Optimum Test Suite Using Fault-Type Coverage-Based Ant Colony Optimization Algorithm

M. Bharathi, Department of Computer Science, Government Arts and Science College, Pennagaram, India*

ABSTRACT

Software product lines (SPLs) cover a mixture of features for testing software application programs (SPA). Testing cost reduction is a major metric of software testing. In combinatorial testing (CT), maximization of fault type coverage and test suite reduction plays a key role to reduce the testing cost of SPA. Metaheuristic genetic algorithms (GA) do not offer the best outcome for test suite optimization problems due to mutation operation and more required computational time. So, fault-type coverage-based ant colony optimization (FTCBACO) algorithm is offered for test suite reduction in CT. FTCBACO algorithm starts with test cases in test suite and assigns separate ants to each test case. Ants elect the best test cases by updating of pheromone trails and selection of higher probability trails. Best test case path of ant with least time are taken as optimal solutions for performing CT. Hence, FTCBACO technique enriches reduction rate of test suite and minimizes computational time of reducing test cases efficiently for CT.

KEYWORDS

Combinatorial Testing, Computational Time, Fault-Type Coverage-Based Ant Colony Optimization, Software Product Lines, Test Cases, Test Suite Optimization, Test Suite Reduction Rate, Testing Cost

1. INTRODUCTION

Software testing is one of the most eminent phase of software development. Main target of software testing is to recognize faults and imperfections arise in software product lines under development. Currently, software testing utilizes additional time and cost consumed on software development. While testing time decreases, testing cost may decrease rapidly. Now a days, software may be published without being checked properly due to market pressure and preplan of software developers to save time and reduce cost spent on software development. Hence, software testers should construct best test cases that discover most of the faults and defects in the software product within the scheduled testing time. Interactions between the pair of features play a key role for constructing best test cases. Combinatorial testing is one of software testing technique to find the interactions between the pair of features in software product line. Additionally, CT reduces the quantity of pair of feature combinations required to check software system excellence as compared to exhaustive testing. Test suite optimization plays essential role to reduce the testing cost of SPA without corrupting their quality factors in CT. So, this investigation work aims test suite optimization proficiently for increasing the ability of CT.

To conduct this research, we have found two different heuristic algorithms called algorithm (Li at el., 2007) Greedy Strategy based and (M.Bharathi & V.Sangeetha, 2019) Genetic algorithm(GA) as baseline algorithms. In Greedy approach (Li at el., 2007), test cases are assembled in downward order
based on their fault coverage ability and start with test cases that cover maximum amount of faults till either all the faults are reported or test adequacy condition is met. Genetic algorithm (M.Bharathi & V.Sangeetha, 2019) reduces the test cases in test suite yet it does not deliver optimal solution for test suite optimization problem due to mutation operation. Moreover, GA involves several operations like fitness evaluation, selection, crossover, mutation. As an outcome, high computational time is required for test suite optimization problem.

To resolve the above quoted issues in test suite optimization problem using GA, novel Fault-Type Coverage Based Ant Colony Optimization (FTCBACO) algorithm Technique is proposed. FTCBACO algorithm is constructed based on highest faults-type coverage analysis and find an efficient optimum results. The main contribution of proposed technique is,

- To enrich the capability of test suite reduction for CT.
- To rise the test suite reduction rate and to cut the computational time of CT compared with genetic algorithm.
- To optimize test cases in test suite in terms of utmost faults type coverage and least execution time.

The research work of this paper is drawn as follows. An overview of related work is presented in Section 2. Section 3 explains proposed FTCBACO for test suite optimization problem. Section 4 exhibits the experimental settings. Section 5 reports the outcomes and discussions. In section 6, we conclude this paper.

2. RELATED WORKS

A several investigation works have been designed to resolve the test suite optimization problems in combinatorial testing. proposed Similarity-based test suite reduction for model-based testing was introduced in (Ana Emilia Victor Barbosa Coutinho et al., 2016). But, this strategy reported poor test suite reduction rate. A regression based test case selection method with aim of shrinking the quantity of test cases for regression testing was introduced in (Tamal Sen & Rajib Mall, 2012). But, it required more time to optimize the test cases in a test suite. Several procedures have reviewed for test suite minimization for improving the quality of software metrics in (Shilpi Singh & Raj Shree, 2016).

A hierarchical clustering approach was proposed in (Fayaz Ahmad Khan et al., 2015) for increasing the minimization of test cases in a test suite. However, an optimized test suite does not deliver complete fault reportage. A fuzzy logic based approach was presented in (Haider at el., 2014) for accomplishing test suite optimizations with multi objective. But, test suite optimization demanded higher time complexity. The priority based test case reduction was presented cost in (Jyoti Prakash Rout at el., 2013) with the aim of reducing the testing. But, more testing cost was needed. A unified method was presented in (Srividhya Jeyaprakasha & K.Alagarsamy, 2015) for reducing the test cases and aimed to report the entire faults with least time. But, computational time of test suite reduction was high.

In (Gong Dandan at el., 2013), a two-step test-suite reduction approach was intended to minimize the test cases with the performance of higher fault-localization. But, performance of test-suite reduction was not high. In (Ning Li at el., 2016), an effective strategy to minimize test suite using spectrum-based fault localization was proposed. But, high computational time required for testing excellence of software program. An Artificial Bee Colony algorithm was designed in (Soma Sekhara Babu Lam at el., 2012) for automated generation of independent paths and test suite optimization. But, reduced test suite not covered several pair of features. Adaptive neuro-fuzzy inference system was developed in (Zeeshan Anwar et al., 2015) with grid partitioning to accomplish test suite minimization. However, computational time was not sufficient.

Fault based Genetic algorithm(GA) approach was developed in (M.Bharathi & V.Sangeetha, 2019) to optimize test case for enhancing software quality. But, GA approach demanded more computational
time due to mutating bits. Test suite reduction was offered in (D.Jeffrey & Neelam Gupta, 2005) based on selective redundancy. In this technique, redundant test cases are identified using branch coverage data and duplicate test cases are removed. Finally, this technique slightly reduced in test suite size. HGS algorithm was suggested in (Harrold et al., 1993). HGS algorithm removed redundant test cases and offered smaller test suite in size. Hybrid algorithm for Pareto efficient multi-objective test suite minimization was projected in (Yoo S & Harman M, 2010) In this hybrid algorithm, Greedy technique is merged with Genetic algorithm for attaining minimized test suite.

A greedy approach was introduced in (P.Harris & N.Raju, 2015) for coverage-based test suite reduction. Reduced test suite size was obtained based on code coverage criteria. Then its performance was compared with bi-objective greedy techniques as well as HGS. TestFilter for reduction of test cases was proposed in (S.Khan at el., 2006). Weights assigned to each test case based on statement coverage criteria. Weight denoted as total existences of each test case. Non-redundant test cases are selected based on weights. Originally, test cases with greater weight are nominated. Then test cases with lesser weight are nominated till all the needs are fulfilled. Selected test cases are involved into reduced test suite set. Lastly TestFilter technique presented reduction in test suite size and computational cost. A safe, efficient regression test selection technique was introduced in (Rothermel G. & Harrold MJ, 1997). Control flow graph was used in this approach and obtained in generic nature. Therefore, it was used to test the software which were written in several languages.

We proposed Fault-Type Coverage Based Ant Colony Optimization (FTCBACO) algorithm technique to rectify the above specified existing complications in test suite minimization problem.

3. IMPLEMENTATION OF FTCBACO

Fault-Type Coverage Based Ant Colony Optimization (FTCBACO) Technique is organized to resolve the problems in genetic Algorithm and Greedy approach for test suite optimization problem CT. FTCBACO constructed based on metaheuristic Ant Colony Optimization (ACO) Algorithm for attaining test suite reduction rate in advanced, minimization of computational time and testing cost to expand the efficiency of CT. Now a days, computational problems are solved using Swarm intelligence (SI) techniques. ACO is built from SI techniques. The ACO algorithm is built using graphs for discovering an optimal path based on the ants behavior. In ACO algorithm, an ant selects path from its colony to food source and return to colony using its Pheromone (chemical substances). Foragers trace the path to reach food source by perceiving pheromone trail of other ants. Finally, finest path is discovered from their colony to food source.

Based on the concept of ACO algorithm, FTCBACO algorithm planned to discover optimum test cases in a test suite in terms of maximization of test suite reduction with minimum execution time. In FTCBACO algorithm, ant assigns test case as vertex and probability of test case as weight of an edge. We allotted distinct ant to every test case. Originally we nominated one ant with associated test case and added its corresponding covered faults type into set S as well as execution time, total number of faults covered is noted. In addition nominated test case marked as visited and added to best test case path. To check whether the nominated ant covered all faults type(objective) defined in a problem or not. If the nominated ant covers all faults type then we shift to next ant. Otherwise we compute probability value for all test cases and select an unvisited test case with higher probability value(edge) as best test case to reach food source(all faults type) and added into best test case path. Furthermore covered faults type added into associated set. Cumulative execution time and total number of faults covered of best test cases are calculated. The probability value of edges is computed using the pheromone value of test case dumped on path. The same procedure repeated till all the ants covered all faults type. Finally best test case path of an ant with least execution time is taken as optimal path and test cases in the optimal path considered as optimal test cases.
The data flow diagram of Fault-Type Coverage Based Ant Colony Optimization (FTCBACO) algorithm is presented in Figure 1 which demonstrates the overall process of proposed FTCBACO algorithm to solve test case reduction for combinatorial testing.

### 3.1 Procedure of FTCBACO

The procedure of proposed FTCBACO Algorithm is presented down.

**Terms used:** T-test suite, CFT-set of covered faults type by each test case, TC_1, TC_2, ..., TC_n - Set of test cases, TFC-Total number of faults type covered by each test case, ET-Execution time of each test case, TFT-Total faults type covered by a test suite, PT-Pheromone trail of each test case, P-Probability of test case, A_1, A_2, ..., A_n-set of ants, TCN-test case number, LTP-length of test case path.

**Input:** Group of test cases {TC_1, TC_2, TC_3, ..., TC_n} in a test suite
\{T\}, covered faults type (CFT) of each test case, total number faults type covered (TFC) and Execution time (ET) of each test case, TCN-test case number.

Output: optimized test cases with minimum execution time and all faults type covered.

1. Begin
2. Derive covered faults type CFT\_i, total number of faults type covered TFC\_i and Execution time ET\_i of test case TC\_i in a test suite T.
3. Set TC\_i \rightarrow A\_i where i=1,2,...n.
4. Initialize PT\_i \rightarrow 1.0, TCN \rightarrow \Phi, CFT \rightarrow \Phi, TFC \rightarrow \Phi, ET \rightarrow \Phi.
5. Define objective function F(x)=total\_faults\_type with minimum execution time to a problem
6. Initialize i \rightarrow 1, k \rightarrow i
7. for each Ai in T do
8. Set TC\_i \leftarrow TC\_k, CFT\_i \leftarrow CFT\_k, TFC\_i \leftarrow TFC\_k, ET\_i \leftarrow ET\_k, LTP \leftarrow 1
9. TCNi marked as visited (*).
10. Initialize t \rightarrow I
11. If (TFC\_i == TFT) Then
12. Goto End for
13. Else
14. While (TFC\_i != TFT)
15. Initialize j \rightarrow 1
16. for each TC\_j in T do
17. if (t== j) then
18. update pheromone PT\_j using equation (1)
19. update probability P\_j using equation (3)
20. Set j \rightarrow j + 1
21. else
22. update pheromone PT\_j using equation (2)
23. update probability P\_j using equation (3)
24. Set j \rightarrow j + 1
25. End if
26. End for
27. Find test case number t with maximum P of T except visited test cases marked as visited (*),
its associated set of covered faults type \( \{c\} \), and execution time \( e \).

28. \( \text{Set } \text{TFC}_i \leftarrow \text{TFC}_i + \text{number of new fault types covered by test case } t \) 
29. \( \text{Set } \text{TCN}_i \leftarrow \text{TCN}_i \cup \{t\} \) 
30. \( \text{Set } \text{CFT}_i \leftarrow \text{CFT}_i \cup \{c\} \) 
31. \( \text{Set } \text{ET}_i \leftarrow \text{ET}_i \cup e \) 
32. \( \text{Set } \text{LTP}_i \leftarrow \text{LTP}_i + 1 \) 
33. \( \text{End While} \) 
34. \( \text{End if} \) 
35. \( \text{Set } i \leftarrow i + 1 \) 
36. \( \text{Set } k \leftarrow i \) 
37. \( \text{End for} \) 
38. \( \text{Select ant with minimum of total execution time and minimum length test case path} \) 
39. \( \text{Consider Test cases in selected test case path of an ant as optimum test cases in } T \) 
40. \( \text{End} \)

3.2 Functions of FTCBACO

FTCBACO has two major functions for selecting next test case to achieve an objective function of the problem.

- Update pheromone
- Update probability

Update pheromone

If the ant number is same as selected test case number then the pheromone of test case is evaluated using equation (1). Otherwise evaluated using equation (2).

\[
\text{pheromone}[i] = (1 - \rho) \times \text{pheromone}[i] + \left( \frac{\text{covered faults}[i]}{\text{total faults}} \right) 
\]  \hspace{1cm} (1)

\[
\text{pheromone}[i] = (1 - \rho) \times \text{pheromone}[i] 
\]  \hspace{1cm} (2)

From equation (1) and (2), pheromone[i] represents amount of pheromone on test case and \( \rho \) represents pheromone evaporation rate and \( \frac{\text{covered faults}[i]}{\text{total faults}} \) represents amount of pheromone deposited.

Update Probability

The probability of test case is updated using equation (3) and the test case of highest probability is selected as next test case visited by the ant.
probability[i] = \frac{[\text{covered faults}[i]/\text{total faults}] * \text{execution time}[i]}{\sum_{i=1}^{n} \left[\frac{\text{covered faults}[i]/\text{total faults}}{\text{execution time}[i]} * [\text{pheromone}[i]]^{\beta}\right]^{\alpha}} (3)

4. EXPERIMENTAL SETTINGS

An investigational assessment of the proposed FTCBACO algorithm is applied in Java Language with schoolmate data set. The SchoolMate dataset covers the several number of open-source programs to investigate software quality. Each open-source program holds the numeral test cases in a test suite. The SchoolMate dataset is available in https://sourceforge.net/projects/schoolmate/?source=directory. Table 1 shows sample input test cases and faults type covered by each test case and its execution time and initially total number of faults type covered by each test.

Selection of required parameters:
- Number of test cases = 8
- Total faults type covered = 10
- Pheromone evaporation rate = 0.4
- alpha (∞) = 2
- beta (β) = 1

Table 1. Sample input Test Suite

| Test case | Execution time | Total faults type covered | Faults type |
|-----------|---------------|---------------------------|-------------|
| TC1       | 7             | 4                         | 2,4,7,9     |
| TC2       | 4             | 2                         | 1,3         |
| TC3       | 5             | 4                         | 1,5,7,8     |
| TC4       | 4             | 3                         | 2,4,9       |
| TC5       | 4             | 3                         | 3,6,10      |
| TC6       | 4             | 2                         | 1,7         |
| TC7       | 4             | 3                         | 3,6,8       |
| TC8       | 2             | 2                         | 2,10        |

5. EVALUATION OF RESULTS AND DISCUSSION

5.1 Process Of Test Case Path Selection

Table 2 represents initial pheromone distribution of each test case and its probability. Table 3 to Table 10 proves step by step implementation of proposed FTCBACO algorithm with sample inputs, process of test case path selection of each ant and optimized test path of a test case in a test suite. Outcome shows the ability of FTCBACO algorithm technique. Amongst 8 input test cases, the recommended
FTCBACO algorithm suggested only 3 test cases to attain the objective of a problem. Figure 2 shows outcome of an existing Genetic Algorithm technique and Figure 3 shows an efficient outcome of proposed FTCBACO Technique. Table 2 Shows Initial pheromone distribution and probability of sample input test suite. From Table 3 to Table 10, the term TCN refers test case number, CFT refers covered faults type, TFC refers Total faults type covered and ET refers Execution time of associated test case.

5.2 Comparative Result Analysis
Step-by-step execution outcomes of existing Greedy approach as presented in Table 11 and outcomes of Genetic algorithm as presented in Fig 2 and outcomes of FTCBACO technique is shown in Fig 3. Selection of required parameters for GA technique are:
- Number of test cases = 8
- Total faults type covered = 10
- Cross over ratio(ρc) = 0.25
- mutation rate(ρm) = 0.1

The above mentioned existing GA technique and greedy approach are implemented and executed with same configurations of proposed FTCBACO technique. While the result of proposed technique is compared with existing technique, an existing technique required more execution time than proposed technique and also 4 optimum test cases are required for achieving an objective function. But FTCBACO needed only 3 optimum test cases to achieve an objective. So we conclude that the proposed FTCBACO provides an efficient optimum results. An efficiency of FTCBACO algorithm is assessed by means of the following factors as

- Test Suite Optimization Rate
- Computational Time
- Testing cost

Table 2. Initial pheromone distribution and probability of each test case

| Testcase (TCi) | Pheromone[i] | \(\sum_{i=1}^{n} \text{pheromone}[i]\) | Probability[i] = \(\frac{\text{pheromone}[i]}{\sum_{i=1}^{n} \text{pheromone}[i]}\) |
|---------------|-------------|---------------------------------|---------------------------------|
| T C 1         | 1           | 8                               | 0                               |
| T C 2         | 1           | 8                               | 0                               |
| T C 3         | 1           | 8                               | 0                               |
| T C 4         | 1           | 8                               | 0                               |
| T C 5         | 1           | 8                               | 0                               |
| T C 6         | 1           | 8                               | 0                               |
| T C 7         | 1           | 8                               | 0                               |
| T C 8         | 1           | 8                               | 0                               |
Table 3. Outcomes of Ant 1

| Ant 1 | Pheromone[i] | EV | [pheromone[i]] β | EV*[pheromone[i]] β | Probability[i] |
|-------|--------------|----|----------------|-------------------|---------------|
| TC1   | 1.0          | 7.84| 1.0            | 7.84              | 0.5724(AV)    |
| TC2   | 0.6          | 0.64| 0.6            | 0.384             | 0.028         |
| TC3   | 0.6          | 4.0 | 0.6            | 2.4               | 0.1752(*)     |
| TC4   | 0.6          | 1.44| 0.6            | 0.864             | 0.0631        |
| TC5   | 0.6          | 1.44| 0.6            | 0.864             | 0.0631        |
| TC6   | 0.6          | 0.64| 0.6            | 0.384             | 0.028         |
| TC7   | 0.6          | 1.44| 0.6            | 0.864             | 0.0631        |
| TC8   | 0.6          | 0.16| 0.6            | 0.096             | 0.007         |

TC1 already visited marked as AV. So select next higher probability marked as (*). TCN={1,3} CFT={1,2,4,5,7,8,9} TFC = 7 ET = 7+5=12

Iteration 2

| TC1   | 0.6          | 7.84| 0.6            | 4.0704            | 0.2001(AV)    |
| TC2   | 0.36         | 0.64| 0.36           | 0.2304            | 0.0098        |
| TC3   | 0.76         | 4.0 | 0.76           | 3.04              | 0.1293(AV)    |
| TC4   | 0.36         | 1.44| 0.36           | 0.5184            | 0.022         |
| TC5   | 0.36         | 1.44| 0.36           | 0.5184            | 0.022         |
| TC6   | 0.36         | 0.64| 0.36           | 0.2304            | 0.0098        |
| TC7   | 0.36         | 1.44| 0.36           | 0.5184            | 0.022(*)      |
| TC8   | 0.36         | 0.16| 0.36           | 0.0576            | 0.0024        |

TCN={1,3,7} CFT={1,2,3,4,5,6,7,8,9} TFC = 9 ET =12+4=16

Iteration 3

| TC1   | 0.36         | 7.84| 0.36           | 2.8224            | 0.0946(AV)    |
| TC2   | 0.216        | 0.64| 0.216          | 0.1382            | 0.0046        |
| TC3   | 0.456        | 4.0 | 0.456          | 1.824             | 0.0611(AV)    |
| TC4   | 0.216        | 1.44| 0.216          | 0.311             | 0.0104        |
| TC5   | 0.216        | 1.44| 0.216          | 0.311             | 0.0104(*)     |
| TC6   | 0.216        | 0.64| 0.216          | 0.1382            | 0.0046        |
| TC7   | 0.516        | 1.44| 0.516          | 0.743             | 0.0249(AV)    |
| TC8   | 0.216        | 0.16| 0.216          | 0.0346            | 0.0012        |

TCN={1,3,7,5} CFT={1,2,3,4,5,6,7,8,9,10} TFC = 10 ET = 16+4=20 Ant 1 reached 10 faults type. So Ant 1 complete its process.

5.2.1 Test Suite Optimization Rate

Test Suite Optimization Rate is represented as TSOR which computed in terms of percentage. TSOR is mathematically formulated with the equation (14) as follows,

\[
\text{TSOR} = \left( \frac{\text{coveredfailures}}{\text{total faults}} \right) \times \text{executiontime} \times \left( \frac{\text{pheromone[i]}}{\text{pheromone}[i]} \right)^\beta \times \text{Probability}[i]
\]
**Table 4. Outcomes of Ant 2**

| TCi | Pheromone[i] | EV \[\text{covered faults} / \text{total faults} \times \text{execution time}\] | [pheromone[i]] ^ {\beta} | EV*[pheromone[i]] ^ {\beta} | Probability[i] |
|-----|--------------|---------------------------------|-----------------|-----------------|----------------|
| Iteration 1 |
| TC1 | 0.1296 | 7.84 | 0.1296 | 1.0161 | 0.0313(*) |
| TC2 | 0.2778 | 0.64 | 0.2778 | 0.1778 | 0.0055(AV) |
| TC3 | 0.1642 | 4.0 | 0.1642 | 0.6568 | 0.0202 |
| TC4 | 0.0778 | 1.44 | 0.0778 | 0.112 | 0.0034 |
| TC5 | 0.2578 | 1.44 | 0.2578 | 0.3712 | 0.0114 |
| TC6 | 0.0778 | 0.64 | 0.0778 | 0.0498 | 0.0015 |
| TC7 | 0.1858 | 1.44 | 0.1858 | 0.2676 | 0.0082 |
| TC8 | 0.0778 | 0.16 | 0.0778 | 0.0124 | 4.0E-4 |

TC2 already visited marked as AV. So select next higher probability marked as (*). TCN=[2,1] CFT={1,2,3,4,7,9} TFC=6 ET=4+7=11

| Iteration 2 |
| TC1 | 0.4778 | 7.84 | 0.4778 | 3.746 | 0.1006(AV) |
| TC2 | 0.1667 | 0.64 | 0.1667 | 0.1067 | 0.0029(AV) |
| TC3 | 0.0985 | 4.0 | 0.0985 | 0.394 | 0.0106(*) |
| TC4 | 0.0467 | 1.44 | 0.0467 | 0.0672 | 0.0018 |
| TC5 | 0.1547 | 1.44 | 0.1547 | 0.2228 | 0.009 |
| TC6 | 0.0467 | 0.64 | 0.0467 | 0.0299 | 8.0E-4 |
| TC7 | 0.1115 | 1.44 | 0.1115 | 0.1601 | 0.0043 |
| TC8 | 0.0467 | 0.16 | 0.0467 | 0.0075 | 2.0E-4 |

TCN=[2,1,3] CFT={1,2,3,4,5,7,8,9} TFC=8 ET=11+5=16

| Iteration 3 |
| TC1 | 0.2867 | 7.84 | 0.2867 | 2.2477 | 0.0539(AV) |
| TC2 | 0.1 | 0.64 | 0.1 | 0.064 | 0.0015(AV) |
| TC3 | 0.4591 | 4.0 | 0.4591 | 1.8364 | 0.0441(AV) |
| TC4 | 0.028 | 1.44 | 0.028 | 0.0403 | 0.001 |
| TC5 | 0.0928 | 1.44 | 0.0928 | 0.1336 | 0.0032(*) |
| TC6 | 0.028 | 0.64 | 0.028 | 0.0179 | 4.0E-4 |
| TC7 | 0.0669 | 1.44 | 0.0669 | 0.0963 | 0.0023 |
| TC8 | 0.028 | 0.16 | 0.028 | 0.0045 | 1.0E-4 |

TCN=[2,1,3,5] CFT={1,2,3,4,5,6,7,8,9,10} TFC=10 ET=16+4=20 Ant 2 reached 10 faults type. So Ant 2 complete its process.

\[
\text{TSOR} = \frac{n - \text{optimum number of test cases in a test suite}}{n} \times 100 \tag{4}
\]
Here \( n \) denotes total size of input test cases in a test suite. With the help of equation (4), FTCBACO technique requires higher percentage of test suite optimization rate than existing GA technique and greedy approach. By using equation (4), size of input test cases in table 1 is 8 and an optimum test cases of existing GA technique and greedy approach is 4. Then the TSOR of GA is calculated as follows,

\[
TSOR = \frac{8 - 4}{8} \times 100 = 50\%
\]

And an optimum test cases of Proposed algorithm technique is 3 then the TSOR of FTCBACO is

\[
TSOR = \frac{8 - 3}{8} \times 100 = 63\%
\]

**Table 5. Outcomes of Ant 3**

| TCi | Pheromone\([i]\) | EV \[
\frac{\text{covered faults}[i]}{\text{total faults}} \times \text{execution time}[i]
\] | \[\text{pheromone}[i]\] \(\beta\) | \[\text{pheromone}[i]\] \(\beta\) \[\text{Probability}[i]\] |
|-----|------------------|------------------|------------------|------------------|
| Iteration 1 |
| Initially Set TCN={3} CFT={1,5,7,8} TFC = 4 ET = 5 |
| TC1 | 0.1032 | 7.84 | 0.1032 | 0.8091 | 0.0179(*) |
| TC2 | 0.036 | 6.64 | 0.036 | 0.023 | 5.0E-4 |
| TC3 | 0.5653 | 4.0 | 0.5653 | 2.2612 | 0.0501(AV) |
| TC4 | 0.0101 | 1.44 | 0.0101 | 0.0145 | 3.0E-4 |
| TC5 | 0.2134 | 1.44 | 0.2134 | 0.3073 | 0.0068 |
| TC6 | 0.0101 | 0.64 | 0.0101 | 0.0065 | 1.0E-4 |
| TC7 | 0.0241 | 1.44 | 0.0241 | 0.0347 | 8.0E-4 |
| TC8 | 0.0101 | 0.16 | 0.0101 | 0.0016 | 0.0 |

TC3 already visited marked as AV. So select next higher probability marked as (*). TCN={3,1} CFT={1,2,4,5,7,8,9} TFC=7 ET=5+7=12

| Iteration 2 |
| TC1 | 0.4619 | 7.84 | 0.4619 | 3.6213 | 0.0719(AV) |
| TC2 | 0.0216 | 0.64 | 0.0216 | 0.0138 | 3.0E-4 |
| TC3 | 0.3392 | 4.0 | 0.3392 | 1.3568 | 0.027(AV) |
| TC4 | 0.0061 | 1.44 | 0.0061 | 0.0088 | 2.0E-4 |
| TC5 | 0.128 | 1.44 | 0.128 | 0.1843 | 0.0037(*) |
| TC6 | 0.0061 | 0.64 | 0.0061 | 0.0039 | 1.0E-4 |
| TC7 | 0.0145 | 1.44 | 0.0145 | 0.0209 | 4.0E-4 |
| TC8 | 0.0061 | 0.16 | 0.0061 | 0.001 | 0.0 |

TCN={3,1,5} CFT={1,2,3,4,5,6,7,8,9,10} TFC=10 ET=12+4=16 Ant 3 reached 10 faults type. So Ant 3 complete its process.
The above results proved that Proposed FTCBACO algorithm technique contributed higher test suite reduction rate that existing GA.

| Ant 4 | Pheromone[i] | EV (coveredfaults[i] * executiontime[i]) | [pheromone[i]]^β | EV*[pheromone[i]]^β | Probability[i] |
|-------|--------------|------------------------------------------|-------------------|---------------------|----------------|
| TC1   | 0.1663       | 7.84                                     | 0.1663            | 1.3038              | 0.0246(•)     |
| TC2   | 0.0078       | 0.64                                     | 0.0078            | 0.005               | 1.0E-4        |
| TC3   | 0.1221       | 4.0                                      | 0.1221            | 0.4884              | 0.0092        |
| TC4   | 0.3022       | 1.44                                     | 0.3022            | 0.4352              | 0.0082(•)     |
| TC5   | 0.2261       | 1.44                                     | 0.2261            | 0.3256              | 0.0062        |
| TC6   | 0.0022       | 0.64                                     | 0.0022            | 0.0014              | 0.0           |
| TC7   | 0.0052       | 1.44                                     | 0.0052            | 0.0075              | 1.0E-4        |
| TC8   | 0.0022       | 0.16                                     | 0.0022            | 4.0E-4              | 0.0           |

TC4 already visited marked as AV. So select next higher probability marked as (•). TCN=[4,1] CFT=[2,4,7,9] TFC=4 ET=4+7=11

| Iteration 2 | TC1 | 0.4998 | 7.84 | 0.4998 | 3.9184 | 0.068(•) |
|-------------|-----|--------|------|--------|--------|----------|
| TC2         | 0.0047 | 0.64  | 0.0047 | 0.003 | 1.0E-4   |
| TC3         | 0.0733 | 4.0   | 0.0733 | 0.2932 | 0.0051(•) |
| TC4         | 0.1813 | 1.44  | 0.1813 | 0.2611 | 0.0045(•) |
| TC5         | 0.1357 | 1.44  | 0.1357 | 0.1954 | 0.0034   |
| TC6         | 0.0013 | 0.64  | 0.0013 | 8.0E-4 | 0.0       |
| TC7         | 0.0031 | 1.44  | 0.0031 | 0.0045 | 1.0E-4   |
| TC8         | 0.0013 | 0.16  | 0.0013 | 2.0E-4 | 0.0       |

TCN=[4,1,3] CFT=[1,2,4,5,7,8,9] TFC=7 ET=11+5=16

| Iteration 3 | TC1 | 0.2999 | 7.84 | 0.2999 | 2.3512 | 0.0379(•) |
|-------------|-----|--------|------|--------|--------|----------|
| TC2         | 0.0028 | 0.64  | 0.0028 | 0.0018 | 0.0     |
| TC3         | 0.444  | 4.0   | 0.444  | 1.776  | 0.0286(•) |
| TC4         | 0.1088 | 1.44  | 0.1088 | 0.1567 | 0.0286(•) |
| TC5         | 0.0814 | 1.44  | 0.0814 | 0.1172 | 0.0019(*) |
| TC6         | 8.0E-4 | 0.64  | 8.0E-4 | 5.0E-4 | 0.0     |
| TC7         | 0.0019 | 1.44  | 0.0019 | 0.0027 | 0.0     |
| TC8         | 8.0E-4 | 0.16  | 8.0E-4 | 1.0E-4 | 0.0     |

TCN=[4,1,3,5] CFT=[1,2,3,4,5,6,7,8,9,10] TFC=10 ET=16+4=20 Ant 4 reached 10 faults type. So Ant 4 complete its process.
Table 7. Outcomes of Ant 5

| TCi | Pheromone[i] | EV | [pheromone[i] ^ β EV * [pheromone[i] ^ β] | Probability[i] |
|-----|--------------|----|--------------------------------|----------------|
|     |              |    | [coveredfaults[i] / total faults] * executiontime[i] |                |
|     |              |    | [pheromone[i] ^ β EV * [pheromone[i] ^ β] | Probability[i] |
|     |              |    | [coveredfaults[i] / total faults] * executiontime[i] |                |

**Iteration 1**

Initially set TCN={5} CFT={3,6,10} TFC=3 ET = 4

| TC1 | 0.1079 | 7.84 | 0.1079 | 0.8459 | 0.0132(*) |
| TC2 | 0.001  | 0.64 | 0.001  | 6.0E-4 | 0.0       |
| TC3 | 0.1598 | 4.0  | 0.1598 | 0.6392 | 0.0099    |
| TC4 | 0.0392 | 1.44 | 0.0392 | 0.0564 | 9.0E-4    |
| TC5 | 0.5093 | 1.44 | 0.5093 | 0.7334 | 0.0114(AV) |
| TC6 | 3.0E-4 | 0.64 | 3.0E-4 | 2.0E-4 | 0.0       |
| TC7 | 7.0E-4 | 1.44 | 7.0E-4 | 0.001  | 0.0       |
| TC8 | 3.0E-4 | 0.16 | 3.0E-4 | 0.0    | 0.0       |

TC5 already visited marked as AV. So select next higher probability marked as (*). TCN={5,1} CFT={2,3,4,6,7,9,10} TFC=7 ET=4+7=11

| TC1 | 0.4647 | 7.84 | 0.4647 | 3.6432 | 0.053(AV) |
| TC2 | 6.0E-4 | 0.64 | 6.0E-4 | 4.0E-4 | 0.0       |
| TC3 | 0.0959 | 4.0  | 0.0959 | 0.3836 | 0.0056(*) |
| TC4 | 0.0235 | 1.44 | 0.0235 | 0.0338 | 5.0E-4    |
| TC5 | 0.3056 | 1.44 | 0.3056 | 0.4401 | 0.0064(AV) |
| TC6 | 2.0E-4 | 0.64 | 2.0E-4 | 1.0E-4 | 0.0       |
| TC7 | 4.0E-4 | 1.44 | 4.0E-4 | 6.0E-4 | 0.0       |
| TC8 | 2.0E-4 | 0.16 | 2.0E-4 | 0.0    | 0.0       |

TCN={5,1,3} CFT={1,2,3,4,5,6,7,8,9,10} TFC=10 ET=11+5=16 Ant 5 reached 10 faults type. So Ant 5 complete its process.

Table 12 illustrates the measurement of test suite reduction rate founded on several numbers of test cases ranges from 10 to 100 in test suite via three approaches specifically Existing Greedy approach, GA Technique and proposed FTCBACO algorithm using Java language. From the illustration of Table 11, we conclude that the test suite reduction rate of proposed FTCBACO algorithm is higher than Existing GA Technique.

Fig 4 shows the performance analysis of test suite reduction rate against various number of test suites using three methods specifically existing greedy approach, genetic algorithm technique and proposed FTCBACO Technique. As in Fig 4, the proposed FTCBACO Technique offers well test suite reduction rate for combinatorial testing as compared to an existing genetic algorithm. Additionally, while raising the number of test case in test suite, the test suite reduction rate is also raised. So, the test suite reduction rate using proposed FTCBACO Technique is higher.

5.2.2 Computational Time

Computational Time is represented as CT which is computed in terms of milliseconds. CT is formulated as follows,
Table 8. Outcomes of Ant 6

| TCi | Pheromone[i] | EV | [pheromone[i]] β | EV*[pheromone[i]] β | Probability[i] |
|-----|--------------|----|------------------|---------------------|----------------|
|     |              |    |                  |                     |                |
| Iteration 1 |
| Initially Set TCN={6} CFT={1,7} TFC=2 ET=4 |
| TC1 | 0.1673       | 7.84| 0.1673          | 1.3116             | 0.0183(*)      |
| TC2 | 2.0E-4       | 0.64| 2.0E-4          | 1.0E-4              | 0.0            |
| TC3 | 0.2745       | 4.0 | 0.2745          | 1.098               | 0.0154         |
| TC4 | 0.0085       | 1.44| 0.0085          | 0.0122              | 2.0E-4         |
| TC5 | 0.11         | 1.44| 0.11            | 0.1584              | 0.0022         |
| TC6 | 0.2001       | 0.64| 0.2001          | 0.1281              | 0.0018(AV)     |
| TC7 | 1.0E-4       | 1.44| 1.0E-4          | 1.0E-4              | 0.0            |
| TC8 | 1.0E-4       | 0.16| 1.0E-4          | 0.0                 | 0.0            |
| TCN={6,1} CFT={1,2,4,7,9} TFC=5 ET=4+7=11 |
| Iteration 2 |
| TC1 | 0.5004       | 7.84| 0.5004          | 3.9231              | 0.0515(AV)     |
| TC2 | 1.0E-4       | 0.64| 1.0E-4          | 1.0E-4              | 0.0            |
| TC3 | 0.1647       | 4.0 | 0.1647          | 0.6588              | 0.0086(*)      |
| TC4 | 0.0051       | 1.44| 0.0051          | 0.0073              | 1.0E-4         |
| TC5 | 0.066        | 1.44| 0.066           | 0.095               | 0.0012         |
| TC6 | 0.1201       | 0.64| 0.1201          | 0.0769              | 0.001(AV)      |
| TC7 | 1.0E-4       | 1.44| 1.0E-4          | 1.0E-4              | 0.0            |
| TC8 | 1.0E-4       | 0.16| 1.0E-4          | 0.0                 | 0.0            |
| TCN={6,1,3} CFT={1,2,4,5,7,8,9} TFC=7 ET=11+5=16 |
| Iteration 3 |
| TC1 | 0.3002       | 7.84| 0.3002          | 2.3536              | 0.0292(AV)     |
| TC2 | 1.0E-4       | 0.64| 1.0E-4          | 1.0E-4              | 0.0            |
| TC3 | 0.4988       | 4.0 | 0.4988          | 1.9952              | 0.0247(AV)     |
| TC4 | 0.0031       | 1.44| 0.0031          | 0.0045              | 1.0E-4         |
| TC5 | 0.0396       | 1.44| 0.0396          | 0.057               | 7.0E-4(*)      |
| TC6 | 0.0721       | 0.64| 0.0721          | 0.0461              | 6.0E-4(AV)     |
| TC7 | 1.0E-4       | 1.44| 1.0E-4          | 1.0E-4              | 0.0            |
| TC8 | 1.0E-4       | 0.16| 1.0E-4          | 0.0                 | 0.0            |
| TCN={6,1,3,5} CFT={1,2,3,4,5,6,7,8,9,10} TFC=10 ET=16+4=20 |

Ant 6 reached 10 faults type. So Ant 6 complete its process.
Here \( n \) denotes total size of input test cases and \( time \) denotes total time needed for optimizing test case. With the help of equation (5), proposed FTCBACO algorithm expects lower computational time than existing greedy approach and GA technique. By using equation (4), size of input test cases in table 1 is 8 and a time taken for finding the optimum test cases of existing GA technique is 0.9 ms. Then the CT of GA is calculated as follows,

\[
CT = n \times time(OT)
\]  

Here \( n \) denotes total size of input test cases and \( time \) denotes total time needed for optimizing test case. With the help of equation (5), proposed FTCBACO algorithm expects lower computational time than existing greedy approach and GA technique. By using equation (4), size of input test cases in table 1 is 8 and a time taken for finding the optimum test cases of existing GA technique is 0.9 ms. Then the CT of GA is calculated as follows,

\[
CT = 8 \times 0.9 \text{ms} = 7.2 \text{ms}
\]

And a time taken for finding the optimum test cases of FTCBACO technique is 0.6 ms. Then the CT of FTCBACO is
Table 10. Outcomes of Ant 8

| TCi | Pheromone[i] | EV | [pheromone[i]]β EV*[pheromone[i]]β | Probability[i] |
|-----|--------------|----|-----------------------------------|----------------|
|     |              |    |                                   |                |
| Iteration 1 |
| TC1 | 0.1004       | 7.84 | 0.1004                            | 0.7871         | 0.0084(*)       |
| TC2 | 1.0E-4       | 0.64 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC3 | 0.1673       | 4.0  | 0.1673                            | 0.6692         | 0.0071         |
| TC4 | 1.0E-4       | 1.44 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC5 | 0.2052       | 1.44 | 0.2052                            | 0.2955         | 0.0032         |
| TC6 | 0.0034       | 0.64 | 0.0034                            | 0.0022         | 0.0            |
| TC7 | 0.0389       | 1.44 | 0.0389                            | 0.056          | 0.03E-4        |
| TC8 | 0.2001       | 0.16 | 0.2001                            | 0.032          | 0.00E-4        |

TC8 already visited marked as AV. So select next higher probability marked as (*). TCN={8,1} CFT={2,4,7,9,10} TFC=5 ET=9+5=14

| Iteration 2 |
| TC1 | 0.4602       | 7.84 | 0.4602                            | 3.608          | 0.0369(AV)     |
| TC2 | 1.0E-4       | 0.64 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC3 | 0.1004       | 4.0  | 0.1004                            | 0.4016         | 0.0041(*)      |
| TC4 | 1.0E-4       | 1.44 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC5 | 0.1231       | 1.44 | 0.1231                            | 0.1733         | 0.0018         |
| TC6 | 0.002        | 0.64 | 0.002                             | 0.0013         | 0.0            |
| TC7 | 0.0233       | 1.44 | 0.0233                            | 0.0336         | 3.0E-4         |
| TC8 | 0.1201       | 0.16 | 0.1201                            | 0.0192         | 2.0E-4(AV)     |

TCN={8,1,3} CFT={1,2,4,5,7,8,9,10} TFC=8 ET=9+5=14

| Iteration 3 |
| TC1 | 0.2761       | 7.84 | 0.2761                            | 2.1646         | 0.0212(AV)     |
| TC2 | 1.0E-4       | 0.64 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC3 | 0.4602       | 4.0  | 0.4602                            | 1.8408         | 0.018(AV)      |
| TC4 | 1.0E-4       | 1.44 | 1.0E-4                            | 1.0E-4         | 0.0            |
| TC5 | 0.0739       | 1.44 | 0.0739                            | 0.1064         | 0.001(*)      |
| TC6 | 0.0012       | 0.64 | 0.0012                            | 8.0E-4         | 0.0            |
| TC7 | 0.014        | 1.44 | 0.014                             | 0.0202         | 2.0E-4         |
| TC8 | 0.0721       | 0.16 | 0.0721                            | 0.0115         | 1.0E-4(AV)     |

TCN={8,1,3,5} CFT={1,2,3,4,5,6,7,8,9,10} TFC=10 ET=14+4=18 Ant 8 reached 10 faults type. So Ant 8 complete its process.
The above results shown that Proposed FTCBACO algorithm technique demanded less computational time than existing GA. The performance analysis of computational time is achieved based on distinct numbers of test suite by means of existing genetic algorithm and proposed FTCBACO Technique is presented in Table 13. From the illustration of Table 12, we conclude that the computational time of proposed FTCBACO algorithm is lesser than existing greedy approach and GA technique.

Fig 5 proves the performance analysis of computational time compared to several number of test Cases with the help of three methods explicitly existing greedy approach, genetic algorithm technique and proposed FTCBACO Technique. As in Fig 5, the proposed FTCBACO Technique suggests a lesser amount of computational time used for combinatorial testing compared towards an existing genetic algorithm and greedy approach. Moreover, while enriching the quantity of test cases in test suite, the computational time is too demoted. So, the computational time using proposed FTCBACO Technique is lesser.

### 5.2.3 Testing Cost

TC is used for testing the excellence of software product lines. Testing Cost denoted as TC which is calculated in milliseconds (ms) for determining the amount of time essential to optimize test cases in a test suite. TC is mathematically formed as follows,

$$TC = \text{Endingtime} - \text{Startingtime}$$

Whereas starting time represents the starting time of testing the software test suite and an ending time represents time taken for finding optimum test cases in a test suite. While TC is lower, the technique is referred to as powerful technique. Using formula (6), Proposed FTCBACO algorithm demands a lesser volume of testing cost than existing GA technique and greedy approach. Size of input test cases in a test suite is 8 in table1 and also existing GA algorithm required 11.3ms called ending time to find optimum test cases of table 1 and starting time taken as 0ms. Then the testing cost of test suite with GA algorithm is calculated as follows,

$$TC = 11.3\text{ms} - 0\text{ms} = 11.3\text{ms}$$

---

**Table 11. Step by step execution of Greedy approach using sample input fault matrix described in Table 1**

| Step no. | Greedy Approach | Minimized Test Suite | Faults Type Covered |
|----------|-----------------|----------------------|----------------------|
| 1        |                 | MTS = {tc1}          | FTC = {ft2, ft4, ft7, ft9} |
| 2        |                 | MTS = {tc1, tc3}     | FTC = {ft1, ft2, ft4, ft5, ft7, ft8, ft9} |
| 3        |                 | MTS = {tc1, tc3, tc4} | FTC = {ft1, ft2, ft4, ft5, ft7, ft8, ft9} |
| 4        |                 | MTS = {tc1, tc3, tc4, tc5} | FTC = {ft1, ft2, ft3, ft4, ft5, ft6, ft7, ft8, ft9, ft10} All faults type covered | Needed Execution time is 0.038 (ms) |

$$CT = 8 \times 0.6\text{ms} = 4.8\text{ms}$$

The above results shown that Proposed FTCBACO algorithm technique demanded less computational time than existing GA.

The performance analysis of computational time is achieved based on distinct numbers of test suite by means of existing genetic algorithm and proposed FTCBACO Technique is presented in Table 13. From the illustration of Table 12, we conclude that the computational time of proposed FTCBACO algorithm is lesser than existing greedy approach and GA technique.

Fig 5 proves the performance analysis of computational time compared to several number of test Cases with the help of three methods explicitly existing greedy approach, genetic algorithm technique and proposed FTCBACO Technique. As in Fig 5, the proposed FTCBACO Technique suggests a lesser amount of computational time used for combinatorial testing compared towards an existing genetic algorithm and greedy approach. Moreover, while enriching the quantity of test cases in test suite, the computational time is too demoted. So, the computational time using proposed FTCBACO Technique is lesser.
And similarly proposed FTCBACO algorithm required ending time as 7.4ms to find optimum test cases of table 1 and starting time taken as 0ms. Then the testing cost of test suite with Proposed FTCBACO algorithm is calculated as follows,
The above outcomes attested that proposed technique demanded a smaller amount of testing cost than an existing GA technique.

Table 13 illustrates the measurement of testing cost founded on several numbers of test cases ranges from 10 to 100 in test suite via three methodologies precisely existing greedy approach and GA Technique, proposed FTCBACO algorithm using Java language. From the illustration of Table13, we determine that the testing cost of proposed FTCBACO algorithm is lesser than Existing GA Technique.

\[ TC = 7.4 \text{ms} - 0 \text{ms} = 7.4 \text{ms} \]

The above outcomes attested that proposed technique demanded a smaller amount of testing cost than an existing GA technique.
Fig 6 shows the performance analysis of testing cost compared to many number of test suites using three techniques particularly existing greedy approach, genetic algorithm technique and proposed FTCBACO Technique. As in Fig 6, the proposed FTCBACO Technique poses lesser testing cost for combinatorial testing than existing genetic algorithm. In addition, while raising the number of test case in test suite, the testing cost is also lowered. So, the testing cost using proposed FTCBACO Technique is lesser.
6. CONCLUSION

Ant Colony Optimization algorithm is an extraordinary technique to find the best test cases in a test suite. In this investigation work, proposed FTCBACO algorithm is designed, implemented using JAVA language in an efficient manner and as well the results of proposed algorithm is compared with an existing Greedy Approach and Genetic Algorithm technique. In existing techniques, additional computational time and testing cost needed to optimize the test suite and also it produced least test suite reduction rate due to the special operators of genetic algorithm like selection, crossover and mutation.
and also mutating bits in mutation operation. But, Proposed FTCBACO Algorithm optimizes the test suite highly. Our investigation work exhibits the evaluation results of proposed algorithm which proves maximum test suite reduction rate, minor computational time and lower testing cost. From the comparative result analysis of existing and proposed techniques, our proposed FTCBACO technique optimizes the test suite in an efficient manner in terms of test suite reduction rate, computational time and testing cost. We hope to continue our research in future with complicated programs, on superior size of test suites and greater fault intensities.

### Table 14. Layout of Testing Cost

| Number Of Test Cases | Existing Greedy Approach | Existing Genetic Algorithm | Proposed FTCBACO Algorithm |
|----------------------|--------------------------|----------------------------|----------------------------|
| 10                   | 15.2                     | 13.9                       | 9.2                        |
| 20                   | 20.2                     | 17.5                       | 12.6                       |
| 30                   | 25.3                     | 21.3                       | 16.9                       |
| 40                   | 30.9                     | 24.8                       | 19.1                       |
| 50                   | 33.8                     | 29.3                       | 23.4                       |
| 60                   | 37.2                     | 32.7                       | 26.3                       |
| 70                   | 42.3                     | 37.6                       | 28.7                       |
| 80                   | 47.4                     | 41.2                       | 31.5                       |
| 90                   | 51.3                     | 45.8                       | 35.8                       |
| 100                  | 58.7                     | 48.3                       | 39.2                       |

### Figure 6. Efficiency of Testing Cost against Number of Test Cases

![Number of Test Cases Vs Testing Cost](image-url)
REFERENCES

Anwar, Ahsan, & Catal. (2015). Neuro-Fuzzy Modeling for Multi-Objective Test Suite Optimization. *Journal of Intelligent Systems*, 25(2).

Bharath & Sangeetha. (2019). Test Case Optimization for Enhancing System Software Quality using Genetic Algorithm. *International Journal of Innovative Technology and Exploring Engineering*, 8(12).

Coutinho, A. E. V. B., Cartaxo, E. G., & Machado, P. D. L. (2016). Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. *Software Quality Journal, Springer*, 24(2), 407–445. doi:10.1007/s11219-014-9265-z

Gong, D. (2013). A test-suite reduction approach to improving fault-localization effectiveness. *Computer Languages, Systems & Structures, Elsevier*, 39(3), 95–108. doi:10.1016/j.cl.2013.04.001

Haider, Ali, Nadeem, Aamer, Rafiq, & Shahzad. (2014). Multiple objective test suite optimization: A fuzzy logic based approach. *Journal of Intelligent & Fuzzy Systems*, 27(2), 863-875.

Harris, P., & Raju, N. (2015). A greedy approach for coverage-based Test Suite reduction. *The International Arab Journal of Information Technology*, 12(1), 17–23.

Harrold, M. J., Gupta, R., & Soffa, M. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3), 270–285. doi:10.1145/152388.152391

Jeffrey, & Gupta. (2005). Test suite reduction with selective redundancy. *21st IEEE International Conference on Software Maintenance (ICSM’05)*, 549-558. doi:10.1109/ICSM.2005.88

Jeyaprakash, S., & Alagarsamy, K. (2015). A Distinctive Genetic Approach for Test-Suite Optimization. *Procedia Computer Science, Elsevier*, 62, 427–434. doi:10.1016/j.procs.2015.08.437

Khan, Gupta, & Bora. (2015). An Efficient Technique to Test Suite Minimization using Hierarchical Clustering Approach. *International Journal of Emerging Science and Engineering*, 3(11), 1–10.

Khan, S., Nadeem, A., & Awais, A. (2006). TestFilter: A Statement-Coverage Based Test Case Reduction Technique. *IEEE International Multitopic Conference*, 275-280. doi:10.1109/INMIC.2006.358177

Lam, Raju, Kiran, & Srivastav. (2012). Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony. Procedia Engineering, 30, 191-200.

Li, N. (2016). An Effective Strategy to Build Up a Balanced Test Suite for Spectrum-Based Fault Localization. *Mathematical Problems in Engineering*.

Li, Zheng, Harman, & Hieron. (2007). Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, 33(4), 225-237.

Rothermel, G., & Harrold, M. J. (1997). A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2), 173–210. doi:10.1145/248233.248262

Rout, Mishra, & Malu. (2013). An Effective Test Suite Reduction Using Priority Cost Technique. *International Journal of Computer Science and Engineering Technology*, 4(4), 372–36.

Sen & Mall. (2012). State-Model-Based Regression Test Reduction for Component-Based Software. *International Scholarly Research Network, ISRN Software Engineering*, 2012, 1–9.

Singh & Shree. (2016). An Analysis of Test Suite Minimization Techniques. *International Journal of Engineering Sciences & Research Technology*, 5, 252–260.

Yoo, S., & Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite minimization. *Journal of Systems and Software*, 83(4), 689–701. doi:10.1016/j.jss.2009.11.706