Abstract

Model-based testing is a crucial but challenging stage of the software development process. The process of model-based testing needs to be optimized, which is a difficult task. In this article, we present an approach for selecting minimum test suites that is based on the meta-heuristic firefly algorithm. We modify the firefly algorithm and define the suitable multi-objective function to optimize the test suites. The suggested approach uses firefly behavior to address the current issue. The modified approach chooses the best test suites that quickly find the maximum coverage in less time.

Keywords: Firefly algorithm, Light intensity, Model-based testing, Multi-objective fitness function, Test suites optimization.

Introduction

Model-based testing is a necessary step in the proper maintenance of software (Suryanarayan, S., et al., 2021). Model-based testing is a procedure of retesting a modified system or piece of software using the old test suite to ensure that any bug fixes and new features do not negatively impact any functionality passed down from a prior version. This means that, while doing model-based testing on a system that contains the test suite and the changed program, the program must be validated in relation to test suite (Dang, X., et al., 2020). This procedure is carried out as a result of modifications to the business logic, the addition of fresh requirements, or adjustments to the current system. Rerunning the test suite consumes a large portion of time and resources (Hashim, F., et al., 2022). The main difficulties arise from the need to reduce time and expense. Some methods to address this issue include test suite selection, optimization, and prioritization (Huang, Y., et al., 2021).

Test suite optimization refers to the process of choosing test suites that should quickly and cheaply reveal errors that have gone undiscovered. Additionally, it assists the system’s productivity and quality control to grow. It simultaneously cut testing expenses and sped up the cycle time (Khatibsarybini, M., et al., 2019). The ant colony optimization (ACO), genetic algorithm (GA), particle swarm optimization (PSO), cuckoo search (CS), artificial bee colony algorithm (ABCA), and others are different optimization strategies (Mahali, P., et al., 2018; Mehboob, F., et al., 2020).

In order to speed up and lower the expense of model-based testing, this paper proposes a model-based test suite optimization approach. Here, a modified-multi objective firefly optimization algorithm (m-MOFOA) is designed to obtain an optimal test suite without the test suites being redundant and to further give maximum coverage in the shortest amount of time. The article’s remaining sections are organized as follows: Section 2 explains the material and methods in more detail using four case studies and a program namely hospital management system (HMS), library management system (LMS), social account system (SAS), hotel management system (HoMS) and triangle classification problem (TCP) and section 3 discusses the results and discussion and also compares this approach with other meta heuristic algorithms that is GA, PSO and a traditional firefly optimization algorithm (FOA), and section 4 wraps up the study with a conclusion.
Materials and Methods
This article proposes a novel m-MOFOA technique for test suite optimization. With the fitness function constructed using a number of objectives, such as cyclomatic complexity, activity coverage, and total cost; we used the firefly algorithm to optimize the test suites. The structural design of the proposed work is shown in Figure 1.

To maximize many objectives simultaneously, multi-objective optimization is used. Because traditional optimization algorithms cannot simultaneously satisfy many objectives, they cannot produce the best optimal solution for test suite selection. By choosing the best test suites using the meta-heuristic firefly optimization technique, a novel solution to this problem is developed. The flowchart for modified-multi objective firefly optimization (m-MOFOA) is shown in Figure 2.

The following mathematical description of the multi-objective optimization problem serves as the foundation for the proposed (m-MOFOA) technique:

\[ f(x) = [f(x_1), f(x_2), f(x_3), \ldots, f(x_m)] \]  
\[ \text{eq. 1} \]

From Eq.1 \( f(x) \) represents a set of multiple objectives \( f(x_1), f(x_2), f(x_3), \ldots, f(x_m) \) where \( x \) identifies a unique solution to the problem that was resolved.

Following is a mathematical expression of the multiple objectives:

\[ f(x) = [f(x_1) + f(x_2) + f(x_3)] \]  
\[ \text{eq. 2} \]

From eq. 2, \( f(x) \) indicates the set of multiple objectives, \( f(x_1) \) indicates a cyclomatic complexity, \( f(x_2) \) indicates total cost and \( f(x_3) \) indicates activity coverage. By using these constraints, fitness is computed for each firefly (for each test suite).

Software metrics like cyclomatic complexity (CC) (Panthi, V., et al., 2017; Reda, N., et al., 2022; Swathi, B., et al., 2020;) are used to quantify a program’s or an application’s complexity. The number of independent paths produced by the McCabe (Panthi, V., et al., 2017; Reda, N., et al., 2022; Swathi, B., et al., 2020;) Basic path testing approach is equal to the graph’s cyclomatic complexity. Thomas J. McCabe, Sr. created it in 1976 (Panthi, V., et al., 2017; Reda, N., et al., 2022). It can be calculated as:

\[ f(x_1) = \{CC = E - N + 2\} \]  
\[ \text{eq. 3} \]

From eq 3, number of edges shows the flow of nodes in current test suite and the is the number of activity nodes in the current test suite.

The cost of independent test suite is computed by adding all weights associated with each node.

\[ f(x_2) = \{TC_{TS} = \sum(\text{Cost of all nodes present in the Test Suite})\} \]  
\[ \text{eq. 4} \]

\[ TC_N = (\text{Out Deg}_N + \text{In Deg}_N + b_{rn} + DN) \]  
\[ \text{eq. 5} \]

Where \( TC_{TS} \) in eq 4 is the total cost of a test suite which can be computed by eq. 5. In eq 5 total numbers of outgoing edges is \( \text{Out Deg}_N \) and total number of incoming edges is \( \text{In Deg}_N \). branch weight is \( b_{rn} \) and \( DN \) is the Decision node and total cost of a node is \( TC_N \).

When every activity in a UML simplified activity graph is visited at least once, activity coverage of the simplified activity graph can be reached.

\[ A_{GOV} = \sum A_i + A_2 + \ldots + A_n \]  
\[ \text{eq. 6} \]

Activity coverage cost \( A_{GOV} \) can be computed by using eq. 6. Where \( A_{GOV} \) is the total number of activities covered by each test suite. The sum of activities covered by a test suite (TS) is represented by \( A_1 + A_2 + \ldots + A_n \) and \( TA_n \) is a representation of the total number of activities present in an application.

The objective function in the proposed approach uses the previously discussed objectives to determine each test suite’s firefly brightness, the fitness function-equivalent brightness function has been framed to measure the optimality of developed test suites or solutions as follows.

\[ \text{Brightness Function (BF)} = \sum(f(x_1) + f(x_2) + f(x_3)) \]  
\[ \text{eq. 7} \]

The development of the modified-multi objective firefly optimization approach was influenced by the fireflies’ behavior of flashing lights. The fireflies are taken as the test suites. Fireflies are all unisexual. Fireflies are attracted to other fireflies based on the brightness of their light (i.e., \( \text{Brightness Function (BF)} = \text{Fitness Function (FF)} \)). Brightness is correlated with the firefly’s attractiveness. The one that is brighter attracts the firefly with the lowest intensity. The firefly then moves on to another brighter location with the highest fitness value. This increases or improves firefly’s intensity. Also, the firefly with the lowest intensity has a decreased fitness for the best solutions available right now. As a result, the brighter one is chosen as the best solution.

Take \( n_1 \) (test suites) i.e. fireflies \( f_1, f_2, f_3, \ldots, f_n \). The brightness function BF was then used to determine each firefly’s intensity, as shown below.

![Figure 1: Structural design of proposed technique](image-url)
According to the eq. 8 above, \( I(f_j) \) denotes the firefly's light intensity. Here, maximum light-intensity firefly attracts the other one, \( I(f_j) > I(f_i) \) where \( j = 1, 2, 3, ..., n \) but \( i \neq j \). The attractiveness of the firefly varies depending on the degree of light it absorbs. The less bright firefly \( f_i \) moves toward the other, brighter firefly \( f_j \) depending on the intensity computation (eq. 8), the higher-brighter firefly is determined to be the best solution. In addition, less bright firefly is removed and the position of the firefly is updated for effectively detecting the best firefly for next iteration which is expressed as follows,

\[
TS_i = TS_i + 1 \quad \text{eq.} \, 9
\]

From the above eq. 9, \( TS_i + 1 \) indicates an updated position of the firefly, \( TS_i \) indicates a current position.

This in turns, the test suite optimization is achieved by proposed m-MOFOA approach. Through the chosen optimal test suite, the object-oriented application is tested for enhancing the quality of software.

**Flow chart of the proposed m-MOFOA technique**

Flow chart for the proposed technique is as follows:

**Flowchart 1:** Algorithm for the proposed technique (m-MOFOA) for test suite optimization

```
Algorithm: modified-Multi Objective Firefly Optimization Algorithm (m-MOFOA) for Test Suite Optimization
Input: Software under test, Set of test suites in
\( TS = [TS_1, TS_2, TS_3, ..., TS_n] \)
Output: Optimized Test Suites

START
1. Initialize the firefly (that is test suites) with random position
2. For every \( TS_i \) in \( TS \) do
3. Compute fitness function using eq. 7
4. if \( FF \geq TS \) then
5. Select test suite as optimal
6. Go to step 19
7. else
8. Compute fitness value of every test suite
9. end if
10. For every firefly in \( TS_i \) do
11. For each firefly \( f_j \) belong to \( TS_1, TS_2, TS_3, ..., TS_n \) do
12. For each firefly \( f_j \) belong to \( TS_1, TS_2, TS_3, ..., TS_n \) do
13. Using the fitness function \( FF \) calculate light intensity \( I(f_j) \) using eq. 8
14. if \( I(f_j) > I(f_i) \) then
15. Firefly \( f_j \) travels towards \( f_i \)
16. end if
17. Optimize the \( f_j \) based on their light intensity
18. Select unique \( f_j \) based on their \( FF \)
19. end for
20. Calculate brightness of each \( TS \) using eq. 8
21. Update \( TS_i + 1 \) using eq. 9
22. Return optimized/minimized test suites
23. While the maximum iteration is reached go to step 2
24. end for
EXIT
```

To test object-oriented applications and improve software quality, the above (m-MODOA) is used to find the best-optimized test suites. In accordance with brightness, all fireflies are then optimized. Up until the maximum iteration, this process is repeated.

**Experimental evaluation of the proposed m-MOFOA technique**

For experimental evaluation of the proposed m-MOFOA technique, we used real-world case studies and a program such as HMS, LMS, SAS, HoMS and TCP. Case studies and a program has 18, 12, 13, 14 and 12 generated test suites and optimized test suites are 7, 6, 5, 5, and 4 (Figure 3). Here we have shown in Table 1 case study HMS’s optimized test suites with their fitness value.

**HMS**

Table 1 shows the fitness value of each test suite and firefly on each test suite and last column i.e., the optimized test suite column, shows the accepted and non-accepted test suites with their fitness value. According to this fitness value test suites with low-intensity fitness value and redundant value of fitness function are not accepted and hence removed. This case study takes total 18 iterations for the optimization of test suites. Therefore, Table 1 shows the fitness value of each test suite and according to their fitness value optimized test suites are 7 i.e. TS2, TS6, TS9, TS13, TS14, TS15, and TS18 are selected or optimized for model based testing of object oriented applications and TS1, TS3, TS4, TS5, TS7, TS8, TS10, TS11, TS12, TS16 and TS17 are rejected. These optimized test suites can be represented by the Figure 3.

**Results**

The two independent parameters used in this experiment were the four case studies, HMS, LMS, SAS, and HoMS, one program, TCP, and the test suites produced by these case studies and program. Four additional dependent
parameters, namely computational time, all-activity coverage percentage, all-edge coverage percentage, and percentage of test suite rejection, were also assessed during the experiment. The suggested m-MOFOA approach was successfully compared to various meta-heuristic approaches (GA, PSO, and FOA) on four case studies and a program.

**Computation Time**

Computation time (Ct) is the time required to optimize test suites for model-based testing. It is expressed in milliseconds (ms), which is calculated by multiplying the total number of test suites \( TSn \) by the time \( Time_{TS} \) required to select a single test suite for the optimization process. Table 2 can be used to display the results.

\[
\text{W}Ct = TSn \times Time_{TS}
\]

The performance analysis of computational time for four techniques with regard to various test suite counts is shown in Table 3 below. The suggested m-MOFOA technique effectively cuts the computational time compared to the other current techniques. The suggested m-MOFOA technique uses the computational time as 36, 18, and 9ms for 18 test suites, whereas the existing GA, PSO, and FOA computational times are roughly 70, 55, and 45 ms for the 18 test suites. According to the description above, the m-MOFOA technique reduces the amount of time needed to choose test suites for MBT testing. The graph is produced between the values in the following Table 3 and depicted in Figure 4.

The performance analysis of computational time for the proposed m-MOFOA technique, current GA, PSO, and FOA are shown in Figure 4. In the aforementioned figure, various test suites are used in a total of 18, 12, 13, 14, and 10 runs. When compared to state-of-the-art works, the experimental findings of computational time employing the suggested m-MOFOA technique are significantly shorter. As a result, compared to current GA, PSO, and FOA techniques, the time needed to select the best test suites is decreased by 9 to 11 ms.

**All-activity coverage percentage**

The term “All Activity Coverage Percentage” (AACP) refers to the percentage of activities that are covered out of all the activities in an application. It can be calculated by dividing the total no. of activities covered by optimized test suites \( Act\_Cov\_Opt\) by total number of activities \( Total\_Act\) present in the application (in %). Table 4 can be used to display the results.

\[
AACP = \frac{Act\_Cov\_Opt(TS)}{Total\_Act} \times 100\%
\]

The comparison of AACPs for the three methods GA, PSO and FOA and proposed m-MOFOA technique, according to various test suite numbers that is 18, 12, 13, 14, and 10 is shown in Table 5 below. When compared to three other techniques, the m-MOFOA technique specifically improves the percentage of all activities covered. When 18 test suites were taken into account, the AACP for the existing GA, PSO, and FOA was 70, 75, and 78%, respectively, whereas the suggested m-MOFOA technique acquired 93% of the total. The values in Table 5 are taken into consideration when creating the graph as Figure 5.
### Table 3: Comparison between proposed m-MOFOA technique and other meta-heuristic techniques for Ct values

<table>
<thead>
<tr>
<th>Case studies</th>
<th>GA</th>
<th>PSO</th>
<th>FOA</th>
<th>Proposed m-MOFOA</th>
<th>Ct with GA</th>
<th>Ct with PSO</th>
<th>Ct with FOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>72</td>
<td>54</td>
<td>45</td>
<td>36</td>
<td>36</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>LMS</td>
<td>48</td>
<td>36</td>
<td>30</td>
<td>24</td>
<td>24</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>SAS</td>
<td>52</td>
<td>39</td>
<td>33</td>
<td>26</td>
<td>26</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>HoMS</td>
<td>56</td>
<td>42</td>
<td>35</td>
<td>28</td>
<td>28</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>TCP</td>
<td>40</td>
<td>30</td>
<td>25</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 4: Results obtained for AACP values using all case studies and a program

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>27</td>
<td>29</td>
<td>(27/29) × 100% = 93%</td>
</tr>
<tr>
<td>LMS</td>
<td>23</td>
<td>26</td>
<td>(23/26) × 100% = 89%</td>
</tr>
<tr>
<td>SAS</td>
<td>25</td>
<td>27</td>
<td>(25/27) × 100% = 93%</td>
</tr>
<tr>
<td>HoMS</td>
<td>27</td>
<td>30</td>
<td>(27/30) × 100% = 90%</td>
</tr>
<tr>
<td>TCP</td>
<td>11</td>
<td>12</td>
<td>(11/12) × 100% = 92%</td>
</tr>
</tbody>
</table>

### Table 5: Comparison between proposed m-MOFOA and other meta-heuristic techniques for AACP values

<table>
<thead>
<tr>
<th>Case Studies</th>
<th>GA (%)</th>
<th>PSO (%)</th>
<th>FOA (%)</th>
<th>Proposed m-MOFOA (%)</th>
<th>AACP with GA (%)</th>
<th>AACP with PSO (%)</th>
<th>AACP with FOA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>70</td>
<td>75</td>
<td>78</td>
<td>93</td>
<td>23</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>LMS</td>
<td>62</td>
<td>66</td>
<td>70</td>
<td>89</td>
<td>27</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>SAS</td>
<td>72</td>
<td>77</td>
<td>79</td>
<td>93</td>
<td>21</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>HoMS</td>
<td>68</td>
<td>70</td>
<td>75</td>
<td>90</td>
<td>22</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>TCP</td>
<td>69</td>
<td>73</td>
<td>77</td>
<td>92</td>
<td>23</td>
<td>19</td>
<td>1</td>
</tr>
</tbody>
</table>

The performance of the overall AACP is shown in Figure 5. The proposed m-MOFOA technique selects fewer ideal test suites for activity coverage than previous efforts. This contributes to a reduction in complexity and an improvement in application quality. As a result, when compared to GA, PSO, and FOA, the suggested m-MOFOA technique's AACP is increased to about 23, 20, and 15%, respectively.

### All-edge coverage percentage

The term “All edge coverage percentage” (AECP) refers to the percentage of edges that are covered out of all the edges in an application. It is calculated by dividing visited edges in optimized test suites $Opt_{TS}(V_{edges})$ by total number of edges $T_{edges}$ present in the application (in %). Table 6 can be used to display the results.

$$AECP = \frac{Opt_{TS}(V_{edges})}{T_{edges}} \times 100\%$$

The comparison of AECPs for the three methods GA, PSO and FOA and proposed m-MOFOA technique, according to various test suite numbers that is 18, 12, 13, 14, and 10 is shown in Table 7 below. When 18 test suites were taken into account, the AECP for the existing GA, PSO, and FOA was 71, 75, and 79%, respectively, whereas the suggested m-MOFOA technique acquired 89% of the total. The values in Table 7 are taken into consideration when creating the graph as a Figure 6.

The performance of the overall AECP is shown in Figure 6. The proposed m-MOFOA technique selects fewer ideal test suites for edge coverage than previous efforts. This contributes to a reduction in complexity and an improvement in application quality. As a result, when...
The results analyzed for the PTSR based on different test suite counts utilizing four techniques, suggested m-MOFOA, the existing GA, PSO, and FOA, are shown in Table 9 below. It is evident from the result analysis that, when compared to other techniques already in use, the suggested m-MOFOA technique enhances the percentage of test suite rejection rate. The suggested m-MOFOA technique’s test suite rejection rate was found to be 62, 50, 63, 65, and 50% compared to existing GA’s rejection rate of 44, 32, 46, 50, and 30%, existing PSO’s rejection rate of 45, 35, 48, 52, and 33%, and FOA’s rejection rate of 50, 42, 52, 55, and 40%. The values in Table 9 are taken into consideration when creating the graph as Figure 7.

### Table 7: Comparison between proposed m-MOFOA and other meta-heuristic techniques for AECP values

<table>
<thead>
<tr>
<th>Case Studies</th>
<th>GA (%)</th>
<th>PSO (%)</th>
<th>FOA (%)</th>
<th>Proposed m-MOFOA (%)</th>
<th>AECP with GA (%)</th>
<th>AECP with PSO (%)</th>
<th>AECP with FOA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>71</td>
<td>75</td>
<td>79</td>
<td>89</td>
<td>18</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>LMS</td>
<td>73</td>
<td>76</td>
<td>82</td>
<td>95</td>
<td>22</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>SAS</td>
<td>72</td>
<td>74</td>
<td>78</td>
<td>94</td>
<td>22</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>HoMS</td>
<td>68</td>
<td>70</td>
<td>76</td>
<td>90</td>
<td>22</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>TCP</td>
<td>62</td>
<td>69</td>
<td>73</td>
<td>87</td>
<td>25</td>
<td>18</td>
<td>14</td>
</tr>
</tbody>
</table>

### Table 8: Results obtained for PTSR values using all case studies and a program

<table>
<thead>
<tr>
<th>Case Studies and a Program</th>
<th>Opt(TS), TS, (TS – Opt(TS))</th>
<th>PTSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>7, 18, (18-7) =11, (11/18) × 100% = 62%</td>
<td></td>
</tr>
<tr>
<td>LMS</td>
<td>6, 12, (12-6) = 6, (6/12) × 100% = 50%</td>
<td></td>
</tr>
<tr>
<td>SAS</td>
<td>5, 13, (13-5) = 8, (8/13) × 100% = 63%</td>
<td></td>
</tr>
<tr>
<td>HoMS</td>
<td>5, 14, (14-5) = 9, (9/14) × 100% = 65%</td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>5, 10, (10-5) = 5, (5/10) × 100% = 50%</td>
<td></td>
</tr>
</tbody>
</table>

compared to GA, PSO, and FOA, the suggested m-MOFOA technique’s AECP is increased to about 24, 20, and 15%, respectively.

**Percentage of test suite rejection**

The term “percentage of test suite rejection” (PTSR) refers to the percentage of test suites that have been rejected from the total number of generated test suites. To get this, subtract the total number of optimized test suites TS' from the total number of test suites TS, and then divide the result by the overall percentage of test suites. Table 8 can be used to display the results.

### Table 9: Comparison between proposed m-MOFOA and other meta-heuristic techniques for PTSR values

<table>
<thead>
<tr>
<th>Case studies</th>
<th>GA (%)</th>
<th>PSO (%)</th>
<th>FOA (%)</th>
<th>Proposed m-MOFOA (%)</th>
<th>PTSR with GA (%)</th>
<th>PTSR with PSO (%)</th>
<th>PTSR with FOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>44</td>
<td>45</td>
<td>50</td>
<td>62</td>
<td>18</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>LMS</td>
<td>32</td>
<td>35</td>
<td>42</td>
<td>50</td>
<td>18</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>SAS</td>
<td>46</td>
<td>48</td>
<td>52</td>
<td>63</td>
<td>17</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>HoMS</td>
<td>50</td>
<td>52</td>
<td>55</td>
<td>65</td>
<td>15</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>TCP</td>
<td>30</td>
<td>33</td>
<td>40</td>
<td>50</td>
<td>20</td>
<td>17</td>
<td>10</td>
</tr>
</tbody>
</table>

The results analyzed for the PTSR based on different test suite counts utilizing four techniques, suggested m-MOFOA, the existing GA, PSO, and FOA, are shown in Table 9 below. It is evident from the result analysis that, when compared to other techniques already in use, the suggested m-MOFOA technique enhances the percentage of test suite rejection rate. The suggested m-MOFOA technique’s test suite rejection rate was found to be 62, 50, 63, 65, and 50% compared to existing GA’s rejection rate of 44, 32, 46, 50, and 30%, existing PSO’s rejection rate of 45, 35, 48, 52, and 33%, and FOA’s rejection rate of 50, 42, 52, 55, and 40%. The values in Table 9 are taken into consideration when creating the graph as Figure 7.

Based on the number of test suites, 18, 12, 13, 14, and 10, Figure 7 displays the performance results of the PTSR. As a result, the m-MOFOA technique increases the PTSR by about 17, 18, and 10% compared to traditional GA, PSO, and FOA.
Discussion
In comparison to state-of-the-art approaches, the results reveal that the m-MOFOA approach boosts the test suite rejection percentage by 12%, the activity and edge coverage percentage by 19 and 16%, and the computation time by 11%, respectively.

Conclusion
The suggested m-MOFOA approach for test suite optimization helps achieve “multi-objective optimization” by reducing the redundancy ratio, reducing test suite size and time, and improving activity and edge coverage percentage. The goal of future study might be to apply the current research to a variety of industrial case studies in order to evaluate its effectiveness and usefulness based on other criteria.

Acknowledgments
We appreciate the systemic research department staff’s assistance in various ways up until this study was finished satisfactorily. The Computer Application and Information Technology Department at Shri Guru Ram Rai University in Dehradun provided support for this study.

References


