© Universiti Tun Hussein Onn Malaysia Publisher's Office

JSCDM

http://penerbit.uthm.edu.my/ojs/index.php/jscdm

e-ISSN : 2716-621X

Journal of Soft Computing and Data Mining

Comparative Analysis of Test Case Prioritization Using Ant Colony Optimization Algorithm and Genetic Algorithm

Muhamad Asyraf Anuar[1], Mohd Zanes Sahid[1*], Nurezayana Zainal[1]

[1]Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author

DOI: https://doi.org/10.30880/jscdm.2023.04.02.005
Received 15 February 2023; Accepted 22 September 2023; Available online 04 October 2023

Abstract: After a software is deployed, every software system will get an upgrade, requiring it to adapt to meet the ever-changing client needs. Thus, regression testing becomes one of the most important operations in any software system. As it is too expensive to repeat the execution of all the test cases available from a previous version of the software system, numerous ways to optimizing the regression test suite have evolved, one of which is test case prioritizing (TCP). This study was carried out to test and compare the effectiveness of evolutionary algorithms and swarm intelligence algorithms, represented by Genetic Algorithm (GA) and Ant Colony Optimization (ACO) algorithms. These algorithms were be implemented to find the Average Percentage Fault Detected (APFD), execution time, and Big O notation, as these are critical aspects in software testing to ensure that high-quality products are produced on time. This study employs data from two separate investigations on comparable issues, denoted as Case Study One and Case Study Two. TCP has been extensively used in recent years, but not much research has been conducted to analyze and evaluate the performance of GA and ACO in a test case prioritization context. The algorithms were compared using APFD and execution time. The APFD and execution time values of 50, 100, 150 and 200 iterations for ACO and GA for both datasets. Both algorithms were determined to work on O(n^2) notation, which indicates they should scale up their execution process similarly on different input scales. Both algorithms performed well in their respective roles. ACO has shown to be more valuable than GA in terms of APFD, whereas GA has shown to be more valuable than ACO in terms of execution time.

Keywords: Test case prioritization, regression testing, Ant Colony Optimization (ACO), Genetic Algorithm (GA)

1. Introduction

According to the ANSI/IEEE 1059 standard [citation required], software testing is a step that occurs at the conclusion of software quality assurance (SQA), when engineers execute the process of analyzing a system or program to determine if it passes or fails a pre-determined condition. Based on the results revealed on the system, such as the number of missing requirements, flaws, and mistakes, the testing process entails assessing the system on numerous aspects, such as reliability, security, and performance. The system or product is tested to find faults and flaws, following which the source of the mistakes is explored, debated, and then corrected or improved. Unfortunately, software testing is often hurried due to how costly and time-consuming the process is, forcing software developers to conclude software testing early due to time and money constraints, degrading software quality [1]. Regression testing, one of the most critical tests, is a method that ensures that any future addition or upgrade to the system will not impair the previously functional system. As the system expands in size over time, it becomes more expensive to run all the test suites in the system, thus different ways are designed to address the regression testing of the test suites. Test case prioritization was one of the ways utilized to overcome the challenge.

*Corresponding author: zanes@uthm.edu.my

The Ant Colony Optimization (ACO) algorithm is a metaheuristic algorithm inspired by ant behavior in nature [citation required]. While the ants are sight handicapped and ineffective on their own, studies show that they may execute complicated tasks by enabling one another to aid them. One of the key concepts is shown when ants release a specific chemical called pheromones to determine the fastest way to a food source or other key destination. An increase in pheromone deposits on a certain route enhances the chance that that path will be followed; the greatest pheromone deposit from the start node may be used to assess a path's optimality [2].

Genetic Algorithm (GA) is another metaheuristic algorithm based on Darwin's theory of evolution, in which the "parents" are chosen from an initial population after exchanging genes, causing crossover, and producing offspring, and mutations are applied to these offspring, which will be passed down to the next generation [citation required]. This procedure will be repeated until a termination condition is satisfied. The program guides the search for the best solutions in the search space by using genetic information from chromosomes [3].

In this study, we presented a test case prioritizing (TCP) study employing GA and ACO algorithms to compare algorithm performance based on average fault percentage detection (AFPD), Big O notation, and execution time when applied to the same datasets. The outcomes will be compared to assess the efficacy of both the GA and ACO algorithms.

2. Related Work

The approach of selecting the best test cases such that it arranges them in a sequence based on characteristics that might boost the effectiveness of the test cases in meeting a performance objective, which is where the TCP process specializes in [4]. Regression testing, one of the most resource intensive yet crucial test for any software may be improved by adopting test case prioritizing strategies, which have been shown to provide better results in terms of identifying objective functions in less time and processing resources than other methods [5].

There are several algorithms inspired by nature that can be used in test case prioritization, as more researchers observe the nature closely to study natural behaviors. Thus, this paper will compare two of the most used algorithms for prioritizing test cases. One of the algorithms is GA, which is one of the widely used algorithms in prioritization techniques. Anu Bajaj et al. [6] had implemented GA in their paper, including Padmnav et al. [7], who used the same algorithm in their study. The next algorithm that will be utilized in this paper is ACO, which is based on the ant colony behavior in finding paths, has also been implemented as a prioritization technique. For instance, ACO algorithm has been implemented by Panwar et al. [8] in their paper for prioritizing test cases.

Thus, this paper will use ACO and GA to prioritize the selected test cases. ACO is a type of swarm intelligence that deals with computational approaches inspired by ant colony behavior. In ACO, the hypothetical ants construct a solution to an existing optimization problem, and information about these solutions is transmitted via a communication technique similar to that employed by actual ants. Aside from TCP, ACO has been found to increase test case creation across a range of testing categories, including regression testing, functional testing, and so on [2]. Because of their environment and size, ants have trouble utilizing their eyesight to choose the optimal way to their objective, therefore they utilize a fluid called pheromones to transmit their journey to other ants. The route with the greatest number of pheromone deposits, which began at the first node, is more likely to be followed.

The next technique, GA is a search strategy based on Darwin's theory of biological evolution's natural selection processes. The premise is that people in a population must compete for limited resources, and those that prevail create more children, which are subsequently passed on to the next generation [6]. GA is widely regarded as one of the finest single-objective algorithms for test case prioritization challenges [8].

Other techniques for optimization include bat algorithm[9], greedy algorithm [10], cuckoo search algorithm [11] and flower pollination algorithm [11]. These techniques can also be customized and hybrid in optimizing the selection of test cases, such as in [12], [13], [14] and [15].

3. Methodology

3.1 Research Framework

In the first stage, all the processes affecting the data that will be utilized throughout the study will be completed in the first stage. This includes locating a research article with acceptable and useable datasets and processing them such that they can be utilized by the suggested methods. For the second stage, the ACO method and GA will be implemented in Python, and the previously generated datasets will be fed through the algorithms and processed until the datasets are successfully prioritized. Finally for the last stage, the metrics of execution time, Average Percentage of Fault Detected (APFD), and Big O Notation are extracted from the process of test case prioritization of both algorithms, which are then compared and analyzed to determine which algorithm is superior in terms of test case prioritization performance. Please introduce Fig. 1 in this paragraph.

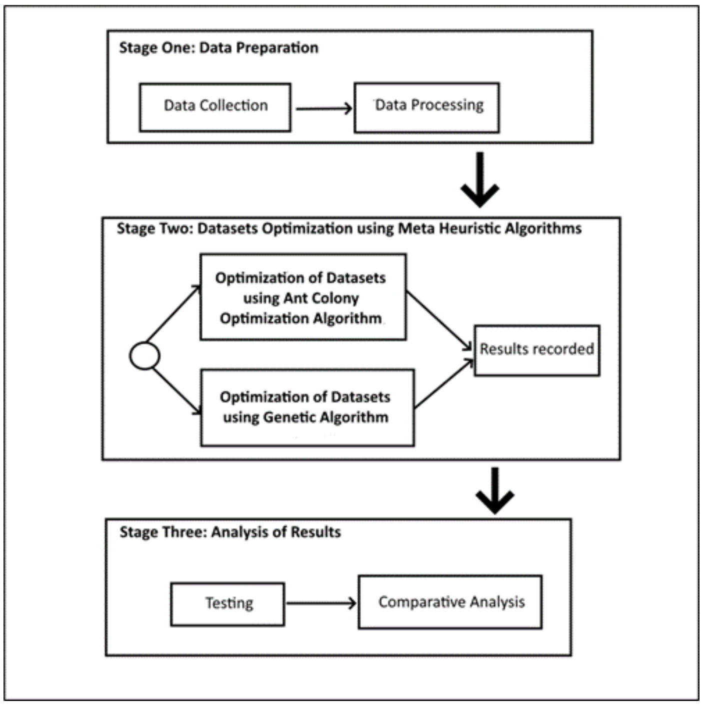
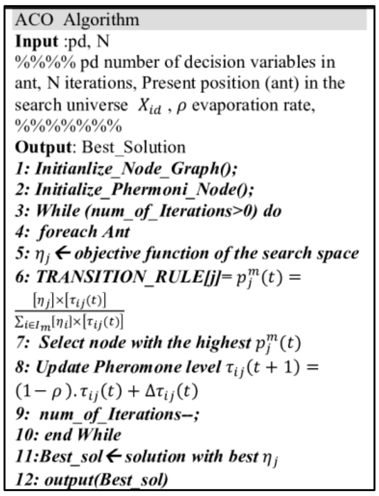

Stage One: Data Preparation
Data Collection
Data Processing
Stage Two: Datasets Optimization using Meta Heuristic Algorithms
Optimization of Datasets using Ant Colony Optimization Algorithm
Optimization of Datasets using Genetic Algorithm
Results recorded
Stage Three: Analysis of Results
Testing
Comparative Analysis

Fig. 1 - Research framework

3.2 Algorithms

3.2.1 Ant Colony Algorithm (ACO)

The ACO algorithm starts with the ants being randomly allocated to test cases. The ants will choose which test case to visit next based on a specified criterion. Each ant remembers its previously travelled route, and on succeeding steps, it picks the way that it has not before visited. After visiting all of the test cases, an ant builds a solution. The probabilistic rule is defined by pheromone values and heuristic information, and the greater the pheromone and heuristic value connected with a route, the more likely the ants will take that road. The pheromone on the trail is updated after all ants have finished their journey. The ACO pseudocode, seen in Figure 2, describes the steps required to implement the algorithm. For the study, the ACO algorithm stages is adopted from a previous work[16]. Please introduce Fig. 2 in this paragraph.

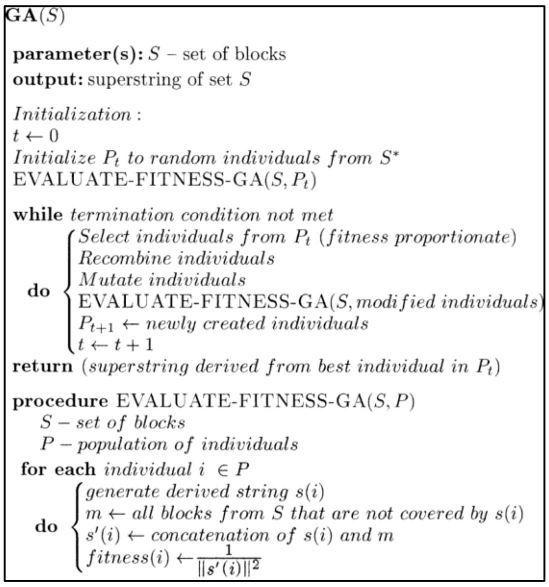


```
ACO  Algorithm
Input :pd, N
%%%% pd number of decision variables in ant, N iterations, Present position (ant) in the search universe $X_{id}$ , $\rho$ evaporation rate, %%%%%%%
Output: Best_Solution
1: Initianlize_Node_Graph();
2: Initialize_Phermoni_Node();
3: While (num_of_Iterations>0) do
4:  foreach Ant
5: $\eta_j \leftarrow$ objective function of the search space
6: TRANSITION_RULE[j]= $p_j^m(t) = \frac{[\eta_j]\times[\tau_{ij}(t)]}{\sum_{i\in I_m}[\eta_i]\times[\tau_{ij}(t)]}$
7:  Select node with the highest $p_j^m(t)$
8: Update Pheromone level $\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t)$
9:  num_of_Iterations--;
10: end While
11:Best_sol $\leftarrow$ solution with best $\eta_j$
12: output(Best_sol)
```

Fig. 2 - ACO pseudocode

3.2.2 Genetic Algorithm (GA)

GA procedure starts with an initial population, P, of individuals. Then, pairs of chosen individuals are cross-bred and mutated to produce a new individual that will be inserted into the population of the next generation. A GA is an approximated algorithm that has no guarantee of convergence. Thus, the search will continue for several predetermined generations until a termination condition is fulfilled. Please introduce Fig. 3 in this paragraph.



```
GA($S$)

parameter(s): $S$ – set of blocks
output: superstring of set $S$

Initialization :
$t \leftarrow 0$
Initialize $P_t$ to random individuals from $S^*$
EVALUATE-FITNESS-GA($S, P_t$)

while termination condition not met
    do {
        Select individuals from $P_t$ (fitness proportionate)
        Recombine individuals
        Mutate individuals
        EVALUATE-FITNESS-GA($S$, modified individuals)
        $P_{t+1} \leftarrow$ newly created individuals
        $t \leftarrow t + 1$
    }
return (superstring derived from best individual in $P_t$)

procedure EVALUATE-FITNESS-GA($S, P$)
    $S$ – set of blocks
    $P$ – population of individuals
  for each individual $i \in P$
    do {
        generate derived string $s(i)$
        $m \leftarrow$ all blocks from $S$ that are not covered by $s(i)$
        $s'(i) \leftarrow$ concatenation of $s(i)$ and $m$
        $fitness(i) \leftarrow \frac{1}{||s'(i)||^2}$
    }
```

Fig. 3 - Genetic Algorithm (GA) pseudocode

3.3 Evaluation Parameters

The key activity for the last part of this study will be the assessment and comparison of the findings for each method. The acquired findings for APFD, Big O notation, and time execution will be analyzed for both algorithms based on the algorithm processes. The parameters will be compared and examined, and a conclusion on the algorithms' performance will be established.

3.3.1 Average Percentage of Fault Detected (APFD)

Rotherm et al. [17] proposed APFD as a metric for assessing prioritizing performance. The weighted mean of the faults covered by the test cases is calculated using APFD value corresponding to the test case's position in the test suite. The mathematical expression is shown in (1).

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + \cdots TF_m}{nm} + \frac{1}{2n} \tag{1}$$

TF_1 is the number of test cases conducted in T before to detect the fault, T is an ordered test suite with n test cases, and F is a collection of m failures identified by T. The following is the APFD formula: The APFD value ranges from 0 to 1, with higher values signifying faster fault detections [18].

3.3.2 Execution Time

The second evaluation parameter is based on the time it took to apply the algorithms. The time is expressed in seconds (s). In this study, the algorithm with the shortest execution time is determined to be the best. The time calculation equation is shown in (2).

$$\text{Execution time} = \text{Start time} \pm \text{End time} \tag{2}$$

3.3.3 Big O Notation

Big-O is a mathematical notation that displays how efficient an algorithm is in the worst-case situation in relation to the amount of its input. We will utilize the frequency count approach to discover Big O. It is the technique of determining how many times a certain statement gets performed. To begin, comments are never performed by the machine, thus they are excluded from the count. This also applies to any declarations made during the algorithm stage. For return statements and assignment statements, it counts as one step. Finally, higher-order exponents take precedence over lower-order exponents. Apart from that, the procedure of counting the code line by line in their programming stage will be covered later in the article.

3.4 Datasets

Two sets of datasets were selected to carry out this study. Case Study One consists of 15 test cases with 15 faults from the work by Palak et al. [19] and Case Study Two consists of 10 test cases with 10 faults from the work by Pradeepa et al. [20]. Table 1 summarizes the test cases and their respective flaws for Case Study One, whereas Table 2 summarizes the test cases and their respective faults for Case Study Two. For Case Study One, 10 number of populations and 10 number of ants were allocated based on a set of test suite which consists of fifteen test cases as shown in Table 1 based on sequential order, represented as T = {T1,T2,T3,T4,T5,T6,T7,T8,T9,T10, T11,T12,T13,T14,T15}. Then, the faults were represented by F = {F1,F2,F3,F4,F5,F6,F7, F8,F9,F10,F11,F12, F13,F14,F15}. For Case Study Two, 10 number of populations and 10 number of ants were allocated based on a test suite which consists of ten test cases as shown in Table 2 based on sequential order, represented as T = {T1,T2,T3,T4,T5,T6,T7,T8,T9,T10 }. Then, the faults were represented by F = {F1,F2,F3,F4,F5,F6,F7, F8,F9,F10 }.

Table 1 - Case study one

Test Case	**Fault**														
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
T1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1
T2	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0
T3	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
T4	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0
T5	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0
T6	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0
T7	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
T8	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0
T9	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0
T10	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0
T11	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0
T12	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0
T13	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0
T14	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
T15	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0

Table 2 - Case study two

Test Case	**Fault**									
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
T1	0	0	0	0	0	1	1	0	0	0
T2	0	1	0	0	1	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	1	0	0	0	0	1	0
T5	0	1	0	0	0	1	0	0	0	0
T6	1	0	0	0	0	0	0	0	0	0
T7	1	0	0	0	0	0	0	1	0	0
T8	0	0	0	0	0	0	1	0	1	0
T9	0	0	0	1	0	0	0	0	0	1
T10	0	1	0	0	0	0	1	0	0	0

4. Results and Analysis

The performance measures taken into consideration for this study are APFD, execution time and Big O Notation which have been recorded by the successful implementation and execution of the algorithms with the datasets. A comparative analysis was put together and discussed in this section to determine the effectiveness of both the algorithms.

APFD value is used to calculate the weighted mean of faults covered by test cases in accordance with the test case position of the test suite. If the fault detection rate is faster, the APFD value is higher. The higher APFD value represents the better performing algorithm. Based on the study conducted using GA and ACO algorithms on two cases studies, the APFD values were calculated based on the prioritized test cases. Table 3 represents the values of the prioritized test cases.

Table 3 - APFD comparison

Algorithm / Iteration	Case Study One APFD				Case Study Two APFD			
	50	**100**	**150**	**200**	**50**	**100**	**150**	**200**
GA	0.59	0.67	0.70	0.71	0.51	0.60	0.66	0.67
ACO	0.60	0.65	0.71	0.72	0.52	0.64	0.70	0.70

Based on the comparison of APFD values illustrated in Table 3, it is shown that the APFD values for ACO algorithm for most of the time are higher than the values of APFD for GA algorithm for both Case Study One and Two, which are 0.72 and 0.70 respectively. From the results, we can conclude that 150 iterations are the best number of iterations to be applied to both algorithms as from Table 4.5 we can see at iterations 200 the increase in APFD value is small.

In the process of SDLC, time plays a very important role in ensuring the product can be delivered within the project timeline. Hence, in software testing, execution time should always be taken into consideration to avoid delays. Time module was used to record the execution time of both algorithms. The execution time of iteration of 150 was used for both algorithms since we had concluded that 150 iterations is the most optimum value for both algorithms. Table 4 shows an execution time comparison between Case Study One and Case Study Two.

Table 4 - Execution time comparison

Algorithm	Case Study One (s)	Case Study Two (s)
GA	0.21	0.19
ACO	0.54	0.35

From the data collected in Table 4, it can be observed that ACO took a longer time which is 0.54 seconds, while GA took only 0.21 seconds. Case Study Two also showed the same outcome whereby ACO and GA took 0.35 seconds and 0.19 seconds respectively. A conclusion can be drawn that GA has a faster execution time than ACO. Big O Notation is a metric for determining an algorithm's efficiency as it gives an approximation of how long your code run on various sets and sizes of inputs. We can also say that it is a way to measure how effectively the code scales as your input size increases or decreases. Table 5 shows Big O notation comparison between Case Study One and Case Study Two.

Table 5 - Big O notation comparison

Algorithm	Case Study One Big O	Case Study Two Big O
GA	$O(n^2)$	$O(n^2)$
ACO	$O(n^2)$	$O(n^2)$

From the step count method calculation performed earlier, we can see that both algorithms, GA and ACO have the same Big O notation. From Table 5, it can be seen that ACO have longer execution time than GA which resulted 0.05 seconds of difference. This difference is acceptable since both algorithms have the same big O notation values. Therefore, we can conclude that the difference in the execution time between the two algorithms would not be large.

5. Conclusion

This study was conducted with the idea of implementing GA and ACO algorithms on Test Case Prioritization to acquire and compare APFD and Execution Time of both algorithms. The algorithms used were GA and ACO in which they were implemented on two different datasets, namely Case Study One and Case Study Two. Case Study One consists of a set of 15 test cases with 15 faults, while Case Study Two consists of 10 test cases and 10 faults. The results were compared. In conclusion, both GA and ACO performance did not differ far between each other. When compared in terms of APFD, GA had shown to give a higher value of fault detection for both the datasets while in terms of execution time, GA also converged faster during the execution. For the algorithms Big O notation, both have $O(n^2)$ notation. Although GA has the upper hand in terms of performance, the difference between them is insignificant. Hence, it can be concluded that both GA and ACO algorithms are similarly good when implemented on test case prioritization. For future research, efforts can be directed into a similar study by proposing hybrid, improved or enhanced GA and ACO algorithms. Besides that, different algorithms such as Black Hole Optimization Algorithm [citation required], Butterfly Algorithm [citation required], Grey Wolf Algorithm [citation required], and many others can also be implemented. Finally, although focusing on implementing the algorithms on TCP is a great move, research can also use these algorithms on a different research path such as Test Case Optimization or Test Case Selection.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, for supporting this work.

References

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

[2] SS, V. C. (2020). An ant colony optimization algorithm based automated generation of software test cases. In Advances in Swarm Intelligence: 11th International Conference, ICSI 2020, Belgrade, Serbia, July 14–20, 2020, Proceedings 11 (pp. 231-239). Springer International Publishing.

[3] Bajaj, A., & Sangwan, O. P. (2019). A systematic literature review of test case prioritization using genetic algorithms. IEEE Access, 7, 126355-126375.

[4] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999, August). Test case prioritization: An empirical study. In Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360) (pp. 179-188). IEEE.

[5] Shaheamlung, G., & Rote, K. (2020, June). A comprehensive review for test case prioritization in software engineering. In 2020 International Conference on Intelligent Engineering and Management (ICIEM) (pp. 331-336). IEEE.

[6] Bajaj, A., & Sangwan, O. P. (2019, February). Study the impact of parameter settings and operator’s role for genetic algorithm based test case prioritization. In Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India.

[7] Padmnav, P., Pahwa, G., Singh, D., & Bansal, S. (2019, January). Test case prioritization based on historical failure patterns using ABC and GA. In 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 293-298). IEEE.

[8] Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2018). A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Transactions on Software Engineering, 46(6), 674-696.

[9] Bajaj, A., & Sangwan, O. P. (2021). Test case prioritization using bat algorithm. Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 14(2), 593-598.

[10] Li, F., Zhou, J., Li, Y., Hao, D., & Zhang, L. (2021). Aga: An accelerated greedy additional algorithm for test case prioritization. IEEE Transactions on Software Engineering, 48(12), 5102-5119.

[11] Dhareula, P., & Ganpati, A. (2022). Comparative Analysis of Efficacious Metaheuristic Technique with Genetically Modified-Flower Pollination Algorithm (GM-FPA) for Test Case Prioritization in Regression Testing.

[12] Silvarajoo, A., & Alkawaz, M. H. (2022). A Framework for Optimizing Software Regression Test Case based on Modified-Ant Colony Optimization (M-ACO). In ITM Web of Conferences (Vol. 42, p. 01007). EDP Sciences.

[13] Akila, T. K., & Malathi, A. (2022). Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing. International Journal of Advanced Technology and Engineering Exploration, 9(88), 384.

[14] Ufuktepe, E., Ufuktepe, D. K., & Karabulut, K. (2021, July). MUKEA-TCP: A mutant kill-based local search augmented evolutionary algorithm approach for test case prioritization. In 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 962-967). IEEE.

[15] Bajaj, A., & Abraham, A. (2022, July). Test Case Prioritization and Reduction Using Hybrid Quantum-behaved Particle Swarm Optimization. In 2022 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.

[16] Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. IEEE computational intelligence magazine, 1(4), 28-39.

[17] Chen, J., Zhu, L., Chen, T. Y., Towey, D., Kuo, F. C., Huang, R., & Guo, Y. (2018). Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. Journal of Systems and Software, 135, 107-125.

[18] Lima, J. A. P., & Vergilio, S. R. (2020). Test Case Prioritization in Continuous Integration environments: A systematic mapping study. Information and Software Technology, 121, 106268.

[19] Palak, P., & Gulia, P. (2019). Hybrid swarm and GA based approach for software test case selection. International Journal of Electrical and Computer Engineering, 9(6), 4898.

[20] Pradeepa, R., & VimalDevi, K. (2013). Effectiveness of testcase prioritization using apfd metric: Survey. In International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications®(IJCA) (pp. 0975-8887).