International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V70I3P203

Volume 70 Issue 3, 22-28, March, 2022


*Original Article*

# Test Cases Prioritization Using Ant Colony Optimization and Firefly Algorithm

Muhammad Afiq Ariffin[1], Rosziati Ibrahim[2], Izrulfizal Saufihamizal Ibrahim[3], Jahari Abdul Wahab[4]

[1,3]*PhD Candidate, Department of Software Engineering, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia*
[2]*Supervisor, Department of Software Engineering, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia*
[4] *Industry Collaborator, Engineering R&D Department, Sena Traffic Systems Sdn. Bhd., Kuala Lumpur, Malaysia*

[1]rosziati@uthm.edu.my

***Abstract*** *- A software testing process is the most complex and important part to be considered in the software development life cycle. This testing process usually takes a lot of time and is also very costly. The modification that has been made must also not affect the other unmodified parts of the software. Regression testing is the most suitable function that can be used for the software testing process, and the method includes the prioritization of test cases. There are several techniques in the prioritization of test cases, and most of the techniques are inspired by nature, such as Ant Colony Optimization (ACO) and Firefly Algorithm (FA). This paper will look at ACO and FA techniques for the prioritization of test cases. These techniques will be executed to identify the performance of each technique, which will be evaluated based on the Average Percentage of Faults Detected (APFD), execution time, and fault coverage. Based on the evaluation results, it showed that the FA technique recorded the lowest execution time and achieved a 100% of fault coverage.*

**Keywords** — *Software testing, Prioritization of test cases, ant colony optimization, Firefly algorithm.*

## I. INTRODUCTION

Software is usually affected by many factors during its development, resulting in the production of a product of lower quality or malfunction in those systems. The Discovery of errors before the product is released is one of the methods used during the development of software to produce a high-quality program by testing the quality of this software. Nowadays, the software development process is something that happens every day, and, in fact, it is becoming more vibrant and has grown successfully. Almost every day, there are new applications and systems in the environment, but some other systems or applications are also becoming less useful due to the development of the new systems or applications with new features that understand the needs of users and are easier to use. However, these new systems require periodic updates to meet the needs of users and make them easier for users to use. Every update added to an existing system needs to be tested first to see whether it is effective or not.

Software testing is important to test the system or software whether it meets its user requirements as well as to test the functionalities of the system is correct or not. Some of the researchers have conducted many techniques in the software testing domain for generating test cases and test cases prioritization, for example, [1–6]. This paper discusses the techniques used in software testing for prioritization of test cases and conducts an experiment for the techniques to find out which technique is better in terms of its performance in prioritization of the test cases.

The next section will discuss the related works, followed by a discussion on the test cases prioritization techniques. Then the results and discussion are presented in Section IV, followed by the conclusion and future works in Section V.

## II. RELATED WORK

The test case prioritization technique schedules test cases in an execution order according to some criteria and provide other methods to reduce regression testing costs. In prioritization, the most powerful algorithm is the nature-inspired algorithm [7]. Since there are several techniques inspired by the nature that can be used in test case prioritization, this paper will compare two of the most used techniques for prioritizing test cases. One of the techniques is Firefly Algorithm (FA), which has been widely used in prioritization techniques. Sahoo et al. [8] had also implemented the FA technique in their paper, including Khatibsyarbini et al. [9], who used the same technique in their study. Indeed, this technique has been widely used for prioritization test cases to obtain better performance. Another technique called Ant Colony Optimization (ACO), which is based on the ant colony nature, has also been implemented as a prioritization technique. For instance, this ACO technique has been implemented by Zhang *et al*. [10] in their paper for prioritizing test cases.

Test case prioritization (TCP) aims to order a set of test cases to attain an early optimization based on preferred properties. TCP helps to find the suitable variation based on

a series of test cases. Once the TCP has been executed, it can produce optimized outcomes as well as reveal faults earlier.

Test cases can be ranked based on their randomness, optimality and branch coverage. Islam et al. [11] presented the test cases prioritization based on latent semantic indexing. The regression test suite can be subjected to a variety of prioritizing criteria with the goal of meeting a specific requirement. Prioritization approaches based on one or more of the chosen criteria have been applied in a variety of ways. Among the most powerful optimization algorithms are those inspired by nature [7]. Hence, this paper will use Ant Colony Optimization (ACO) and Firefly Algorithm (FA) to prioritize the given test cases. The ACO technique is a process based on the real-life of ants and serves as an adaptive meta-heuristic optimization method. This method is inspired by the behaviour of ants in nature; after finding their food source, the ants will carry the food back to their nest. In returning to the nest, the ants will be guided with the smell of pheromones that they left while going out to find the food source. This pheromone path helps the ants find the shortest path between their nest and food source [1]. Besides, various combinatorial optimization issues have also been successfully solved using the ACO approach, such as test data generation [12]. Furthermore, this technique presents a positive feedback parallel mechanism with several benefits, including high robustness, a superior distributed computer system, and ease of interaction with other methods. In several cases, the ACO technique has also effectively handled complicated optimization challenges, thus becoming a hub for research in the field of intelligent optimization. The ACO approach has been used to solve a variety of combinatorial optimization issues, including the travelling salesman problem [13], target assignment [14] and test data generation [15]. For test case generation [4], the tool helps in reducing the generation of test cases. Reducing on generating the test cases has also been discussed in [3]. Other techniques for optimization include black hole optimization [16], whale optimization algorithm [17-19], cuckoo search algorithm [20-23] and honey bee optimization algorithm [24, 25], which is based on artificial bee colony algorithm [26]. These techniques can also be customized and hybrid in optimizing the selection of test cases, such as in [27].

## III. TEST CASES PRIORITIZATION TECHNIQUES

For the test cases prioritization techniques, two algorithms will be reviewed. They are the ant colony optimization algorithm and firefly algorithm.

### *A. Ant Colony Optimization Algorithm*

Dorigo [11] presented Ant Colony Optimization as one of the adaptive meta-heuristic optimization approaches. The goal of the ACO technique is to shorten the path and reduce the time in searching for a fault. Fig. 1 shows the pseudocode of the ACO technique and the steps taken in the process.

```
Begin
    Initialize
    While stopping criterion not satisfied do
        Position each ant in starting node
        Repeat
            For each ant do
                Choose next node by applying the state transition rule
                Apply step by step pheromone update
            End for
        Until every ant has built an solution
        Update best solution
        Apply offline pheromone update
    End while
End
```

**Fig. 1 Ant colony optimization algorithm**

Based on Fig. 1, for test cases prioritization, the state transition rule is used for each node to update the pheromone rule.

### *B. Firefly Algorithm*

Fireflies will get attracted to the lights flashed by nearby fireflies. The flashing lights can be defined by associating them with an optimization objective function, which allows for the creation of a new optimization technique [13]. Fig. 2 shows the pseudocode of the Firefly Algorithm.

```
Objective function f(x), x = (x1,.. xd)^T
Generate initial population of fireflies xi (i = 1, 2, ..., n)
Light intensity Ii at xi is determined by f(xi)
Define light absorption coefficient γ
while (t < MaxGeneration)
for i =1: n all n fireflies
  for j =1: i all n fireflies
      if (Ij > Ii),Move firefly i towards j in d-dimension; end if
      Attractiveness varies with distance r via exp[−γr]
      Evaluate new solutions and update light intensity
  end for j
end for i
Rank the fireflies and find the current best
end while
Postprocess results and visualization
```

**Fig. 2 Firefly algorithm**

Based on Fig. 2, the algorithm begins with the derivation of an objective function at the beginning of the selection. Subsequently, the distance matrix between the firefly agent (FA) and its brightness is calculated to identify which one is used to determine each firefly's attractiveness. A firefly's subsequent movement will be determined by the brightness value. When all the fireflies have been visited, the movement comes to a halt, and all movements are tracked. Finally, the optimal firefly sequence is determined by the shortest distance.

## IV. RESULTS AND DISCUSSION

The ant colony optimization algorithm and the firefly algorithm are implemented using two case studies in order to see which technique is better in the test cases prioritization. The performance of both techniques will be evaluated, and the performance evaluation is based on the execution time, Average Percentage of Faults Detected (APFD), and fault coverage. These parameters are often used for performance evaluation of the techniques and can be found in the literature [6].

### *A. DataSet*

In this study, the two techniques will be applied to two datasets, namely Case Study One and Case Study Two. Each Case study consists of a graph with data such as destinations, paths, and the cost for each path. The graph for each case study is different; Case Study One has six destinations, and Case Study Two has five destinations. The original ordering for the graph is {N1, N2, N3, N4, N5, N6}. The graph data were executed and turned into an adjacency matrix for the algorithm of each technique to prioritize. Fig. 3 shows the graph dataset for Case Study One.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 11 | 13 | 10 | 8 | 12 |
| 11 | 0 | 12 | 7 | 18 | 9 |
| 13 | 12 | 0 | 9 | 12 | 19 |
| 10 | 7 | 9 | 0 | 17 | 21 |
| 8 | 18 | 12 | 17 | 0 | 13 |
| 12 | 9 | 19 | 21 | 13 | 0 |

**Fig. 3 DataSet for case study one**

Meanwhile, Case Study Two consists of five destinations and each destination are connected with a path. Each path has its own cost, which can also be defined as distance. The original ordering for the graph is {N1, N2, N3, N4, N5}. The graph for this case study was transformed into an array to be executed by ACO, and FA. Fig. 4 shows the dataset of the graph for Case Study Two.

| | | | | |
|---|---|---|---|---|
| 0 | 11 | 13 | 12 | 15 |
| 11 | 0 | 4 | 16 | 9 |
| 13 | 14 | 0 | 10 | 15 |
| 12 | 16 | 10 | 0 | 17 |
| 15 | 9 | 15 | 17 | 0 |

**Fig. 4 DataSet for case study two**

The datasets in Fig. 3 and Fig. 4 will be used in both techniques. However, there are significant differences between the two techniques by which the algorithm in the FA technique needs to have an objective function to solve the problem compared to the ACO technique, which can directly solve the problem. Therefore, for the FA technique, the objective function is encoded as light intensity. Fig. 5 and Fig. 6 show the data for intensity for the FA algorithm in Case Studies One and Two, respectively.

| | | | | | |
|---|---|---|---|---|---|
| -0.124 | -0.619 | 0.371 | -0.495 | -0.248 | -0.743 |
| -0.005 | -0.025 | -0.015 | -0.020 | -0.010 | -0.030 |
| -0.021 | -0.106 | -0.064 | -0.085 | -0.042 | -0.127 |
| 0.059 | 0.293 | 0.176 | 0.235 | 0.117 | 0.352 |
| 0.054 | 0.270 | 0.162 | 0.216 | 0.108 | 0.324 |
| 0.026 | 0.130 | 0.078 | 0.104 | 0.052 | 0.156 |

**Fig. 5 Intensity for firefly algorithm in case study one**

Through light intensity, the number of fireflies is based on the matrix of the intensity graphs. In the first step of this technique, the related library will be imported, and the time is subsequently recorded at the beginning of the process. Next, all of the fireflies will be released into the intensity graphs, and the fireflies will move to the solution. To find the solution from the intensity graphs, the fireflies will be released randomly, and they will be attracted to the highest light intensity. The location of the fireflies can be assumed as the solution for the optimization problem. All of the paths travelled by the fireflies will be recorded and sorted to identify the shortest path. After all, fireflies have completed the iterations, and the route has been taken, the time measurement will be stopped and calculated. The last step includes printing out the output, such as the shortest path, the execution time, and the total costs of the routes taken.

| | | | | |
|---|---|---|---|---|
| -0.066 | -0.264 | -0.198 | -0.330 | -0.132 |
| 0.032 | 0.130 | 0.097 | 0.162 | 0.065 |
| 0.014 | 0.056 | 0.042 | 0.070 | 0.028 |
| 0.029 | 0.115 | 0.086 | 0.143 | 0.057 |
| 0.008 | 0.031 | 0.023 | 0.039 | 0.016 |

**Fig. 6 Intensity for firefly algorithm in case study two**

### *B. Implementation of the Algorithms*

For Case Study One, the path consists of ten artificial ants and six destinations with different routes and weights. After the artificial ants are released into this path, they will

go through the path with 100 iterations. The APFD value for the first case study is 0.783. All routes have been fully covered and the best routes for these test cases are {N1-> N5-> N3-> N4-> N2 ->N6 ->N1}. After all of the iterations had been completed, the artificial ants obtained 57 costs from the covered path. Additionally, the execution time for this case study is 0.4103 seconds. The segmentation of the source code for this execution process is shown in Fig. 7.

Case Study Two consists of ten artificial ants. However, the path only consists of five destinations with different routes and weights. After the artificial ants have been released into this path, they will go through the path with 100 iterations. The APFD value for the second case study is 0.780. All routes have also been fully covered, and the best routes for these test cases are {N1-> N2-> N5-> N3-> N4 ->N1}. After all the iterations had been completed, the artificial ants obtained 57 costs from the covered path. In addition, the execution time for this case study is 0.3312 seconds. The segmentation of the source code for this execution process is shown in Fig. 8.

```
for ite in range(iteration):
    rute[:,0] = 1
    for i in range(m):
        temp_visibility = np.array(visibility)
        for j in range(n-1):
            combine_feature = np.zeros(6)
            cum_prob = np.zeros(6)
            cur_loc = int(rute[i,j]-1)
            temp_visibility[:,cur_loc] = 0
            p_feature = np.power(pheromne[cur_loc,:],beta)
            v_feature = np.power(temp_visibility[cur_loc,:],alpha)
            p_feature = p_feature[:,np.newaxis]
            v_feature = v_feature[:,np.newaxis]
            combine_feature = np.multiply(p_feature,v_feature)
            total = np.sum(combine_feature)
            probs = combine_feature/total
            cum_prob = np.cumsum(probs)
            r = np.random.random_sample()
            city = np.nonzero(cum_prob>r)[0][0]+1
            rute[i,j+1] = city

        left = list(set([i for i in range(1,n+1)])-set(rute[i,:-2]))[0]
        rute[i,-2] = left

    rute_opt = np.array(rute)
    dist_cost = np.zeros((m,1))
```

**Fig. 7 Implementation of case study one**

```
intensityGraph = ([[-0.124, -0.619, -0.371, -0.495, -0.248, -0.743]
         ,[-0.005, -0.025, -0.015, -0.020, -0.010, -0.030]
         ,[-0.021, -0.106, -0.064, -0.085, -0.042, -0.127]
         ,[0.059, 0.293, 0.176, 0.235, 0.117, 0.352]
         ,[0.054, 0.270, 0.162, 0.216, 0.108, 0.324]
         ,[0.026, 0.130, 0.078, 0.104, 0.052, 0.156]])

resultantGraph = np.dot(costGraph,intensityGraph)

for i in range(city):
    nodeAlpha = chr(ord('@')+int(round(resultantGraph[0][i])))
    node = int(round(resultantGraph[0][i]))
    bestRoute.append(node)
    bestRouteAlpha.append(nodeAlpha)
bestRoute.append(bestRoute[0])
bestRouteAlpha.append(bestRouteAlpha[0])

for j in range(city):
    bestCost = bestCost + costGraph[bestRoute[j]-1][bestRoute[j+1]-1]
```

**Fig. 8 Implementation of case study two**

### *C. Comparison of the Results*

In order to find the best technique for the prioritization of test cases, the important part is to know what needs to be measured at the evaluation parameters. Based on the review of related papers and research, the most frequently selected evaluation parameters by other authors are the Average Percentage of Faults Detected (APFD) and the execution time. The APFD evaluation method quantifies the fault detection rates, and the value ranges from 0 to 100, which means that a greater value signifies a better fault detection rate. As such, the technique that achieves a high APFD value indicates good performance, while the execution time is evaluated based on the minimum time taken for the execution process, by which the less time taken indicates good technique performance. Another evaluation parameter is the fault coverage; if the technique can cover all faults in the test cases, then it indicates good performance.

To discover faults in test cases and measure how rapidly a prioritized test suite detects the fault, the average percentage of fault detected (APFD) is used. The fault detection rate will be represented by the APFD values; if the fault detection rate is faster, then the APFD value is also high. The APFD results for the Ant Colony Optimization technique are 0.783 for Case Study One (ACO 1) and 0.780 for Case Study Two (ACO 2). The Firefly Algorithm technique also achieved the same APFD values as the ACO technique, with 0.783 for Case Study One (FA 1) and 0.780 for Case Study Two (FA 2). Table 1 shows the difference in APFD values for each technique and case study.

**Table 1. Comparison of APFD results for ACO and FA**

| Technique | APFD Value |
|---|---|
| ACO 1 | 0.783 |
| ACO 2 | 0.780 |
| FA 1 | 0.783 |
| FA 2 | 0.780 |

Based on Table 1, there are no differences between the ACO technique and the FA technique. The APFD values achieved by the ACO and FA techniques are about the same.

In any system, execution time is the most important element to be considered. Based on the results, the ACO technique achieved 0.4103 seconds for the first case study (ACO 1) and 0.3312 seconds for the second case study (ACO 2). Meanwhile, the FA technique showed the lowest execution time for both case studies with 0.001000 seconds for Case Study One (FA 1) and 0.002000 seconds for Case Study Two (FA 2). Table 2 shows the differences in the execution time of each technique.

**Table 2. Results for execution time**

| Technique | Time (Seconds) |
|---|---|
| ACO 1 | 0.4103 |
| ACO 2 | 0.3312 |
| FA 1 | 0.001000 |
| FA 2 | 0.002000 |

Based on Table 2, there are significant differences in the execution time for both techniques. The Firefly algorithm technique showed the shortest execution time for both test suites in prioritizing test cases as well as outperforming the Ant Colony Optimization technique in terms of time, as shown in Fig. 9.

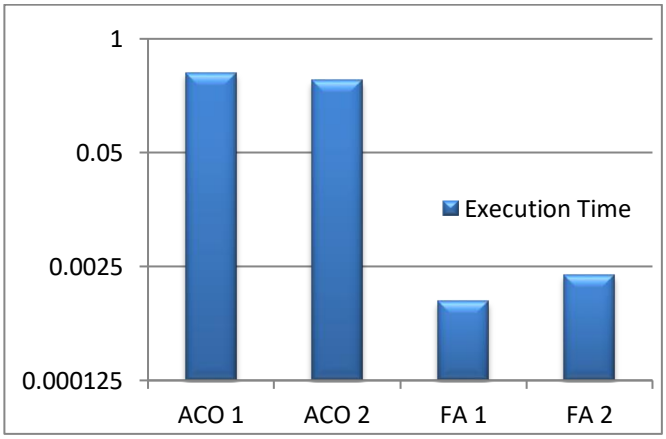


**Fig. 9 Execution time for case studies**

Fault coverage is important in the prioritization of test cases. From the execution of the techniques, the results showed that the Ant Colony Optimization technique had reached 90% of the fault coverage for both case studies (ACO 1 and ACO 2). Meanwhile, as for the Firefly Algorithm technique, the solution is already available through the use of the objective function. This further enables the Firefly Algorithm technique to achieve the full fault coverage through the intensity graphs, and it does not need much to show that both of the case studies (FA 1 and FA 2) achieved a 100% fault coverage. The results for the fault coverage are shown in Table 3.

**Table 3. Results for fault coverage**

| Technique | Fault Coverage |
|---|---|
| ACO 1 | 90% |
| ACO 2 | 90% |
| FA 1 | 100 % |
| FA 2 | 100 % |

Based on the fault coverage results, there are differences between both techniques and the differences are illustrated in Fig. 10.

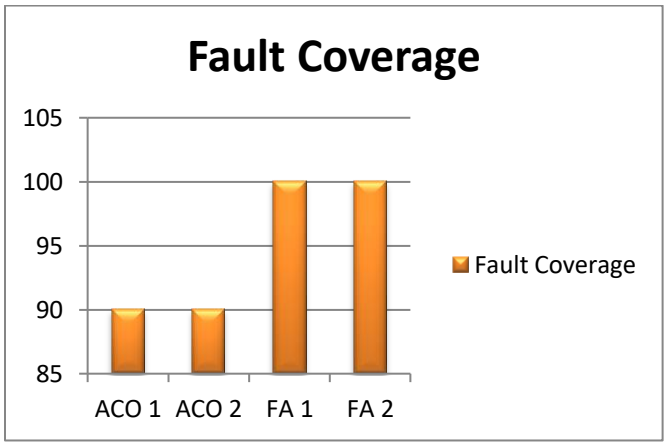


**Fig. 10 Fault coverage for case studies**

The results obtained from the case studies on the prioritization techniques reveal the performance of each technique and are further analyzed comparatively. Each technique will be measured based on its performance to obtain the best technique.

Based on the results of each technique, Firefly Algorithm is the best technique in prioritizing test cases since it has the lowest execution time and covers 100% of the fault coverage. The ranking of all results presented by each technique is summarized in Table 4.

**Table 4. Comparison of results for ACO and FA**

| Technique / Measurement | ACO 1 | ACO 2 | FA 1 | FA 2 |
|---|---|---|---|---|
| APFD | 0.783 | 0.780 | 0.783 | 0.780 |
| Execution Time | 0.4103 | 0.3312 | 0.00100 | 0.00200 |
| Fault Coverage | 90% | 90% | 100% | 100% |

Based on Table 4, the Firefly Algorithm and Ant Colony Optimization techniques have achieved the same percentage for fault coverage. However, in terms of the execution time, the FA technique took lesser time in the execution process. Thus, the FA technique is ranked first in the two measurements of execution time and fault coverage. This also shows that the FA technique has good performance and outperforms the ACO technique. However, even though the FA technique has the lowest execution time, it still cannot outperform the ACO technique in terms of problem-solving because the ACO technique can solve the test case prioritization directly rather than the FA technique, which needs the objective function to solve the problem. This means that the FA technique does not have the same ability as the ACO technique, which can directly solve the problem without using the objective function. Thus, Ant Colony Optimization still has better prioritization performance.

## V. CONCLUSION AND FUTURE WORKS

In this paper, two techniques with two case studies for each technique have been applied in the evaluation of good technique performance in prioritizing test cases involving Ant Colony Optimization and Firefly Algorithm. The evaluation method has been applied for each technique, and the results of each technique have been discussed and further compared in order to evaluate its performance. Based on the results, Firefly Algorithm is the best technique in prioritizing test cases since it has the lowest execution time and covers 100% of the fault coverage compared with Ant Colony Optimization. However, in terms of problem-solving, Ant Colony Optimization has the ability to prioritize the test cases directly within the algorithm without using the objective function.

For future works, several techniques for test case prioritization can be used for the purpose of hybriding two or more techniques. The hybrid technique in prioritizing test cases promises future work for software testing in order to improve the accuracy and redundancy of generating test cases. These techniques for optimizing the generation of test cases include black hole optimization, whale optimization algorithm, cuckoo search algorithm and honey bee optimization algorithm.

## ACKNOWLEDGMENT

The authors would like to thank Universiti Tun Hussein Onn Malaysia (UTHM) for supporting this research. The authors received funding for this study from Industry Grant under Grant Vote No M081.

## REFERENCES

[1] D. Gao, X. Guo and L. Zhao, Test case prioritization for regression testing based on ant colony optimization., 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), (2015) 275-279.

[2] W. Su, Z. Li, Z. Wang and D. Yang, A Meta-heuristic test case prioritization method based on the hybrid model, 2020 International Conference on Computer Engineering and Application (ICCEA), (2020) 430-435.

[3] R. Ibrahim, M. Ahmed, R. Nayak and S. Jamel, Reducing redundancy of test cases generation using code smell detection and refactoring. Journal of King Saud University - Computer and Information Science, 32(3) (2020) 367-374.

[4] R. Ibrahim, A. A. B. Amin, S. Jamel and J. A. Wahab, EPiT: A software testing tool for generating of test cases automatically, International Journal of Engineering Trends and Technology, 68(7) (2020) 8-12.

[5] O. Dahiya and K. Solanki, An Efficient APHT Technique for Requirement-Based Test Case Prioritization, International Journal of Engineering Trends and Technology, 69(4) (2021) 215-227 .

[6] O. Dahiya and K. Solanki. An Efficient Requirement-Based Test Case Prioritization Technique using Optimized TFC-SVM Approach, International Journal of Engineering Trends and Technology, 69(1) (2021) 5-16.

[7] X. S. Yang. Firefly algorithms for multimodal optimization. In: O. Watanabe and T. Zeugmann (eds) Stochastic Algorithms: Foundations and Applications. SAGA 2009. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. 5792 (2009).

[8] R. K. Sahoo, D. P. Mohapatra and M.R. Patra. A Firefly algorithm-based approach for automated generation and optimization of test cases., International Journal of Computer Sciences and Engineering, 4(8) (2016) 54-58.

[9] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi and R. Tumeng. "Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 93 (2018) 74-93.

[10] W. Zhang, Y. Qi, X. Zhang, B. Wei, M. Zhang et al., On test case prioritization using ant colony optimization algorithm, 2019 IEEE 21st International Conference on High-Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), (2019). 2767-2773.

[11] M. M. Islam, A. Marchetto, A. Susi and G. Scanniello, A multi-objective technique to prioritize test cases based on latent semantic indexing, 2012 16th European Conference on Software Maintenance and Reengineering, (2012) 21-30.

[12] K. Ayari, S. Bouktif and G. Antoniol, Automatic mutation test input data generation via ant colony. GECCO'07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, (2007) 1074-1081.

[13] M. Dorigo and L.M. Gambardella, Ant colonies for the travelling salesman problem. Biosystems. 43(2)(1997) 73-81.

[14] M. L. Mohd-Shafie, W. M. N. Wan-Kadir, M. Khatibsyarbini and M. A. Isa, Model-based test case prioritization using selective and even-spread count-based methods with scrutinized ordering criterion. PLOS ONE, 15(2) (2020).

[15] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, H. N. A. Hamed and M. D. Mohamed Suffian, Test case prioritization using firefly algorithm for software testing, IEEE Access, 7 (2019) 132360-132373.

[16] H. N. Nsaif Al-Sammarraie and D. N. A. Jawawi, Multiple black holes inspired meta-heuristic searching optimization for combinatorial testing, IEEE Access, 8 (2020) 33406-33418.

[17] S. Mirjalili and A. Lewis. The Whale Optimization Algorithm. Advances in Engineering Software 95 (2016) 51-67.

[18] A. A. Hassan, S. Abdullah, K. Z. Zamli and R. Razali, Combinatorial test suites generation strategy utilizing the whale optimization algorithm, IEEE Access, 8 (2020) 192288-192303.

[19] S. M. Bozorgi, S. Yazdani, IWOA: An improved whale optimization algorithm for optimization problems, Journal of Computational Design and Engineering, 6(3) (2019) 243-259.

[20] X.S. Yang and S. Deb, Cuckoo search: recent advances and applications, Neural Computing & Applications, 24 (2014) 169–174.

[21] A.H. Gandomi, X.S. Yang and A.H. Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems., Engineering with Computers, 29 (2013) 17–35.

[22] M. Mareli and B. Twala, An Adaptive Cuckoo Search Algorithm for Optimisation, Applied Computing and Informatics, 14(2) (2018) 107-115.

[23] P. Lakshminarayana and T.V. Suresh Kumar. Automatic Generation and Optimization of Test Case using Hybrid Cuckoo Search and Bee Colony Algorithm, Journal of Intelligent Systems, 30(1) (2021) 59-72.

[24] D. Rai and K. Tyagi, Regression Test Case Optimization using Honey Bee Mating Optimization Algorithm with Fuzzy Rule-Based, World Applied Science Journal, 31(4) (2014) 654-662.

[25] S. Nayak, C. Kumar, S. Tripathi, N. Mohanty and V Baral, Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base, Soft Computing., 25 (2012). 9925–9942.

[26] D. Karaboga., Artificial Bee Colony Algorithm. Scholarpedia, 5(3) (2010) 6915.

[27] Palak, P. Gulia and N.S. Gill., Optimized Test Case Selection using Scout-less Hybrid Artificial Bee Colony Approach and Crossover Operator, International Journal of Engineering Trends and Technology, 69(3) (2021) 39-45.