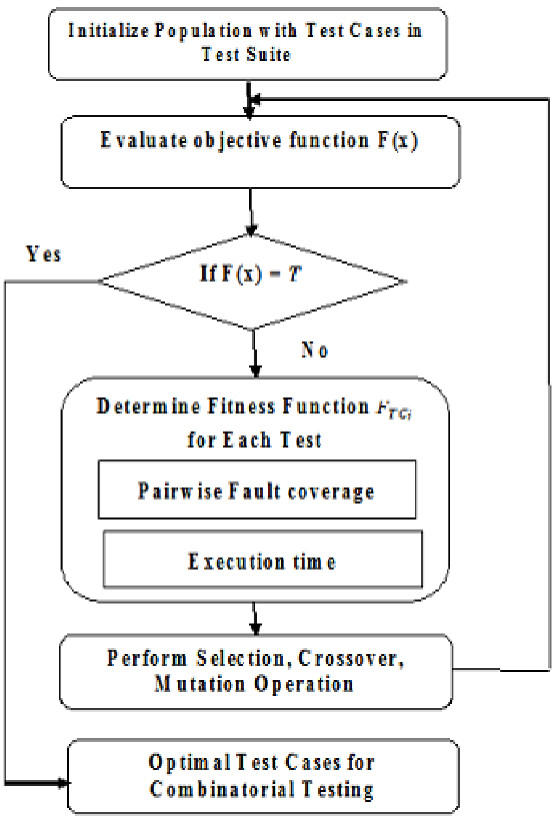

Initialize Population with Test Cases in Test Suite
Evaluate objective function F(x)
Yes
If F(x) = T
No
Determine Fitness Function F_{TC_i} for Each Test
Pairwise Fault coverage
Execution time
Perform Selection, Crossover, Mutation Operation
Optimal Test Cases for Combinatorial Testing

Figure 1 Process of Proposed Genetic Algorithm

The procedure of proposed Genetic Optimization Algorithm is presented down,

Input: Set of test cases { $TC_1, TC_2, \ldots, TC_n$ } in test suite TS, fault types covered by every test case {F1,F2,…,Fn}, execution time of every testcase {etime}.
Output: Best testcases of a testsuite

```
1.  begin
2.  Initialize the populations
3.  Define objective function F(x) of a problem
4.  for each testcase TCi in T do
5.      If(F(x)=T) then
6.          choose testcases as optimal
7.          goto 14
8.      Else
9.          Calculate fitness function by using equation (1)
10.         Held selection process
11.         Carry out crossover operation
12.         Perform mutation process
13.     endif
14. end for
15. end
```

Fault	1	2	3	4	5	6	7	8	9	10	Error
Ch1	0	1	0	1	0	0	1	0	1	0	4

A. Initialize Population

The development of GA initates population using a group of individuals. Every individual is called as chromosome. Every chromosome is a resolution to the input problem and also denoted by group of variables called genes. In this paper schoolmate dataset problem is presented with 8 testcases and 10 type of faults. Group of 8 testcases assumed as population of the problem and every testcase referred as chromosome of the problem and each corresponding fault type gene in a population represented as 1 while type of falut in a chromosome is detected during testsuite optimization process. Otherwise 0. For example, if chromosome1 covers fault types as 2,4,7,9 then chromosome1 initialized as follows.

And errors covered represents total number of fault types covered by each chromosome in a test suite.

B. Fitness Function Evaluation

Fitness function evaluation comprises two assessments called achievements of objective function and fitness functions. In this paper, objective of proposed genetic algorithm is to achieve total faults type by each testcase. When the proposed algorithm achieves the objective function is to be considered as met the threshold. Then algorithm terminates evaluation process.

Fitness function measures capability of separate test case to match with further test cases. The judgement of fitness function delivers a fitness value to every individual by examining the more objectives called maximum fault coverage and minimum execution time of each test case. The following equation (1) is used to evaluate fitness value of each test case or chromosome.

$$F_{TC_i} = \frac{\text{fault coverage}}{\text{total fault types}} * \text{executiontime} \qquad (1)$$

Where F_{TC_i} represents fitness function each test case, faults coverage represents every test case gains sum of fault types in each generation and execution time denotes time taken for identifying fault types in test case. For example fitness of chromosome1 is evaluated based on equation (1) with the inputs of fault coverage is 4 and total fault type is 10 and execution time is 7, then fitness of chromosome1 is

$$F_{TC_1} = \frac{4}{10} * 7 = 2.8$$

Similarly fitness of all chromosomes are evaluated.

C. Selection

The selection operation selects the best chromosomes based on higher fitness probability value from initial population to produce new population for next generation. The probability of each chromosome is mathematically calculated as follows,

$$P_{TC_i} = \frac{F_{TC_i}}{\sum_{i=1}^{n} F_{TC_i}} \qquad (2)$$

From (2), F_{TC_i} represents fitness value of each chromosome

and roulette wheel method used for reproduction. Now acumulative probability C_i values are measured and to produce arbitrary number R from 0 to 1 for each chromosome in a testsuite and compare if arbitrary number R_i is superior than C_i and slighter than C_{i+1} suddenly selects chromosome of i+1 position as a best chromosome for next generation[10].

D. Crossover

Genetic operations are applied on selected best individuals to produce new parents. Based on objective function and input data, type of crossover is suggested. In this paper, uniform crossover is proposed to produce new chromosomes from parent chromosomes. After reproduction of new chromosomes, type of faults covered by each chromosome and total number of errors covered by each chromosome is updated for next iteration. In this paper, we select parental chromosome randomly for mating process and are self-controlled by means of crossover ratio (ρc). Now we assume the crossover ratio is 0.25 and to produce random number for each chromosome. Now crossover ratio is compared with produced random value of each chromosome and select the corresponding chromosome as a parent chromosome for crossover process if produced random value of a chromosome is minor than 0.25. Afterwards, genes of parent chromosomes are combined to form new chromosome based on uniform crossover method. For example, we consider

Parent1

1	0	0	0	1	0	1	0	0	0

Parent2

0	0	1	0	1	0	0	0	1	1

And after making uniform crossover between parent1 and parent2, the resultant child chromosomes of parent1 and parent2 are

1	0	1	0	1	0	1	0	1	1

E. Mutation

Quantity of genes in the chromosomes are nominated arbitrarily for mutation procedure [10]. Mutation procedure is done by substituting the gene at arbitrary position with a fresh value. For this procedure, first we need compute the whole length of genes in the population using equation (3).

Total_gene = size of chromosome * size of population (3)

Quantity of chromosomes take mutations in a population is determined by the mutation rate (ρm) using equation (4).

Number of mutations = ρm * Total_gene (4)

In this paper, we consider ρm is 0.1 and total_gene is 80 then 8 genes will be mutated. Now we produce arbitrary number between 1 to Total_gen and if produced arbitrary number is lesser than ρm then the gene at the arbitrary position of population should be substituted with fresh value. After mutation process we have one completed generation of genetic algorithm and evaluate the objective function for next generation. Software tester performs the combinatorial testing and monitors every product lines in the source program with the optimal test cases. The SchoolMate dataset is taken from https://sourceforge.net/projects/schoolmate/?source=directory. Table 1 shows sample input test cases and faults type covered by each test case represented as 1 while specific faults type detected and its execution time and initially total number of errors covered by each test

Table 1 : Sample Input data

Faults	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	Errors	Exe time
Testcase												
TC1		1		1			1		1		4	7
TC2	1		1								2	4
TC3	1				1		1	1			4	5
TC4		1		1					1		3	4
TC5			1			1				1	3	4
TC6		1					1				2	4
TC7			1			1		1			3	4

IV. EXPERIMENTAL SETTINGS

The experimental evaluation of the proposed Genetic algorithm technique is implemented in Java Language using schoolmate data set. The SchoolMate dataset comprises the number of open-source programs to perform software quality test. The open-source program contains the number case in a test suite.

Parameter Selection:

Number of test cases	= 8
Total faults type	= 10
Total iterations	= 10
Ratio of crossover	= 0.25
Ratio of mutation	= 0.1

V. EVALUATION OF RESULTS AND DISCUSSION

A. Optimized test cases

Figure 2 to Figure 6 shows the implementation of proposed genetic algorithm with sample inputs, outcomes of each generations and optimized resultant testcases in testsuite. This result shows the efficiency of proposed genetic algorithm technique. Among 8 input testcases, the proposed genetic algorithm optimized only 4 testcases to achieve the objective of a problem.

B. Comparative result analysis

Efficieny of projected technique is related with two existing approaches explicitly Fuzzy Clustering [1] and test-suite reduction model based on the subtraction operation (TSRSO) [2]. To verify the results of suggested techniques, the above existing approaches are executed in same configurations of projected technique. An efficiency of proposed algorithm is estimated using the Test Suite Reduction Rate is denoted as TSRR which measured in terms of percentage, Computation Time denoted as CT measured in terms of millisecods, Fault Coverage Rate denoted as CR measured in terms of percentage and testing cost denoted as TC measured in terms of milliseconds. TSRR is determined using the following equation (14)

$$TSRR = \frac{optimized\ number\ of\ test\ cases\ in\ test\ suite}{n} * 100 \qquad (14)$$

Which results better test suite reduction rate for proposed GA compared with existing Fuzzy clustering [1] and TSRSO [2] and comparative results of TSRR shown in Figure 7.
CT is formulated using the below equation (15).

$$CT = n * time(OT) \qquad (15)$$

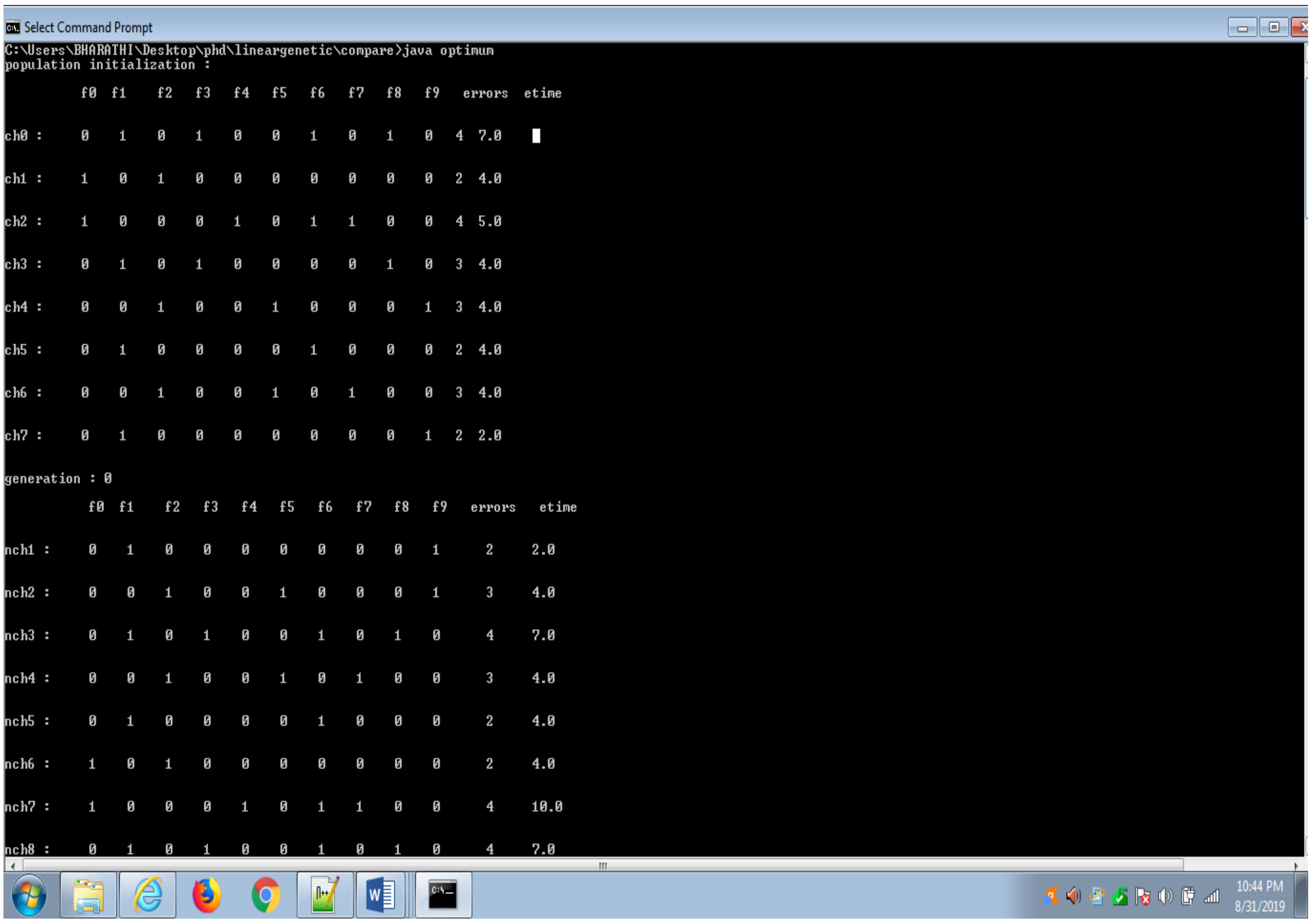
Where n refers total number of input test cases and time refers total time required for optimizing test case. By using this equation (15) proposed GA requires minor computational time compared with existing Fuzzy clustering [1] and TSRSO [2] and comparative results of CT shown in Figure 8.
CR is mathematically designed using the following equation (16)

$$CR = \frac{Number\ of\ Faults\ Covered\ by\ Optimized\ Test\ Suite}{Size\ of\ Software\ Product\ Lines} * 100 \qquad (16)$$

Proposed Genetic Algorithm delivers higher fault coverage rate than existing techniques and comparative results of CR shown in Figure 9. TC is formulated using (17) for determining the total time offered to optimize test cases in a test suite.

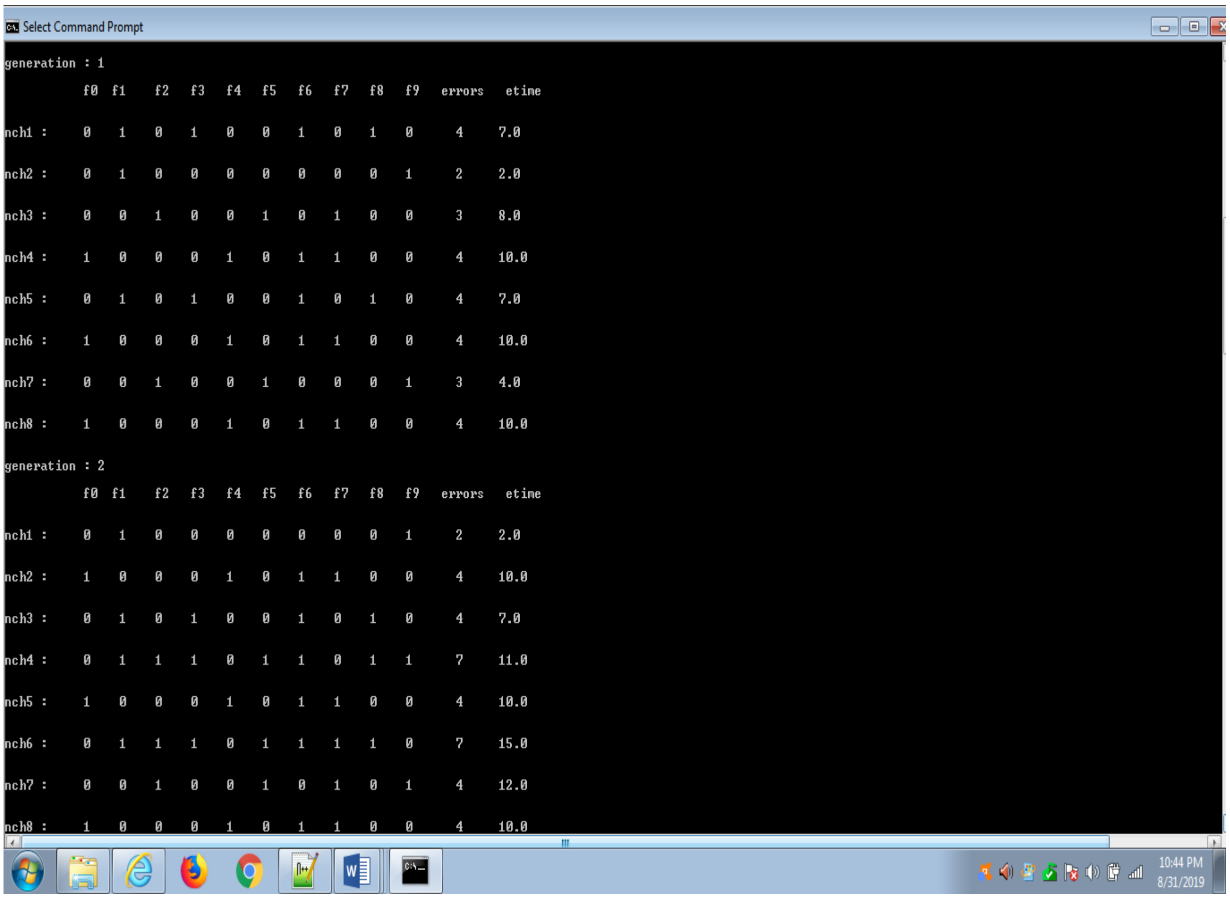
TC=Number of Software Product Lines * T (testing) (17)

Using equation (17) Proposed Genetic Algorithm Technique needs less testing cost than other methods and comparative results of TC shown in Figure 10 using graphs.



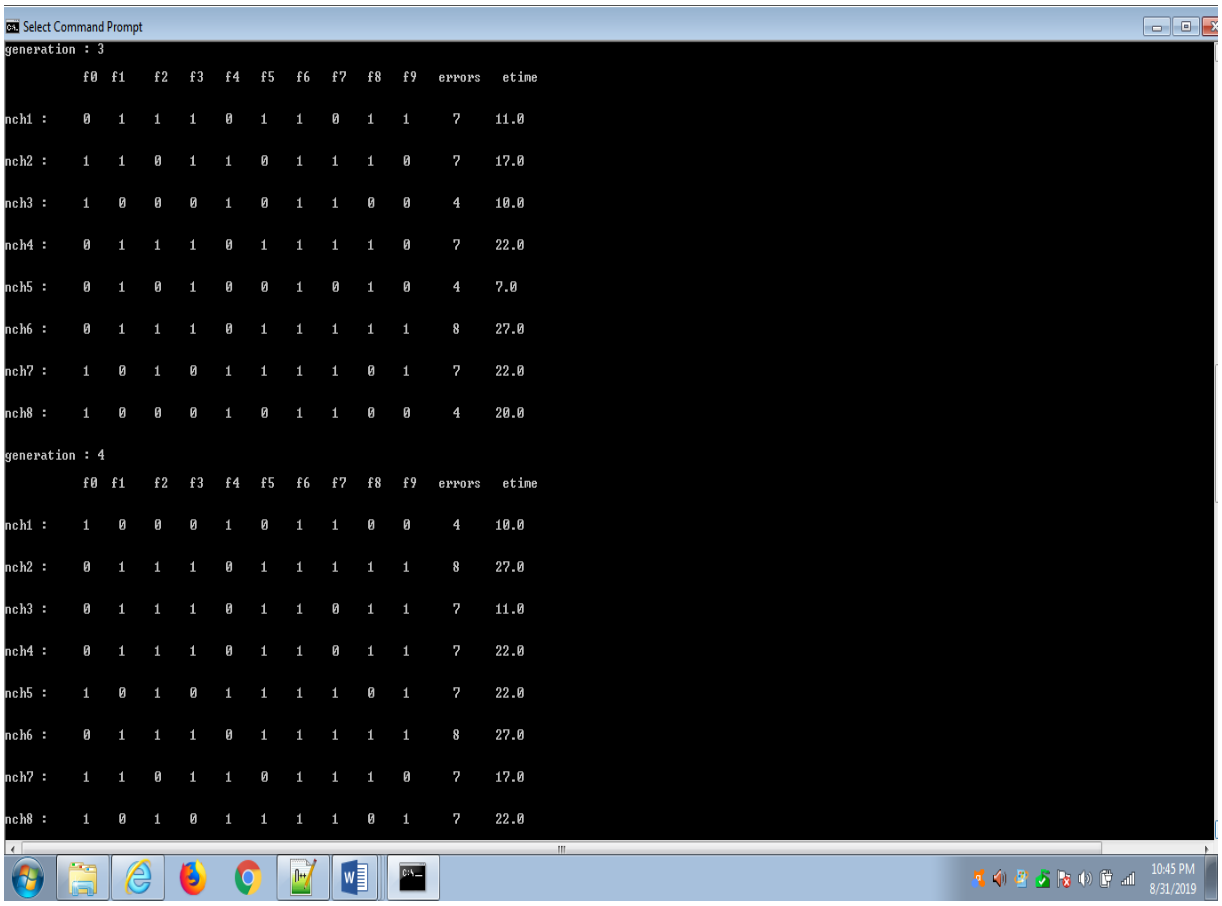
```
C:\Users\BHARATHI\Desktop\phd\lineargenetic\compare>java optimum
population initialization :

         f0  f1   f2   f3   f4   f5   f6   f7   f8   f9   errors  etime

ch0 :    0   1    0    1    0    0    1    0    1    0   4  7.0

ch1 :    1   0    1    0    0    0    0    0    0    0   2  4.0

ch2 :    1   0    0    0    1    0    1    1    0    0   4  5.0

ch3 :    0   1    0    1    0    0    0    0    1    0   3  4.0

ch4 :    0   0    1    0    0    1    0    0    0    1   3  4.0

ch5 :    0   1    0    0    0    0    1    0    0    0   2  4.0

ch6 :    0   0    1    0    0    1    0    1    0    0   3  4.0

ch7 :    0   1    0    0    0    0    0    0    0    1   2  2.0

generation : 0
          f0  f1   f2   f3   f4   f5   f6   f7   f8   f9   errors   etime

nch1 :    0   1    0    0    0    0    0    0    0    1      2      2.0

nch2 :    0   0    1    0    0    1    0    0    0    1      3      4.0

nch3 :    0   1    0    1    0    0    1    0    1    0      4      7.0

nch4 :    0   0    1    0    0    1    0    1    0    0      3      4.0

nch5 :    0   1    0    0    0    0    1    0    0    0      2      4.0

nch6 :    1   0    1    0    0    0    0    0    0    0      2      4.0

nch7 :    1   0    0    0    1    0    1    1    0    0      4      10.0

nch8 :    0   1    0    1    0    0    1    0    1    0      4      7.0
```

Figure : 2 Population Initialization and Chromosome Values of Generation 0



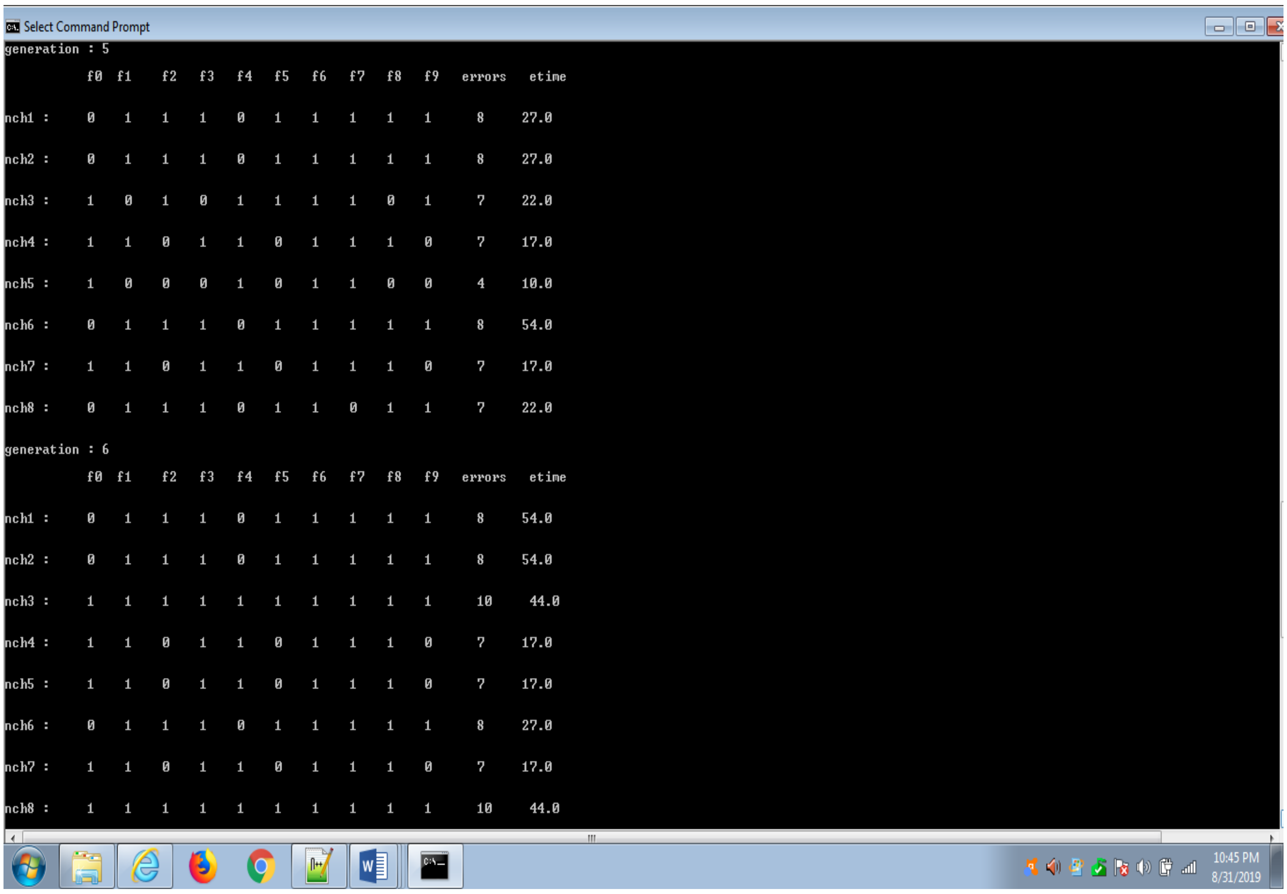
```
generation : 1
          f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :    0   1    0   1   0   0   1   0   1   0      4      7.0

nch2 :    0   1    0   0   0   0   0   0   0   1      2      2.0

nch3 :    0   0    1   0   0   1   0   1   0   0      3      8.0

nch4 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch5 :    0   1    0   1   0   0   1   0   1   0      4      7.0

nch6 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch7 :    0   0    1   0   0   1   0   0   0   1      3      4.0

nch8 :    1   0    0   0   1   0   1   1   0   0      4      10.0

generation : 2
          f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :    0   1    0   0   0   0   0   0   0   1      2      2.0

nch2 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch3 :    0   1    0   1   0   0   1   0   1   0      4      7.0

nch4 :    0   1    1   1   0   1   1   0   1   1      7      11.0

nch5 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch6 :    0   1    1   1   0   1   1   1   1   0      7      15.0

nch7 :    0   0    1   0   0   1   0   1   0   1      4      12.0

nch8 :    1   0    0   0   1   0   1   1   0   0      4      10.0
```

Figure : 3 Output Chromosomes of Generation 1 and 2



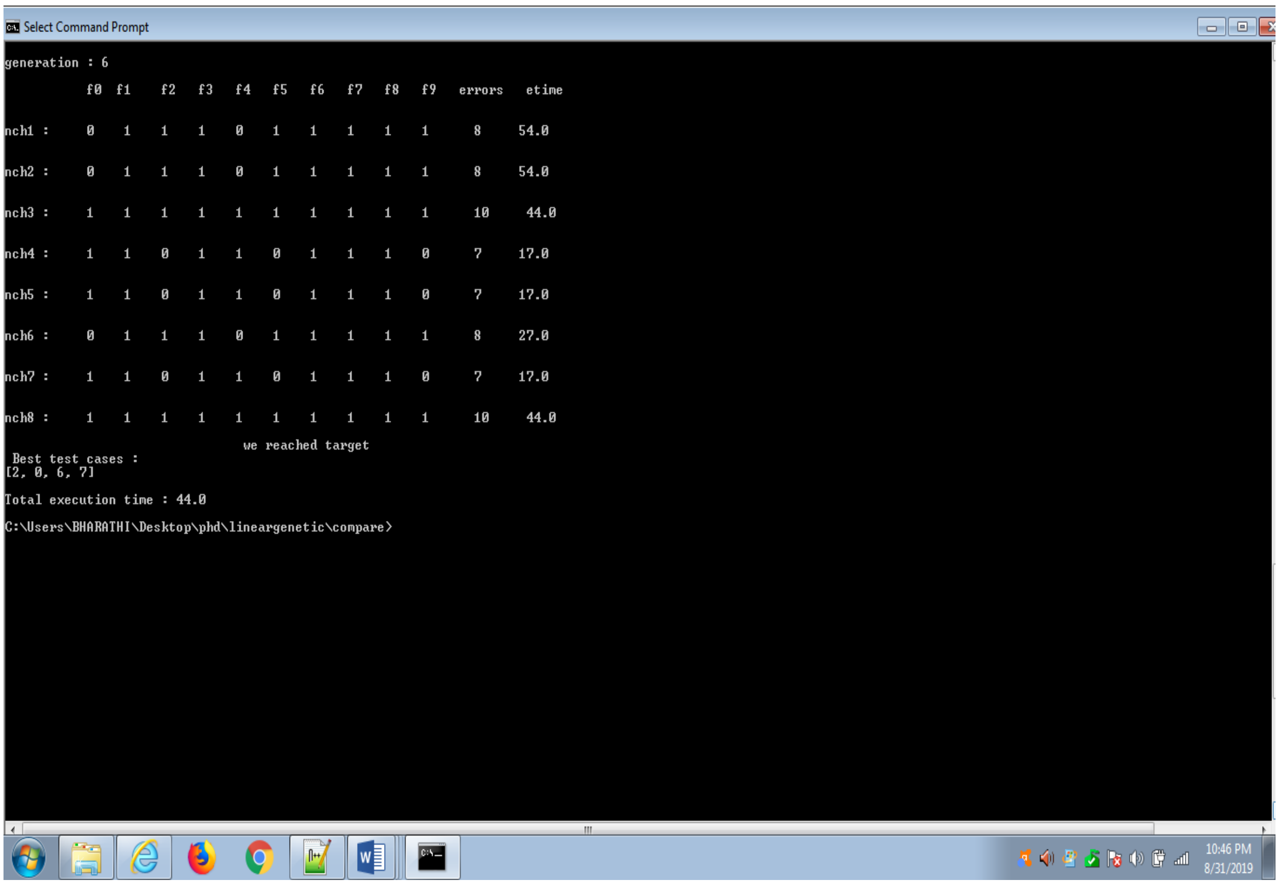
```
generation : 3
          f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :    0   1    1   1   0   1   1   0   1   1      7      11.0

nch2 :    1   1    0   1   1   0   1   1   1   0      7      17.0

nch3 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch4 :    0   1    1   1   0   1   1   1   1   0      7      22.0

nch5 :    0   1    0   1   0   0   1   0   1   0      4      7.0

nch6 :    0   1    1   1   0   1   1   1   1   1      8      27.0

nch7 :    1   0    1   0   1   1   1   1   0   1      7      22.0

nch8 :    1   0    0   0   1   0   1   1   0   0      4      20.0

generation : 4
          f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :    1   0    0   0   1   0   1   1   0   0      4      10.0

nch2 :    0   1    1   1   0   1   1   1   1   1      8      27.0

nch3 :    0   1    1   1   0   1   1   0   1   1      7      11.0

nch4 :    0   1    1   1   0   1   1   0   1   1      7      22.0

nch5 :    1   0    1   0   1   1   1   1   0   1      7      22.0

nch6 :    0   1    1   1   0   1   1   1   1   1      8      27.0

nch7 :    1   1    0   1   1   0   1   1   1   0      7      17.0

nch8 :    1   0    1   0   1   1   1   1   0   1      7      22.0
```

Figure : 4 Output Chromosomes of Generation 3 and 4



```
generation : 5
           f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :     0   1    1   1   0   1   1   1   1   1      8      27.0

nch2 :     0   1    1   1   0   1   1   1   1   1      8      27.0

nch3 :     1   0    1   0   1   1   1   1   0   1      7      22.0

nch4 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch5 :     1   0    0   0   1   0   1   1   0   0      4      10.0

nch6 :     0   1    1   1   0   1   1   1   1   1      8      54.0

nch7 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch8 :     0   1    1   1   0   1   1   0   1   1      7      22.0

generation : 6
           f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :     0   1    1   1   0   1   1   1   1   1      8      54.0

nch2 :     0   1    1   1   0   1   1   1   1   1      8      54.0

nch3 :     1   1    1   1   1   1   1   1   1   1      10      44.0

nch4 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch5 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch6 :     0   1    1   1   0   1   1   1   1   1      8      27.0

nch7 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch8 :     1   1    1   1   1   1   1   1   1   1      10      44.0
```

Figure : 5 Output Chromosomes of Generation 5 and 6



```
generation : 6
           f0  f1   f2  f3  f4  f5  f6  f7  f8  f9   errors   etime

nch1 :     0   1    1   1   0   1   1   1   1   1      8      54.0

nch2 :     0   1    1   1   0   1   1   1   1   1      8      54.0

nch3 :     1   1    1   1   1   1   1   1   1   1      10      44.0

nch4 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch5 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch6 :     0   1    1   1   0   1   1   1   1   1      8      27.0

nch7 :     1   1    0   1   1   0   1   1   1   0      7      17.0

nch8 :     1   1    1   1   1   1   1   1   1   1      10      44.0
                              we reached target
 Best test cases :
[2, 0, 6, 7]

Total execution time : 44.0

C:\Users\BHARATHI\Desktop\phd\lineargenetic\compare>
```

Figure : 6 Output of Optimized Testcase of Proposed Genetic Algorithm

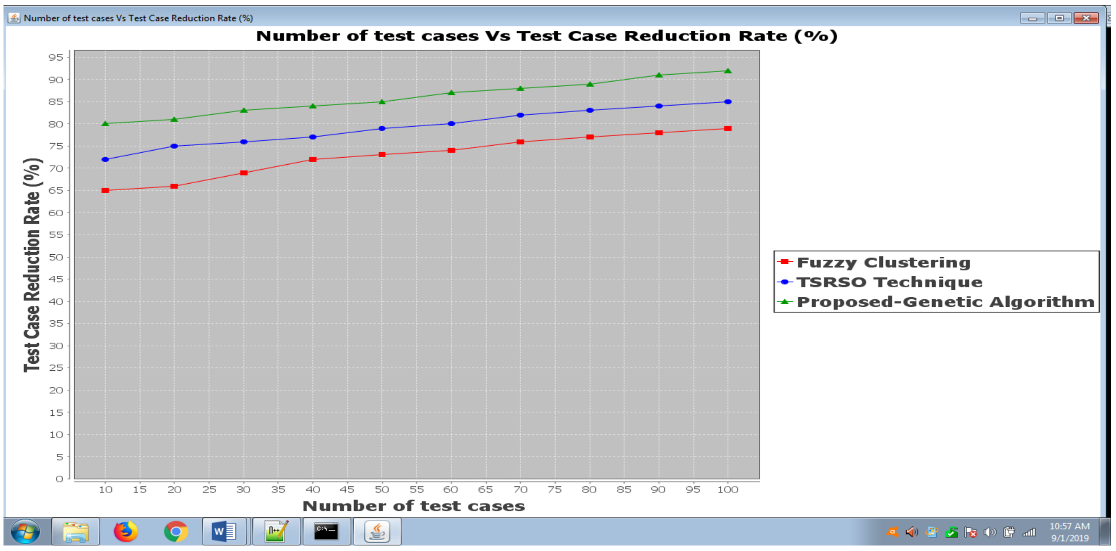


Figure 7 Efficiency of Test Suite Reduction Rate

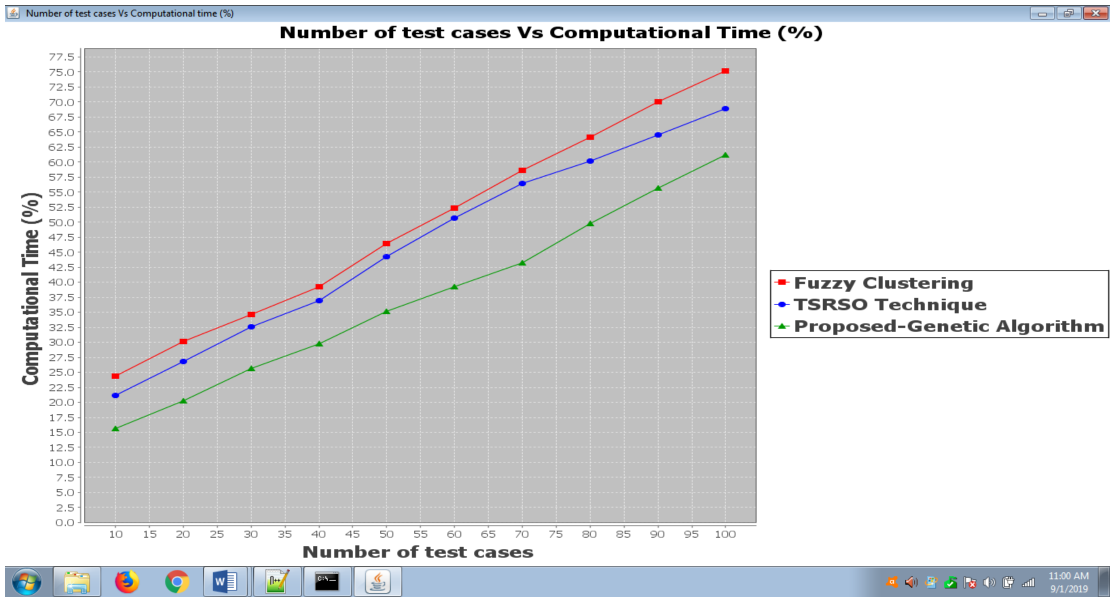


Figure 8 Efficiency of Computational time

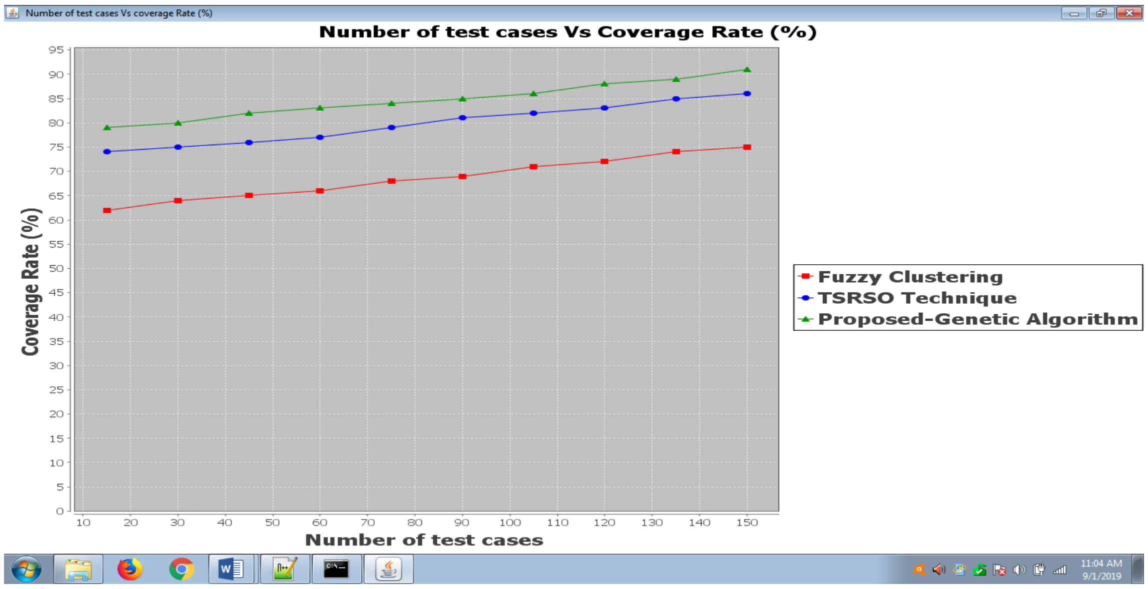


Figure 9 Efficiency of Coverage Rate

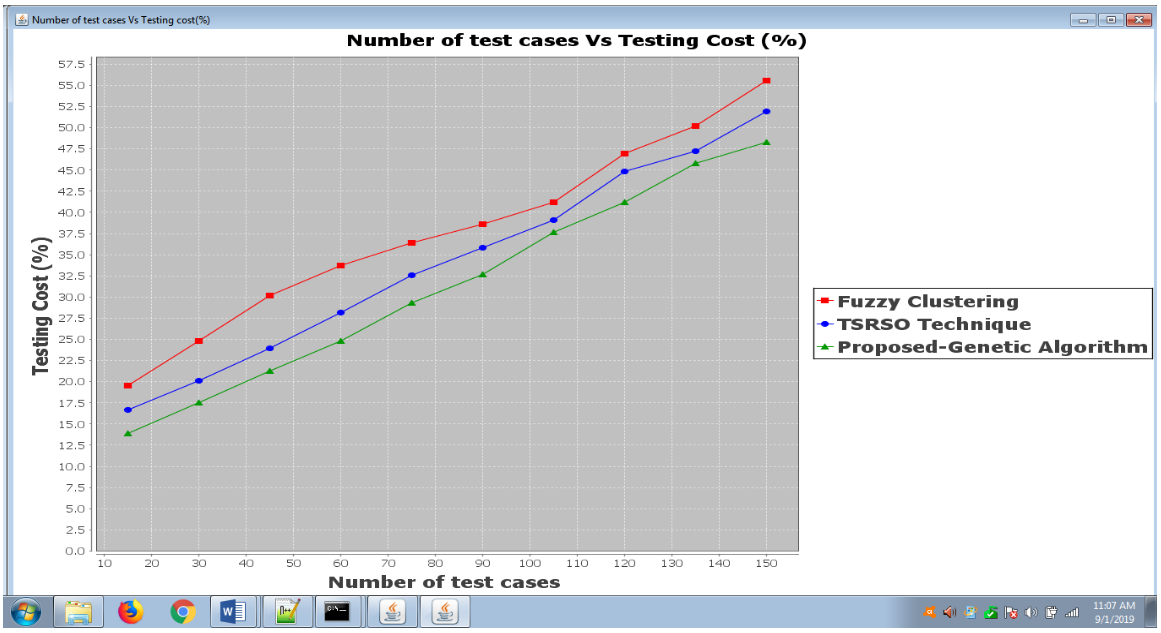


Figure 10 Efficiency of Testing Cost

VI. CONCLUSION

Genetic algorithm is a gifted technique for optimizing the test cases in a test suite. In this research work, Proposed Genetic algorithm is presented, implemented in effective manner and also compared with other techniques. Anyhow, Proposed Genetic Algorithm optimizes the test suite highly. Our research work shows the comparison results which illustrates higher test suite reduction rate, lower computational time, maximum faults test coverage and minor testing cost. Proposed Genetic Algorithm shrinks the test cases in test suite however it does not promise finest resolution for optimization problem due to the procedure of accidentally mutating genes of chromosome in mutation process which causes reduction rate of test suite. Also the Proposed GA proceeds several process like evaluation of fitness function, selection of parents, crossover of genes, and mutation of specific genes. As a result the Proposed Genetic Algorithm required extra computational time for test suite optimization problem. These complications will be experimented in our further research.

REFERENCES

1. Gaurav Kumar, Pradeep Kumar Bhatia, “Software testing optimization through test suite reduction using fuzzy clustering”, CSI Transactions on ICT, Springer, Volume 1, Issue 3, pp 253–260, September 2013
2. Qing Shen, Yunliang Jiang, Jungang Lou, “A new test suite reduction method for wearable embedded software”, Computers and Electrical Engineering, Elsevier, Volume 61, Pages 116–125, 2017
3. Bestoun S. Ahmed, “Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing”, Engineering Science and Technology, an International Journal, Volume 19, Pages 737–753, 2016
4. Andrea Calvagna, Angelo Gargantini, “A Formal Logic Approach to Constrained Combinatorial Testing”, Journal of Automated Reasoning, Springer, Volume 45, Issue 4, Pages 331–358, December 2010
5. Xiaofang Qi, Jun He, Peng Wang, Huayang Zhou, “Variable strength combinatorial testing of concurrent programs”, Frontiers of Computer Science, Springer, Volume 10, Issue 4, pp 631–643, August 2016
6. Abhishek Pandey, Soumya Banerjee, “Test Suite Optimization Using Chaotic Firefly Algorithm in Software Testing”, International Journal of Applied Metaheuristic Computing, Volume 8, Issue 4, 2017
7. Manju Khari, Prabhat Kumar, “An Effective Meta-Heuristic Cuckoo Search Algorithm for Test Suite Optimization”, Informatica, Volume 41, Pages 363–377, 2017
8. Shilpi Singh, Raj Shree “A combined approach to optimize the test suite size in regression testing”, CSI Transactions on ICT, Volume 4, Issue 2, Pages 73–78, December 2016
9. Luciano Soares de Souza, Ricardo Bastos Cavalcante Prudencio and Flavia A. de Barros, “A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection”, Journal of the Brazilian Computer Society, Volume 21, Issue 19, Pages 1-20, 2015
10. Lubna Zaghlul Bashir “Solve Simple Linear Equation using Evolutionary Algorithm”, World Scientific News, WSN 19, 2015, 148-167.

AUTHORS PROFILE

M.Bharathi is a Research Scholar in the Department of Computer Science, Periyar University, Salem, Tamilnadu, India. She had completed B.Sc(Computer Science) at Periyar University in the year of 1999. She had completed MCA during the year of 2005 and M. Phil (Computer Science) degree in 2008 and she has passed Tamilnadu State Eligibility Test (TNSET) 2012 of Bharathiyar University, Tamilnadu. Her research interests are software engineering and software testing. She has specialized in the area of system software, Data mining, Digital image processing and data Science. She has fourteen years teaching experience in the field of computer science. She has published several articles in peer reviewed and reputed journals. She has attended and presented several papers in various conferences organized by well esteemed institutions.

Dr.V.Sangeetha is working as an Assistant professor in the Department of Computer Science, Periyar University college, Pappereddipatty, Dharmapuri, Tamilnadu, India. She has completed her Msc(cs) during the year of 2001 and M.Phil(Computer Science) during the year 2004 and completed her Ph.D in the year of 2014. She has been specialized in this area of software engineering, Data mining, Compiler design, and programming languages, python and data science. She has attended many national and international conferences and presented several papers. She has fifteen year experience in the field of computer science. Delivered Guest Lecture at various Engineering and Arts Colleges. She has supervised several research scholars and her aim to produce more number of valuable research works to this environment.