Journal of Informatics and Web Engineering

Vol. 5 No. 1 (February 2026) eISSN: 2821-370X

Test Case Prioritization Using Ant Colony Optimization to Improve Fault Detection and Time

Nurezayana Zainal[1*], Muhammad Sh Salleh[2], Nur Atiqah Wahidah Sulaiman[3], Ammar Alazab[4], Nur Liyana Sulaiman[4]**

[1,2,3,5]Department of Software Engineering, Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, Johor, Malaysia

[4]Centre for Artificial Intelligence Research and Optimisation (AIRO), Torrens University Australia, Adelaide, Australia

**corresponding author: (nurezayana@uthm.edu.my; ORCiD: 0000-0002-5424-841000)*

*** corresponding author: (ammar.alazab@torrens.edu.au ORCiD -0000-0001-9443-937X)*

Abstract - Regression testing plays a critical role in ensuring the reliability and quality of software following continuous integration and development. However, executing all test cases during regression testing can be time-consuming and resource-intensive. Test Case Prioritization (TCP) addresses this challenge by determining an optimal execution order of test cases that maximizes early fault detection while minimizing execution time. Optimization algorithms contribute significantly to enhancing the effectiveness of TCP while utilizing limited resources. This study proposes an Ant Colony Optimization (ACO) algorithm to address the TCP problem, leveraging its strength in navigating complex search spaces inspired by the foraging behavior of real ant colonies. It involves four phases: dataset selection, dataset conversion, algorithm implementation, and performance evaluation. ACO was implemented and evaluated on two datasets (Case Study One and Case Study Two) of differing sizes and complexity. The results demonstrate its potential to improve testing efficiency and effectiveness with limited resources using the Average Percentage Fault Detected (APFD) and execution time. Case Study One, which involved a larger dataset, achieved a higher APFD (0.6911), but required more iterations and execution time (0.3733 s). In contrast, Case Study Two, with fewer test cases and faults, demonstrated a faster convergence and execution time (0.2596 s), with a slightly lower APFD (0.6700). These findings demonstrate a trade-off between early fault detection and execution efficiency, indicating that dataset characteristics such as size and fault density influence the performance of the algorithm.

Keywords—Software Testing, Test Case Prioritization, Ant Colony Optimization, Meta Heuristic, Test Case Optimization

Received: 17 July 2025; Accepted: 12 October 2025; Published: 16 February 2026

1. INTRODUCTION

Software testing plays a critical role in ensuring the reliability and quality of software systems by executing the software and identifying potential defects. A system that fails to produce the expected results is considered faulty and requires debugging and corrective action [1]. Despite its importance, software testing remains one of the most overlooked phases of the software development lifecycle [2]. Time constraints and high costs often lead to rushed testing processes, resulting in premature software release and diminished product quality [2]. Among the various testing processes, regression testing is essential to verify that the existing functionalities remain unaffected by new changes. However, regression testing frequently encounters challenges, particularly because of the poor communication between stakeholders and developers. This disconnect often results in redundant testing of the entire test suite, a resource-intensive and time-consuming effort that can erode developer confidence and negatively impact the overall system quality. Furthermore, ineffective regression testing increases the risk of defects being deployed in production, ultimately compromising product reliability and client trust. Approaches such as test-case reduction, selection, and prioritization are always being considered.

One effective approach to address the challenges in regression testing is the application of Test Case Prioritization (TCP) techniques, which enhance the efficiency and effectiveness of software testing processes [2]. TCP techniques prioritize test cases according to their significance, ensuring that the most critical tests are executed early in the testing cycle [3]. This prioritization facilitates early fault detection, provides timely feedback to developers, and improves the overall efficiency of the testing process [4]. Among these three approaches, TCP is the most used because it does not eliminate or select test cases. Instead, it simply rearranges them so that the most critical ones are checked first [5]. However, traditional TCP approaches such as backtracking and dynamic programming often incur significant computational costs, particularly for large datasets. In contrast, metaheuristic algorithms, a class of optimization techniques, offer an efficient alternative by providing near-optimal solutions with reduced computational effort [6]. These algorithms are especially advantageous in regression testing scenarios, where limited resources and complex test-case dependencies are common challenges.

Among various metaheuristic algorithms, ACO is a promising approach for solving TCP problems. Ants deposit pheromones on their paths, guiding subsequent ants towards potentially fruitful areas [7]. Similarly, ACO utilizes virtual "ants" that traverse a network representing the test case space, leaving a virtual pheromone on paths which signifies the path it takes. Paths with more pheromones become more attractive to subsequent ants, guiding them towards potentially high-impact test cases[8]. This iterative process gradually converges on a prioritized list, maximizing fault detection while minimizing execution time.

This study investigated the effectiveness of the ACO algorithm for TCP by evaluating its performance on two diverse datasets sourced from existing research. The performance of the algorithm was assessed based on the fault-detection rate and execution time, demonstrating its potential to improve the efficiency of regression testing. The findings of this study contribute to a deeper understanding of ACO's capabilities of ACO in software testing and encourage further exploration of its applications in the field.

This paper is organized as follows. Section 1 presents the introduction, outlining the background, problem statement, research objectives, and scope of the study. Section 2 reviews the related literature, summarizing previous work and the key concepts relevant to this research. Section 3 describes the research methodology, details the research framework, and processes adopted in the study. Section 4 explains the implementation of the ACO algorithm for the TCP. Section 5 discusses the results and key findings. Finally, the paper concludes with a summary of the study's contributions and provides recommendations for future research.

2. LITERATURE REVIEW

2.1 Regression Testing

In the ever-evolving world of software development, software updates are essential for remaining relevant and meeting evolving customer requirements. However, these updates can mistakenly introduce unintended consequences, potentially causing financial losses or endangering lives. To mitigate these risks, regression testing steps act as safeguards to validate the stability and reliability of the updated software [9]. Regression testing is performed in a meticulous process that involves revisiting previously tested code segments with the following modifications to ensure that no new errors have been inadvertently introduced during the update process [10]. As software evolves through continuous updates, its complexity inevitably increases, posing a significant challenge to exhaustive testing within the constraints of time and resources. To mitigate these challenges, regression testing is strategically categorized into three primary approaches: test suite minimization (TSM), test case selection (TCS), and TCP [9].

Testing large programs can be time consuming, especially when executing redundant test cases. To reduce testing time and costs, it is essential to minimize the number of test cases while maintaining comprehensive coverage. TSM strategies can help achieve this by identifying and removing redundant tests, ultimately streamlining the testing process, and improving its effectiveness [11].

TCS is the process of selecting a specific group of test cases from a larger collection based on predetermined criteria. The primary aim of this approach is to minimize redundant or unnecessary test data while identifying the most effective fault-detecting test cases efficiently [11]. This approach aims to identify smaller test cases that can efficiently identify potential defects with minimal effort and time.

Although TSM and TCS reduce the overall number of test cases, TCP simply rearranges them without removing any test cases. TCP is preferred because it preserves all the test cases for potential future use. Therefore, TCP is a safe alternative for permanent removal. TCP is a secure, reliable, and cost-effective approach for regression testing [3]. The implementation of ACO in TCP aims to enhance fault detection capabilities while simultaneously reducing execution time compared with manual TCP approaches [12].

2.1.1 TCP

TCP is a widely used regression testing technique in software development that aims to optimize the execution sequence of test cases to achieve specific objectives efficiently, such as maximizing the fault detection rate. Coverage-based TCP is a particularly effective and valuable approach among the various TCP techniques. The core principle of coverage-based TCP lies in strategically ordering the test cases for execution. For instance, if the goal is to maximize the fault detection rate, then the test cases with higher priority for the fault detection rate will be executed earlier[13].

The primary evaluation metric used in TCP is APFD. APFD represents the weighted average of the faults detected throughout the execution of prioritized test cases. Its values range from 0 to 1, where higher values indicate a faster and more effective fault detection. A detailed calculation of APFD is presented in Section 3. Aside from APFD, execution time is another critical metric for assessing TCP performance. The execution time measures the duration required for the algorithm to generate a prioritized sequence of test cases based on factors such as fault severity. When optimization algorithms are applied to TCP, the optimal cost, often represented as distance, is determined to identify the most efficient prioritization path. A summary of the algorithms applied to TCP is presented in Table 1.

According to Table 1, several studies have explored the application of metaheuristic algorithms to address the challenges in TCP, particularly in optimizing the execution time and enhancing the fault detection rates. A study conducted by Akila and Arunachalam [12] highlighted the potential of the ACO algorithm to address the challenges. The findings demonstrate that ACO excels in fault detection, efficient path exploration, and optimization of both the number of iterations and test cases required for effective testing. When applied to TCP, ACO aims to achieve two primary objectives: enhancing the fault detection rate and reducing the execution time, particularly when compared with manual approaches in regression testing. In this study, ACO was employed to further investigate its potential to improve the efficiency of test case execution, offering a more streamlined and effective regression-testing process.

2.1.2 Overview of ACO

ACO is a swarm intelligence technique inspired by the foraging behaviour of real ant colonies to solve complex computational problems in the real world. In ACO, artificial ants collaborate to find the best solutions to the problem, whereas artificial ants exchange information about their progress through a communication mechanism used by real ants. This method has been demonstrated to be effective in enhancing test case scenarios across a variety of testing domains, including regression testing and functional testing [8]. Like real ants, these artificial ants leave pheromones behind to indicate the quality of the path they have taken[16]. The higher the pheromone concentration that starts from the first node, the higher the likelihood that other ants will follow that path. This indirect communication and collective decision making allows the colony to move towards the optimal solution[8]. The Ant Colony System (ACS), also known as ACO, which was first proposed by Dorigo et al., is well regarded and demonstrates superior performance compared to other Ant System (AS) strategies owing to its ability to balance exploration and exploitation during the search for optimal solutions[17]. Exploration refers to the process of seeking a new and potentially better solution, whereas exploitation focuses on strengthening and improving known promising solutions, which significantly reduces the cost of executing prioritization.

The ACO algorithm initiates the process by randomly assigning test cases to the artificial ants within a colony. Each ant evaluates the test cases based on their fault detection capabilities and deposit pheromones in proportion to their effectiveness. Pheromone levels are iteratively updated, reinforcing successful paths and reducing the influence of less-effective paths. The process continues until a predefined number of iterations is completed or a satisfactory fault-detection rate is achieved[18]. Through this iterative process, the ant colony collectively identifies the most effective sequence of test cases. Figure 1 illustrates the general ACO framework.

Table 1. Comparison of Algorithms Used in TCP

Author	Algorithm	Purpose	Problem Statement	Results
M. Khatibsyarbini et al. [2]	Firefly Algorithm	Utilize the Firefly Algorithm, coupled with a similarity distance model-based fitness function, for optimal TCP.	Current TCP techniques are inadequate due to the extensive search space involved in identifying the optimal test case sequence regarding execution time and fault detection rate.	The Firefly Algorithm demonstrated superior performance in terms of execution time and APFD. The algorithm emerged as a promising method for TCP applications.
P. Padmnav et al. [14]	Artificial Bee Colony and Genetic Algorithm	Apply Artificial Bee Colony and Genetic Algorithm, guided by historical regression cycle execution data, to improve fault detection effectiveness.	Traditional code coverage-based algorithms have been applied to TCP testing, but their efficacy has been limited due to their reliance on data that is often difficult to obtain.	ABC has led to a remarkable improvement in fault detection performance.
A. Bajaj and O. P. Sangwan [3]	Genetic Algorithm	Analysing the impact of various GA parameter configurations on the performance of TCP through an experimental study.	Default GA parameters provide a solid starting point, but there is potential for further refinement by using different parameter values.	The tournament selection operator emerged as the most effective selection method, with a crossover rate of 1 and a mutation rate of 0.5 yielding the most promising results.
T.K. Akila and A. Malathi [12]	Modified Genetic Algorithm and ACO	Implement a multi-objective strategy that combines the Modified Genetic Algorithm and ACO Algorithm to optimize TCP problem-solving.	Metaheuristic techniques have proven to be effective in addressing TCP. Usually, it will solve a single objective, which is not entirely ineffective, but can adopt a multi-objective approach.	The proposed model surpasses previous techniques which is the single objective model by achieving a significantly higher fault detection rate for multi-objective solutions.
L. Z. Yue and R. Ibrahim [15]	Particle Swarm Optimization and Firefly Algorithm	Examine and compare the performance of PSO and FA algorithms in prioritizing test cases based on execution time, Big-O, and APFD.	Conducting all test cases is expensive and time-consuming which leads to a solution to reduce the time and cost of regression testing by using PSO and FA.	FA outperformed PSO as a TCP technique with faster execution time, lower complexity, and similar APFD values.

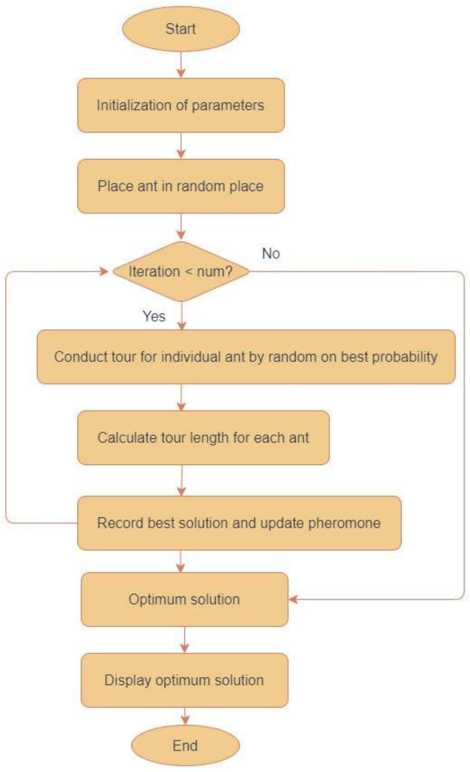

Start
Initialization of parameters
Place ant in random place
Iteration < num?
No
Yes
Conduct tour for individual ant by random on best probability
Calculate tour length for each ant
Record best solution and update pheromone
Optimum solution
Display optimum solution
End

Figure 1. ACO Framework

Based on a study conducted in[19], each ant in the ACO algorithm navigates the solution space based on two key factors: pheromone trails and heuristic information. Pheromone trails, deposited by previous ants, represent collective knowledge of the colony regarding promising paths. The strength of these trials is directly proportional to the quality of the solutions constructed using these paths. Heuristic information quantifies the inherent desirability of traversing a particular edge, often in the form of proximity or cost. The ant's decision-making process employs a probabilistic rule known as the "random-proportional rule," which assigns a probability to each potential next step based on a weighted combination of the pheromone trail and heuristic information. Equation (1) provides the mathematical expression for the next step.

$$P(k)ij = \frac{[\tau ij]^\wedge\alpha * [\eta ij]^\wedge\beta}{\sum u \in J(k)[\tau iu]^\wedge\alpha * [\eta iu]^\wedge\beta} \tag{1}$$

where k is the ant identifier; i and j are the current and next potential places, α and β are parameters controlling the relative influence of pheromone trails and heuristic information, respectively; J(k) is the list of unvisited places for ant k; (τij) is the strength of the trails; and (nij) refers to the heuristic information. Parameter nij is given by Equation (2).

$$nij = \frac{1}{dij} \tag{2}$$

Where, dij is the distance between the ith place and jth place.

Generally, an ant at place i decides on its next move to place j based on two factors: attractiveness and exploration. The more pheromones (τij)and the shorter the distance (nij)between two places, the more attractive the connection becomes. Sometimes, the ant might take a chance and choose a different place, even if it is not the most attractive score. The mathematical equation for the ant's decision-making is given in Equation (3).

$$s = \begin{cases} \text{argmax}\{\tau ij\text{^}\alpha * \eta ij\text{^}\beta\} & \text{if } q \leq q0 \text{ (exploitation)} \\ S & \text{otherwise (biased exploration)} \end{cases} \tag{3}$$

where q is a random number uniformly distributed in the range [0,1], q0 is a fixed parameter ($0 \leq q0 \leq 1$), and S is a random variable selected according to the probability distribution.

The effectiveness of the ACO algorithm depends on the dynamic adjustment of the pheromone levels on the solution paths. Two distinct update mechanisms play a vital role in local and global updating. During the construction of the solutions, individual ants traverse edges, modifying their pheromone levels through a local update rule, as shown in Equation (4).

$$\tau ij(t+1) = (1-\rho)\tau ij(t) + \rho\tau 0 \tag{4}$$

where $\tau ij(t+1)$ represents the pheromone level on edge (i,j) after several iterations $(t+1)$, ρ is the pheromone evaporation factor$(0 < \rho < 1)$, $\tau ij(t)$represents the existing pheromone level on edge (i,j), and $\tau 0$ is the initial pheromone level.

Equation (4) incorporates two key factors, namely evaporation and reinforcement. The mathematical term for $(1-\rho)\tau ij(t)$ represents the natural decay of pheromones over time, preventing stagnation on outdated paths. Conversely, the mathematical term for $\rho\tau 0$ injects a constant amount of pheromone onto each traversed edge, reinforcing its attractiveness to other ants. Following individual ant exploration, a global update mechanism reinforces the most promising path discovered during the current iteration. This selective reinforcement amplifies the influence of the optimal solution on the future ant behaviour. The global update rule, presented in Equation (5), focuses on edges belonging to the "best ant tour."

$$\tau ij(t+1) = (1-\rho)\tau ij(t) + \rho\Delta\tau ij(t+1) \tag{5}$$

where,$\tau ij(t+1)$ represents the updated pheromone level on edge (i,j) after the number of iterations $(t+1)$ and ρ is the pheromone evaporation factor. The mathematical term for $\Delta\tau ij(r)$is defined by Equation (6).

$$\Delta\tau ij(r) = \begin{cases} 1/L; & \text{if}(i,j) \in \text{global} - \text{best} - \text{ant} \\ 0; & \text{otherwise} \end{cases} \tag{6}$$

where L is the total path length of the best ant tour. Through the interplay of local and global updates, the ACO algorithm converges towards optimal solutions. The local updates provide continuous feedback, shaping the pheromone landscape, while the global updates ensure that the best path is effectively communicated and exploited by the ant colony, which focuses on the single best solution identified within each iteration.

3. METHODOLOGY

This section outlines the methodology used to evaluate the effectiveness of the ACO algorithm for addressing the TCP problem. The methodology was structured into several key phases: dataset selection, dataset conversion, algorithm implementation, and performance evaluation.

3.1 Dataset Selection

Tables 2 and 3 present the sample test suites obtained from [20] and [21] for the demonstrations. The first test suite (Case Study One) consisted of 15 test cases and 15 faults. The second test suite (Case Study Two) was obtained from [21] 10 test cases and 10 faults. Case Study One utilized ten ants were assigned under a test suite encompassing 15 test cases and 15 faults detected, as specified in Table 2 in sequential order, represented as T = {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15}. Faults were represented by F = {F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15}. Case Study Two utilized ten ants were assigned under a test suite encompassing ten test cases and ten faults detected, as specified in Table 3 in sequential order, represented as T = {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10}. Faults are represented by F = {F1, F2, F3, F4, F5, F6, F7, F8, F9, F10}.

Table 2. Case Study One

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

Table 3. Case Study Two

	Faults									
Test Cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
T1						/	/			
T2		/			/					
T3										
T4				/					/	
T5										
T6	/				/					
T7			/					/		/
T8		/					/			
T9	/			/				/		
T10			/						/	

This study reveals an interesting connection between dataset complexity and the effectiveness of the ACO algorithm for TCP. A dataset with more test cases and faults, such as Case Study One, may result in a higher average percentage of detected faults (APFD), where faults might be more evenly distributed across the test cases. However, these larger datasets also required more time for the algorithm to converge to the optimal order, leading to longer execution times. Conversely, smaller datasets with fewer test cases and faults, such as Case Study Two, exhibited faster convergence times. Although this led to quicker test execution, it may have resulted in the slight cost of a lower APFD.

3.2 Dataset Conversion to Distance Matrix Using Hamming Distance

The ACO algorithm requires distance matrices as input, a common format in Traveling Salesman Problem (TSP) applications. To adapt the selected datasets to this format, both were transformed into distance matrices. Fault occurrences in the original test case-fault tables were assigned a value of 1, whereas the absence of faults was assigned a value of 0, effectively converting the data into a binary representation. Given this binary nature, Hamming Distance was chosen as the most suitable metric. The Hamming Distance calculates the number of different bit positions between two strings, making it a suitable choice for binary data. Unlike other distance metrics, such as Euclidean or Manhattan distances, which are better suited for continuous data, Hamming Distance consistently yields integer values greater than or equal to 1. This is particularly important because some algorithms may not function well at very small distances [11]. The expression **xi − yi** in Equation (7) is evaluated as zero only when **xi** is equal to **yi**. Otherwise, it evaluates to one.

$$\mathcal{H}_d(x,y) = \sum_{i=1}^{k} |xi - yi| \qquad (7)$$

The resulting distance matrices for Case Study One and Case Study Two are presented in Tables 4 and 5, respectively. These results are used by the ACO algorithm to determine the optimal distance.

Table 4. Distance Matrix of Case Study One

	Faults														
Test Cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**	**F14**	**F15**
T1	0	6	5	7	6	3	7	6	2	9	8	6	8	4	7
T2	6	0	7	7	8	7	5	8	6	7	6	8	6	4	5
T3	5	7	0	4	5	2	6	7	5	6	7	3	7	3	4
T4	7	7	4	0	3	6	4	3	7	2	7	3	3	5	6
T5	6	8	5	3	0	5	5	4	4	5	8	6	4	4	7
T6	3	7	2	6	5	0	6	5	3	8	7	3	7	3	4
T7	7	5	6	4	5	6	0	3	7	2	5	5	1	5	2
T8	6	8	7	3	4	5	3	0	6	3	6	4	2	6	5
T9	2	6	5	7	4	3	7	6	0	9	8	6	8	4	7
T10	9	7	6	2	5	8	2	3	9	0	7	5	1	7	4
T11	8	6	7	7	8	7	5	6	8	7	0	8	6	6	5
T12	6	8	3	3	6	3	5	4	6	5	8	0	6	6	5
T13	8	6	7	3	4	7	1	2	8	1	6	6	0	6	3
T14	4	4	3	5	4	3	5	6	4	7	6	6	6	0	5
T15	7	5	4	6	7	4	2	5	7	4	5	5	3	5	0

3.3 Algorithm Implementation

In the implementation phase, the ACO algorithm, inspired by the TSP principles, was applied to the selected datasets. The experimental procedures were conducted using MATLAB R2023a running on a desktop computer equipped with an Intel Core i5-9500 processor operating at 3.00 GHz.

Figure 2 shows the steps involved in implementing the ACO algorithm for the TCP. In this approach, artificial agents, referred to as ants, explore various sequences in which to execute test cases, analogous to navigating cities in the classical TSP. The primary goal was to determine the most efficient execution order to identify faults early in the testing process. Unlike the physical distances used in TSP, the distances between the test cases represent the estimated effort or cost associated with uncovering and fixing potential faults. This distance is calculated using the Hamming Distance, as defined in Equation (7), where a shorter distance reflects less effort to detect faults.

Table 5. Distance Matrix of Case Study Two

	Faults									
Test Cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
T1	0	5	2	4	2	4	5	2	5	4
T2	5	0	3	5	3	3	4	3	6	5
T3	2	3	0	2	0	2	3	2	3	2
T4	4	5	2	0	2	4	5	4	3	2
T5	2	3	0	2	0	2	3	2	3	2
T6	4	3	2	4	2	0	5	4	3	4
T7	5	4	3	5	3	5	0	5	4	3
T8	2	3	2	4	2	4	5	0	5	4
T9	5	6	3	3	3	3	4	5	0	5
T10	4	5	2	2	2	4	3	4	5	0

```
Initialize distance matrix and parameters
While (max number of iterations is not reached)
    Initialization Back ants numbering and cities
    For each ant
        Initialization Back cities
        Place first ants at random first city
        While (not all cities are visited)
            Select unvisited city by random probability
            Update tour length for each ant
            Update and evaporation of pheromones
        End while
            Record best distance
    End for
End while
Display best distance
```

Figure 2. ACO Pseudocode

The ACO algorithm was run over a fixed number of iterations. During each iteration, ants are initially placed in random test cases, and their paths are built by selecting the next test case based on two main factors: pheromone levels and distances. Pheromone levels reflect historical success, guiding ants towards paths previously associated with early fault detection, whereas the distance factor encourages the selection of test cases that are easier and cheaper to execute. Additionally, the algorithm incorporates a probabilistic exploration mechanism that allows ants to occasionally select less-travelled paths, thereby facilitating the discovery of potentially better solutions. Throughout the iterative process, the best paths identified by the individual ants and the overall best path across all ants were continuously recorded. After completing all the iterations, the algorithm outputs the optimal prioritized test case sequence along with the total execution time.

The experimental parameters used in this study are presented in Table 6. For both the datasets, the algorithm was configured with 150 iterations and 10 ants. Parameter values, such as alpha, beta, and the initial pheromone levels, were carefully selected to balance the influence of distance and pheromone trails on the ants' decision-making process, ensuring effective exploration and exploitation during the search for the optimal solution.

Table 6. ACO Initialization Values

Constant Parameter	Case Study One	Case Study Two
t, Iteration	150	
k, Number of ants	10	
α, Alpha	1	
β Beta	1	
τij, Pheromone Value	0.5	

4. RESULTS AND DISCUSSIONS

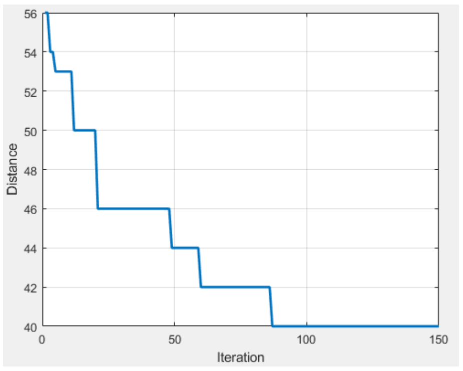
This section evaluates the ACO algorithm for TCP.

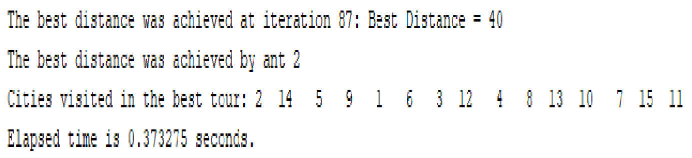
4.1 Prioritized Test Cases

In Case Study One, the ACO algorithm was applied to identify the optimal execution order of the test cases. After completing 150 iterations, the algorithm produced the following optimal test case sequence: T2, T14, T5, T9, T1, T6, T3, T12, T4, T8, T13, T10, T7, T15, and T11. The prioritized orders are summarized in Table 7. The optimization process required 0.3733 s to determine the optimal sequence. As illustrated in Figure 3(a), Case Study One achieved its best optimization distance of 40, with the algorithm converging at iteration 87, indicating the point at which further iterations no longer produced a better solution. A more detailed view of the optimal result is provided in Figure 3(b), highlighting the convergence behavior and efficiency of the ACO algorithm in reaching the optimal solution. Table 7 presents the prioritized test case order. The double slash symbol (//) indicates the earliest instance in which each fault was detected in this order.

Table 7. ACO Prioritized Test Cases for Case Study One

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





```
The best distance was achieved at iteration 87: Best Distance = 40
The best distance was achieved by ant 2
Cities visited in the best tour: 2  14   5   9   1   6   3  12   4   8  13  10   7  15  11
Elapsed time is 0.373275 seconds.
```

(a) (b)

Figure 3. (a) ACO Optimal Solution Graph for Case Study One; (b) ACO Optimal Solution for Case Study One

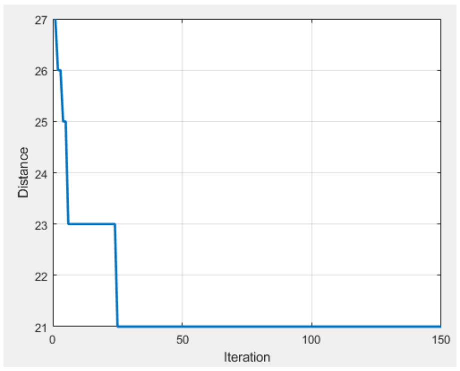
In Case Study Two, after 150 iterations, the algorithm produced the optimal test case sequence T7 – T10 – T4 – T9 – T6 – T2 – T8 – T1 – T3 – T5, as summarized in Table 8. The optimization process was completed in 0.2596 s, reflecting the efficiency of the algorithm when applied to a smaller dataset. As illustrated in Figure 4(a), the best optimization distance achieved was 21, with convergence occurring at iteration 25. This indicates that the algorithm required fewer iterations to reach an optimal solution than Case Study One, likely because of the reduced complexity of the dataset. Further details of the optimization results are presented in Figure 4(b), providing a closer view of the convergence pattern and algorithm performance throughout the iterations. Table 8 lists the prioritized test case order using the ACO algorithm. The double slash symbol (//) indicates the earliest instance in which each fault was detected in this order.

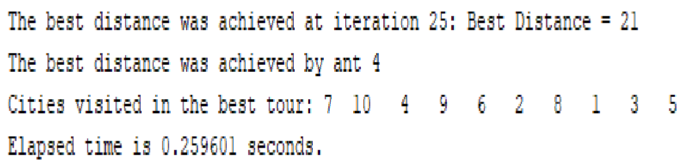
Table 8. ACO Prioritized Test Cases for Case Study Two

	Faults									
Test Cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
T7			//					//		//
T10			/						//	
T4				//					/	
T9	//			/				/		
T6	/				//					
T2		//			/					/
T8		/					//			
T1						//	/			
T3										
T5										

4.2 APFD

APFD considers both the number of faults detected and the order in which they are discovered. A higher APFD, closer to 1, indicates faster fault detection, whereas a lower APFD, that is, a minimum of 0, suggests slower detection. This metric translates into a more efficient testing process when the value is higher. In this study, APFD was calculated for the ACO algorithm in two separate case studies. The calculations were based on prioritized test cases, allowing a comparison of the effectiveness of the ACO algorithm in identifying faults early. Table 9 presents the APFD values for both case studies. Table 9 compares the APFD values of both the case studies. As shown in Figure 7, Case Study One achieved a higher APFD (0.6911) and outperformed Case Study Two (0.6700). This indicates that Case Study One is more efficient in detecting software defects.





```
The best distance was achieved at iteration 25: Best Distance = 21
The best distance was achieved by ant 4
Cities visited in the best tour: 7  10   4   9   6   2   8   1   3   5
Elapsed time is 0.259601 seconds.
```

(a) (b)

Figure 4. (a) ACO Optimal Solution Graph for Case Study Two; (b) ACO Optimal Solution for Case Study Two

Table 9. Comparison of APFD

Case Study One	Case Study Two
0.6911	0.6700

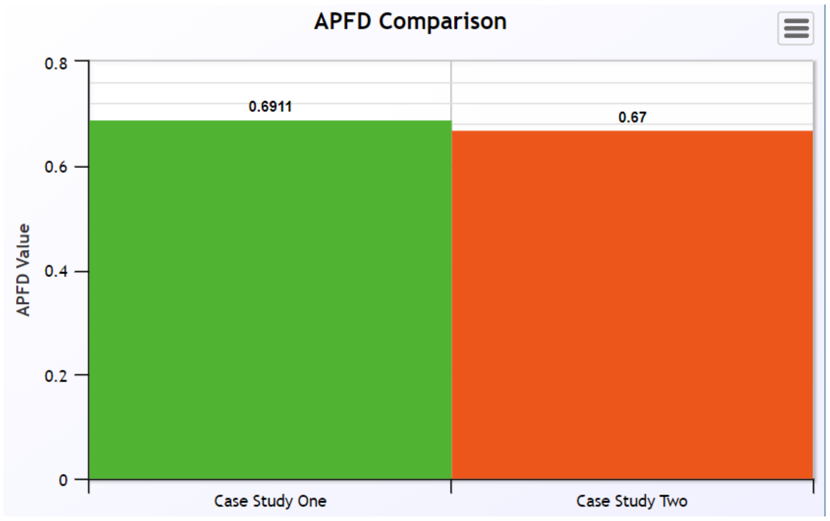


Figure 7. APFD Comparison Graph

4.3 Execution Time

Time is a critical factor in SDLC, as it ensures the timely delivery of the final product. Therefore, during software testing, execution time is a crucial consideration to prevent project delays. The results of implementing this approach are presented in Table 10, which compares the execution times of both case studies. Table 10 presents the execution times for both the case studies. Figure 8 presents a graph comparing the execution times between the case studies, based on Table 10. While neither case study takes more than a second, Case Study One exhibits a slightly longer execution time (0.3733 s) than Case Study Two (0.2596 s). This difference is due to Case Study One having a larger dataset, which makes the optimization process need to be explored further. Case Study Two outperformed Case Study One by 0.1137 s.

Table 10. Comparison of Execution Time

Case Study One	Case Study Two
0.3733	0.2596

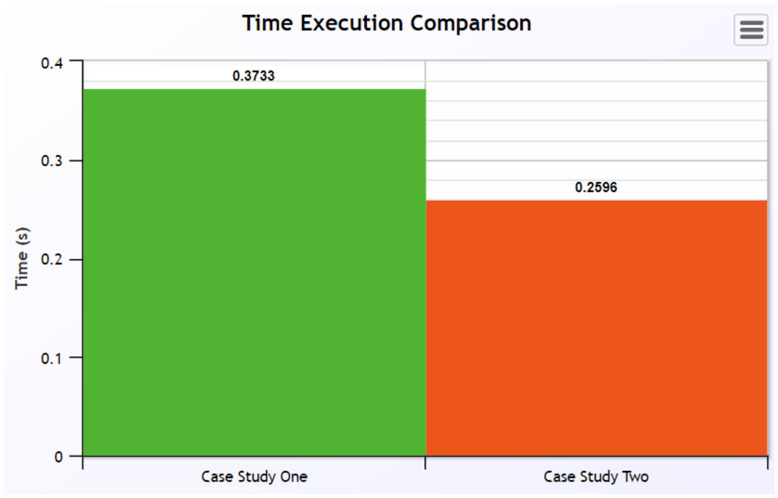


Figure 8. Time Execution Comparison Graph

The analysis of Case Studies One and Two revealed interesting insights into how dataset characteristics influence TCP techniques. Case Study One, with its larger dataset size and potentially higher complexity, achieved higher APFD. This suggests that the TCP technique is more effective in identifying faults within this dataset. However, Case Study Two, with a smaller and potentially less complex dataset, exhibited a significantly faster execution time. The ACO algorithm converged in only 87 iterations compared with the 25 iterations required for Case Study One. This highlights the clear impact of dataset size and complexity on both the algorithm's execution time and APFD. These findings emphasize the importance of considering both APFD and execution time when selecting a TCP technique. The optimal choice may depend on the specific characteristics of the dataset, such as its density (number of test cases and faults).

5. CONCLUSION

This study explored how the ACO algorithm can be used for TCP in software testing. The goal was to compare the ACO effectiveness in terms of how quickly bugs could be found, which was measured using the APFD metric, and the execution time, which was tested on two different datasets: one with 15 test cases and 15 known bugs, referred to as Case Study One, and another with 10 test cases and 10 known bugs, referred to as Case Study Two. This study shows how the algorithms are performed under different conditions, which are the differences between the sizes and defect density. After running the experiments, the analysed results identified the size and defect density of the datasets from the top. It was observed that datasets with more test cases and faults (such as Case Study One) achieved higher APFD, suggesting a better ability to detect faults early in the test run, but took longer to execute (more iterations). Conversely, datasets with fewer test cases (Case Study Two) prioritized faster execution, converging in only 25 iterations compared with the 87 iterations required by Case Study One, but had a slightly lower APFD. This suggests that Case Study Two might be more suitable for scenarios in which faster test execution is critical, even if it comes at a cost in terms of immediate fault identification. As recommended, future studies could investigate the development of hybrid, improved, or enhanced versions of the ACO algorithm to achieve a better balance between the APFD and execution time.

ACKNOWLEDGEMENT

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

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Nurezayana Zainal: Project Administration, Supervision, Validator, Writing – Review & Editing.
Muhammad Sh Salleh: Conceptualization, Methodology, Writing – Original Draft Preparation.
Nur Atiqah Wahidah Sulaiman: Writing – Review & Editing.
Ammar Alazab: Writing – Review & Editing.
Nur Liyana Sulaiman: Project Administration, Writing – Review & Editing.

CONFLICT OF INTERESTS

No conflict of interests were disclosed.

ETHICS STATEMENTS

Our publication ethics follow The Committee of Publication Ethics (COPE) guideline. https://publicationethics.org/

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] V. H. S. Durelli et al., "Machine Learning Applied to Software Testing: A Systematic Mapping Study," *IEEE Trans Reliab*, vol. 68, no. 3, pp. 1189–1212, Sep. 2019, doi: 10.1109/TR.2019.2892517.

[2] 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*, vol. 7, pp. 132360–132373, 2019, doi: 10.1109/ACCESS.2019.2940620.

[3] A. Bajaj, and O. P. Sangwan, "A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms," *IEEE Access*, vol. 7, pp. 126355–126375, 2019, doi: 10.1109/ACCESS.2019.2938260.

[4] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf Softw Technol*, vol. 93, pp. 74–93, Jan. 2018, doi: 10.1016/j.infsof.2017.08.014.

[5] A. Bajaj, A. Abraham, S. Ratnoo, and L. A. Gabralla, "Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization," *Sensors*, vol. 22, no. 12, p. 4374, Jun. 2022, doi: 10.3390/s22124374.

[6] N. M. Minhas, K. Petersen, J. Börstler, and K. Wnuk, "Regression testing for large-scale embedded software development – Exploring the state of practice," *Inf Softw Technol*, vol. 120, p. 106254, Apr. 2020, doi: 10.1016/j.infsof.2019.106254.

[7] P. R. Srivastava, V. Ramachandran, M. Kumar, G. Talukder, V. Tiwari, and P. Sharma, "Generation of test data using meta heuristic approach," in *TENCON 2008 - 2008 IEEE Region 10 Conference, IEEE*, Nov. 2008, pp. 1–6, doi: 10.1109/TENCON.2008.4766707.

[8] S. S. Vinod Chandra, "An Ant Colony Optimization Algorithm Based Automated Generation of Software Test Cases," in *International Conference on Swarm Intelligence* 2020, pp. 231–239, doi: 10.1007/978-3-030-53956-6_21.

[9] A. Bajaj, and O. P. Sangwan, "A Survey on Regression Testing Using Nature-Inspired Approaches," in *2018 4th International Conference on Computing Communication and Automation (ICCCA), IEEE*, Dec. 2018, pp. 1–5, doi: 10.1109/CCAA.2018.8777692.

[10] B. Ba-Quttayyan, H. Mohd, and F. Baharom, "Regression testing – A protocol for systematic literature review," In *AIP Conference Proceedings*, 2018, pp. 020032, doi: 10.1063/1.5055434.

[11] M. A. Asyraf, M. Z. Sahid, and N. Zainal, "Comparative Analysis of Test Case Prioritization Using Ant Colony Optimization Algorithm and Genetic Algorithm," *Journal of Soft Computing and Data Mining*, vol. 4, no. 2, Oct. 2023, doi: 10.30880/jscdm.2023.04.02.005.

[12] T.K. Akila, and A. Malathi, "Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing," *International Journal of Advanced Technology and Engineering Exploration*, vol. 9, no. 88, Mar. 2022, doi: 10.19101/IJATEE.2021.874727.

[13] C. Lu, J. Zhong, Y. Xue, L. Feng, and J. Zhang, "Ant Colony System With Sorting-Based Local Search for Coverage-Based Test Case Prioritization," *IEEE Trans Reliab*, vol. 69, no. 3, pp. 1004–1020, Sep. 2020, doi: 10.1109/TR.2019.2930358.

[14] P. Padmnav, G. Pahwa, D. Singh, and S. Bansal, "Test Case Prioritization based on Historical Failure Patterns using ABC and GA," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), IEEE*, Jan. 2019, pp. 293–298, doi: 10.1109/CONFLUENCE.2019.8776936.

[15] L. Z. Yue, and R. Ibrahim, "Comparative Analysis for Test Case Prioritization Using Particle Swarm Optimization and Firefly Algorithm," *Journal of Soft Computing and Data Mining*, vol. 4, no. 2, Oct. 2023, doi: 10.30880/jscdm.2023.04.02.007.

[16] A. P. Agrawal, and A. Kaur, "A Comprehensive Comparison of Ant Colony and Hybrid Particle Swarm Optimization Algorithms Through Test Case Selection," In *Data Engineering and Intelligent Computing: Proceedings of IC3T 2016*, pp. 397–405, doi: 10.1007/978-981-10-3223-3_38.

[17] M. Z. Zahir Ahmad, R. R. Othman, M. S. A. Rashid Ali, and N. Ramli, "A Self-Adapting Ant Colony Optimization Algorithm Using Fuzzy Logic (ACOF) for Combinatorial Test Suite Generation," *IOP Conf Ser Mater Sci Eng*, vol. 767, no. 1, p. 012017, Feb. 2020, doi: 10.1088/1757-899X/767/1/012017.

[18] K. K. Mohan, and N. Zainal, "Test Case Prioritization Using Swarm Intelligence Algorithm to Improve Fault Detection and Time for Web Application," *Journal of Soft Computing and Data Mining*, vol. 4, no. 2, Oct. 2023, doi: 10.30880/jscdm.2023.04.02.006.

[19] A. Zannou, A. Boulaalam, and E. H. Nfaoui, "Relevant node discovery and selection approach for the Internet of Things based on neural networks and ant colony optimization," *Pervasive Mob Comput*, vol. 70, pp. 101311, Jan. 2021, doi: w10.1016/j.pmcj.2020.101311.

[20] P. Palak, and P. Gulia, "Hybrid swarm and GA based approach for software test case selection," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 6, pp. 4898, Dec. 2019, doi: 10.11591/ijece.v9i6.pp4898-4903.

[21] K. Umanath, and D. Devika, "Optimization of electric discharge machining parameters on titanium alloy (ti-6al-4v) using Taguchi parametric design and genetic algorithm," in *MATEC Web of Conferences*, EDP Sciences, Jun. 2018. doi: 10.1051/matecconf/201817204007.

BIOGRAPHIES OF AUTHORS

	Nurezayana Zainal is a Senior Lecturer at the Faculty of Computer Science and Information Technology from Universiti Tun Hussein Onn Malaysia, Malaysia. She received her PhD in Computer Science from Universiti Teknologi Malaysia (UTM) in 2018 and her Master of Science (Computer Science) from the same university in 2014. Her research interests include soft computing, modeling, optimization, statistical modeling, regression, swarm algorithm and machine learning. She can be contacted at email: nurezayana@uthm.edu.my.
	Muhammad Sh Salleh is a former undergraduate student pursuing a Bachelor of Computer Science (Software Engineering) at the Faculty of Computer Science and Information Technology (FSKTM), Universiti Tun Hussein Onn Malaysia (UTHM). His previous research interests include software testing and software engineering. He can be contacted at email: muhammadshsalleh@gmail.com.
	Nur Atiqah Wahidah binti Sulaiman is a postgraduate student in pursuing in Master of Computer Science (Software Engineering) at the Faculty of Computer Science and Information Technology (FSKTM), Universiti Tun Hussein Onn Malaysia (UTHM). Her research focuses on software testing, optimization technique and metaheuristic algorithms. She can be contacted at email: hi230016@student.uthm.edu.my.
	Ammar Alazab is a Senior Lecturer in Cybersecurity at Torrens University Australia and a member of the Centre for Artificial Intelligence Research and Optimisation (AIRO). He holds a PhD in Computer Science from Deakin University, with expertise spanning AI, cybersecurity, malware detection, federated learning, and blockchain. Author of 60+ publications with 1,400+ citations (h-index 16), he has delivered 50+ talks across academia, industry, and government. He can be contacted at email: ammar.alazab@torrens.edu.au.
	Nur Liyana Sulaiman is a Senior Lecturer at the Faculty of Computer Science and Information Technology from Universiti Tun Hussein Onn Malaysia, Malaysia. She received the BIT degrees from Universiti Utara Malaysia, while Master and PhD degrees in Software Engineering from Universiti Teknologi Malaysia. She was certified by International Software Testing Qualification Board (ISTQB) in Certified Tester Foundation Level (CTFL). She can be contacted at email: nrliyana@uthm.edu.my.