Computer Systems Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.040259

ARTICLE

Security Test Case Prioritization through Ant Colony Optimization Algorithm

Abdulaziz Attaallah[1], Khalil al-Sulbi[2], Areej Alasiry[3], Mehrez Marzougui[3], Mohd Waris Khan[4,*], Mohd Faizan[4], Alka Agrawal[5] and Dhirendra Pandey[5]

[1]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, 21589, Saudi Arabia

[2]Department of Computer Science, Al-Qunfudah Computer College, Umm Al-Qura University, Mecca, Saudi Arabia

[3]College of Computer Science, King Khalid University, Abha, 61421, Saudi Arabia

[4]Department of Computer Application, Integral University, Lucknow, Uttar Pradesh, 226026, India

[5]Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, Uttar Pradesh, 226025, India

*Corresponding Author: Mohd Waris Khan. Email: waris.khan070@gmail.com

Received: 11 March 2023 Accepted: 31 July 2023 Published: 09 November 2023

ABSTRACT

Security testing is a critical concern for organizations worldwide due to the potential financial setbacks and damage to reputation caused by insecure software systems. One of the challenges in software security testing is test case prioritization, which aims to reduce redundancy in fault occurrences when executing test suites. By effectively applying test case prioritization, both the time and cost required for developing secure software can be reduced. This paper proposes a test case prioritization technique based on the Ant Colony Optimization (ACO) algorithm, a metaheuristic approach. The performance of the ACO-based technique is evaluated using the Average Percentage of Fault Detection (APFD) metric, comparing it with traditional techniques. It has been applied to a Mobile Payment Wallet application to validate the proposed approach. The results demonstrate that the proposed technique outperforms the traditional techniques in terms of the APFD metric. The ACO-based technique achieves an APFD of approximately 76%, two percent higher than the second-best optimal ordering technique. These findings suggest that metaheuristic-based prioritization techniques can effectively identify the best test cases, saving time and improving software security overall.

KEYWORDS

Confidentiality; integrity; authentication; non-repudiation; resilience; authorization; Ant Colony Optimization algorithm

1 Introduction

Software-based systems have made their way into every possible space of human society. From mobile phone applications, households, transportation, businesses, banking, and power supply, to space explorations, and other industries, software systems are now dominating modern society [1].

Though software applications have revolutionized human endeavours, it is important to investigate whether these applications are secure and, if so, to what extent. Researchers and developers constantly improvise upon existing development activities to make software systems better and more secure [2].

The software development lifecycle incorporates a major portion where changes are an inevitable part of the development process. Hence, it becomes difficult to manage the security regression test suites [3]. Testers have to run security regression testing whenever the changes are performed in a software application. In cases of multiple applications, or if the entire project needs to be automated, then security regression testing becomes a very complex, time-consuming and costly expense. Optimization techniques are used to overcome the problems of exhaustive testing and save time and cost incurred in the testing process. Optimizing security test cases can highlight security issues as it prioritizes the faults so that the mitigation process can be run sequentially [4,5]. If optimization of security testing is not done, then the testing process involving fault detection also becomes chaotic. To mitigate a fault, the test should quickly find a fault or vulnerability. After identifying the faults, the priority decides which fault/vulnerability should be tackled first [6,7]. This decreases the workload of the developer team as it saves time on reworks, and since the prioritization has already been done, a sequence has to be followed to mitigate the faults.

A thorough literature review in this context reveals that the ACO technique is rarely used to optimise security test cases. Therefore, the present study employs ACO based algorithm for optimizing security test case prioritization. Security regression testing can remove the redundant test cases permanently to reduce the test cases. Test case optimization will give a quicker pass or fail result, thus enhancing the quality of software structure as optimization increases the fault detection rate more accurately [8]. This reduces the computational time and the cost needed to modify the software application.

The ACO algorithm is used to prioritize security test cases concerning the Mobile Wallet Payment application. The result of the ACO-based technique was then compared with the traditional techniques and other state-of-the-art algorithms like Genetic Algorithm (GA) [9], Particle Swarm Optimization (PSO) [10] and Ant Lion Optimization (ALO) [11].

The major contributions of this work are as follows: Firstly, we examined various published standards, and subsequently, we presented a critical review. An ACO-based algorithm for selecting security test case prioritization is proposed. Along with a comparison of the proposed method to other techniques like reverse ordering, random ordering, Genetic Algorithm, Particle Swarm Optimization, Ant Lion Optimization. The fault detection rate is used to evaluate the performance of the algorithm presented in this paper.

Rest of the paper is organized into the following sections: In Section 2, an overview of the relevant work that ACO has done on test cases is presented. Section 3 summarizes various security test case optimization and prioritization techniques along with the complete workflow for selecting test cases, and the ACO-based proposed approach is explained. Section 4 provides us with the implementation of the proposed ACO algorithm on the Mobile Payment wallet. Test case sampling and analysis are presented in Section 5. In Section 6, we discuss the results obtained at each iteration. Section 7 presents the comparison between the traditional algorithm and another evolutionary algorithm. Section 8 shows the discussion, and finally, the conclusion is presented in Section 9.

2 Related Works

Bo et al. elucidated the primary phase of the exploration process in Chimp Optimization Algorithm (ChOA), where the integration of opposition-based learning (OBL) techniques takes precedence to augment ChOA's capability for thorough searches. By incorporating OBL, the traditional ChOA overcomes stagnation issues and achieves a faster convergence rate. In this study, the researchers employ the greedy selection technique to guide ChOA's focus towards the promising search domain within the search space [12].

Liu et al. addressed the ChOA convergence constraint in multimodal optimization problems in this study, and the universal learning paradigm is offered as a solution to this problem. An innovative method for dealing with limitations in real-world optimization issues is presented. The ULChOA has undergone rigorous testing and evaluation across a diverse range of scenarios. Specifically, it has been subjected to assessment on fifteen extensively employed multimodal numerical test functions, ten challenging tests from the IEEE CEC06-2019 suite, and twelve real-world optimization problems characterized by constraints in various engineering domains [13].

Demircioğlu et al. recommended regression testing using network packets. The authors constructed a proof-of-concept test automation tool and assessed it in finance. This technique uses actual data packets in software testing, unlike previous research. The use of network packets contributes to an improvement in the framework's level of generalization. The research finds exceptional reuse capacity and impacts a real-world business-critical system by minimizing effort and enhancing API regression testing automation [14].

Jung et al. proposed a method that explores a collection of products that have s common block of code when covered by the test case with common results followed by the removal of products form the collection on which the test is applied. The assessment findings revealed that the complete selection strategy and the approach of repeatedly using an RTS method for a single software system decreased test executions by 59.3% and 40%, respectively [15].

Lou et al. undertook a rigorous survey to carefully evaluate test case prioritization works from six aspects: algorithms, criteria, measurements, limitations, empirical investigations, and scenarios. The authors described current work and test case prioritization trends for each of the six elements. They also explored the existing challenges and limits in research on test case prioritizing [16].

Taghavi et al. used a unique GWO-based MA and two supplementary features, Individual Best Memory (IBM) and Penalty Factor (PF), to train a Feed-forward Neural Network (FNN) to classify Sonar and Radar datasets. FNN is supplemented by GWO-based Feature Selection (FS). The University of California, Irvin (UCI) sonar and radar datasets were used to test the proposed MA's classification accuracy, local optima avoidance, and convergence speed without increasing computational complexity. This framework works for atmospheric research and naval navigation systems [17].

Bajaj et al. suggested an enhanced quantum-behaved particle swarm optimization technique for regression testing. A fix-up procedure was used for combinatorial TCP perturbation; the dynamic contraction-expansion coefficient speeds convergence. The data revealed that the fault coverage had greater inclusiveness, test selection, and cost reduction percentages than statement coverage, but at the expense of a large fault detection loss (around 7%) during the test case reduction [18].

Saju Sankar et al. [19] aimed to build optimum test cases in an automated manner using a modified ACO technique that provides maximum coverage. The prediction model employed in this study assures that the design of test inputs is more accurate. A comparison of related optimization techniques utilized in automated test case generation was also examined. The viable test cases are produced to cover all transitions at least once.

Zhang et al. [20] introduced a solution based on an ACO algorithm and presented its two distinct implementation processes: distance-based and index-based implementation. Firstly, the test cases have been evaluated with a general indicator based on the requirements. Secondly, the concept of attractiveness in the test case was proposed, and it was based on the definition of the distance between test cases. The main design strategies were finally established, including the pheromone update strategy, optimal update strategy, and a local mutation strategy.

Ning et al. [21] presented a paper on "Comparative study of Ant Colony Algorithms for Multi-Objective Optimization" in which the authors explained the basic ACO process for solving MOPs. A detailed classification of algorithms from different aspects was also presented in the analysis to reflect the various algorithmic characteristics. The study also proposed the selection of MOACOs to solve the problem.

Nayak et al. 's [22] study entitled "Enhancing Efficiency of the Test Case Prioritization Technique by Improving the Rate of Fault Detection" is based on improving the efficacy of the regression test by organizing test cases to meet certain criteria. The paper also suggested one potential criterion, i.e., prioritization, to maximize the fault detection rate in the test suite. Focus was given to arranging test cases in a specific order in which the higher priority test cases were made to operate earlier than the lower test cases.

Sahoo et al. 's [23] study on "Automated Test Case Generation and Optimization" emphasized using an Evolutionary Algorithm to identify test cases with resources. The study also identified the critical domain requirements. Evolutionary Algorithms such as Bee Colony Algorithm (BCA) have gained superiority over other algorithms. This paper also explained the use of the Harmony Search Algorithm (HSA), which is based on the enhancement process of music. Furthermore, the author also explained the use of Particle Swarm Optimization (PSO), the intelligence-based meta-heuristic algorithm.

Hridoy et al. 's [24] study, "Regression Testing Based on Hamming Distance and Code Coverage" introduced the latest method to prioritize test cases, which extends hamming distance-based prioritization with code coverage techniques. The proposed approach helps find past and current bugs in the early regression testing process.

Muthusamy et al. [25] suggested that the method of regression tests be strengthened with priority test cases to coordinate the degree of test case results. The analysis increases the fault rate when the test suites cannot be completed. The authors in this paper have revisited the average Percentage of Faults Detected (APFD) technique. The proposed algorithm proved to be more efficient and beneficial in early fault detection goals.

In the study conducted by Pravin et al. [26], a novel approach of reordering test cases based on time fault was proposed. The underlying premise of this research was to address situations where a fault is detected in the source code, and subsequently, the code is stored in an open-source framework such as Webkit. By reordering the test cases based on the timing of the fault, the researchers aimed to enhance the efficiency of the fault detection and debugging process. This approach recognizes the importance of capturing the temporal aspects of faults and leverages the open-source framework to facilitate effective code management and collaboration among developers.

Table 1 summarizes the relevant research papers on security test case prioritization using the Ant Colony Optimization algorithm. It highlights the key findings and approaches utilized in each study.

Table 1: Existing approaches for test case prioritization

Author's name/year	The focus of the relevant work	Summary of the contribution
Bo et al., 2023 [12]	Developing the chimp optimization algorithm involved integrating a weighted opposition-based technique and a greedy search for effective solutions to complex multimodal engineering problems	This study presents the ChOA algorithm, which combines greedy search (GS) and opposition-based learning (OBL) to enhance exploration and exploitation in solving real-world engineering constrained problems. The algorithm's performance is compared against CMA-ES, SHADE, and cutting-edge methods. Also compared are ordinary ChOA and the best benchmark OBL-based algorithms, such as OBL-GWO, OBL-SSA, and OBL-CSA, which won the CEC competition.
Liu et al., 2022 [13]	Optimizing constraint engineering challenges with robust universal learning chimp optimization shows its ability to solve complexity	The ULChOA, a version of the ChOA, used a revolutionary learning mechanism to update future chimp placements using all prior best knowledge about other chimps. Keeping chimp variation prevented early convergence. Fifteen multimodal functions, 10 IEEE CEC06-2019 suit tests, and 12 limited real-world optimization difficulties rated the ULChOA's performance.
Demircioğlu et al., 2022 [14]	API message-driven regression testing framework	In this work, the technique facilitates test case reuse and provides 100% automation in client-server API regression testing. This automation optimizes testing accuracy, especially for key systems requiring a 100% passing score for release.

(Continued)

Table 1 (continued)

Author's name/year	The focus of the relevant work	Summary of the contribution
Jung et al., 2020 [15]	Efficient regression testing of software product lines with reduced redundant executions	This article proposes a strategy to reduce duplicate SPL regression test runs. This approach avoids irrelevant test runs by choosing products in phases without repeating comparable tests.
Lou et al., 2019 [16]	A survey on regression test case prioritization	The author analyzed the current test case prioritizing techniques and suggested future options. The authors discussed test case prioritization concerns, challenges, and future prospects based on these findings.
Taghavi et al., 2019 [17]	Classification of sonar and radar datasets using a grey wolf optimizer with customizable best memory and penalty factor	This study presents a GWO-enhanced MA for FNN classification training. FSphase reduces time and improves categorization. Current research FS employs MA. IPG outperformed IG, PG, GWO, and AGPSO in experiments. FS improves classification accuracy with fewer features. Finally, the technology may help ships and submarines prevent mishaps, and scientists track ionosphere erosion.
Bajaj et al., 2022 [18]	Enhanced quantum-behaved Particle Swarm Optimization for prioritizing, selecting, and reducing test cases	In this paper, both fault analysis and statement coverage is performed on the algorithm to evaluate its robustness. The suggested technique outperforms the genetic algorithm, bat algorithm, grey wolf optimization, particle swarm optimization and its variants for prioritizing test cases.
Sarkar et al., 2020 [19]	Using an ACO algorithm for automated software test case generation	This paper's prediction model improves test input design. Automated test case creation optimization strategies were compared, which also helps cut down on testing time.

(Continued)

Table 1 (continued)

Author's name/year	The focus of the relevant work	Summary of the contribution
Zhang et al., 2019 [20]	Prioritizing test cases using the ACO algorithm	In this work, the findings of the experiments indicate that the approach has a strong capacity for global optimization and that its overall impact is superior to that of the particle swarm optimization algorithm, the genetic algorithm, and random testing.
Ning et al., 2018 [21]	Multi-objective optimization using Ant Colony Optimization algorithms: A comparative analysis	In this article, the author describes the multi-objective ant method, which was used to find that the number of optimization objectives increases as the complexity of the problem increases. As a result, making use of the operational information to guide the process of multi-objective Ant Colony Optimization will be an efficient method to raise the ant colony's performance even further.
Nayak et al., 2017 [22]	Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection	This study is compared to other ways using APFD values. The suggested approach has a superior fault identification rate than similar prioritization strategies. Analysis of the planned work suggests maximum faults will be discovered in less time.
Sahoo et al., 2016 [23]	Analyzing methods for optimizing and generating test cases automatically	This research analyses the function of Artificial Bee Colony, particle swarm optimization, and harmony search in producing random test data and optimizing it. It discusses how random test cases are produced and how to optimize a problem.

(Continued)

Table 1 (continued)

Author's name/year	The focus of the relevant work	Summary of the contribution
Hridoy et al., 2015 [24]	Utilizing hamming distance and source code coverage for regression testing	This article proposes a novel method for ranking test cases that combines hamming distance and code coverage methodologies. This approach can detect historical problems early and new bugs without changing the priority suite.
Muthusamy et al., 2014 [25]	The efficiency of regression-based prioritization methods for test cases	The authors suggest prioritizing test cases for better regression testing. It improves defect identification when test suites cannot finish. This study revisits the APFD approach. The suggested technique improved early failure detection.
Pravin et al., 2013 [26]	Effective test case selection and prioritization in regression testing	In this paper, the fault detection approach is presented to rank test cases based on their efficiency in finding flaws in the test suite. The fault detection technique decreases test case selection while prioritizing the test suite.

From the above literature survey, it can be seen that the majority of Test Case Prioritization (TCP) approaches have used Evolutionary Algorithm and attempted to enhance a test suite's rate of fault recognition. The study literature review shows a relative study of the old and new techniques of optimization. Moreover, various methods to measure the depth of fault detection during the testing process have also been researched. The approaches discussed in the literature survey show interrelationships among criteria and sub-criteria to identify stronger sets of criteria and sub-criteria so as to focus on their prioritization to deliver better and more accurate results. To identify and uncover faults/bugs at a faster rate, the Average metric Percentage of Faults Detected (APFD) has been utilized.

3 Security Test Case Optimization

The ever-evolving complexities and extensibilities of software applications have had an impending impact on the security structure and corresponding security testing activities. This phenomenon has posed a unique challenge for the developers and researchers who are now devising newer techniques for the security testing of software applications. The issue of recurrent failures during fault identification within the test suites of various modules can be resolved by effectively employing the security testing optimization process. This will help save time and cost in the development process of a software application. Moreover, it will also prevent the rework of a specific development activity [27,28]. While researchers have applied the ACO technique to various sub-systems and domains within software development, its application within the realm of security testing remains untapped. Therefore, after a thorough study, the use of the ACO technique is considered in this proposed article to resolve the problems related to the optimization of security testing of a software application.

3.1 Catalog of Optimized Test Case Prioritization Techniques

The objective of locating faults within security regression testing can be done by prioritizing test cases. Test case prioritization primarily orders test cases and eliminates methods that may not be effective for traditional regression testing. We considered and studied various prioritization techniques, which are as follows [4].

3.1.1 No Ordering (N_o)

The empirical analysis undertaken for this study is based on different prioritization techniques. The first of the methods is simply based on skipping any technique for prioritization, i.e., using test cases as it is without any treatment.

3.1.2 Random Ordering (Rm_o)

In the random ordering technique, we randomly pick a test case order from the test suite for the empirical study.

3.1.3 Reverse Ordering (Re_o)

It refers to the techniques of No ordering or Unordered.

3.1.4 Optimal Ordering (Ot_o)

From a test case, an optimal ordering of the test case can be obtained that may help maximize the fault detection rate of the test suite. Various optimizing algorithms, including evolutionary and bio-inspired algorithms, can be used for this task.

3.2 Ant Colony Optimization (ACO)

ACO was proposed by Dorigo et al. in 1996 [8]. It was developed on the idea of the behaviour of ants. Ants travel long distances in search of food, leaving behind a trail of pheromone deposits in their path. The other ants follow the same path, mainly the path where the pheromone deposit is more than the other paths, thus reinforcing the pheromone deposits in their way. Moreover, ants tend to select the shortest route from the food source to home because the pheromone deposits dissipate after some time. Hence, ants need to decide the shortest possible path [29,30]. Ultimately, other ants which follow the same path also cover the shortest route. ACO method has been used for many optimization problems related to test data generation, multi-objective resource allocation, travelling salesman problems, vehicle routing, etc. [31,32].

However, this method has not been used in the case of security testing of a software application for optimizing the faults during the testing process. The purpose of using ACO based algorithm in this proposed research work is simply to derive the shortest possible approach to cover more and more faults and vulnerabilities in comparatively lesser attempts of test cases. This will help in optimizing the test case suit as it will eliminate the extra time consumed in the traditional testing techniques of software security. Figs. 1 and 2 illustrate the selection process for test cases and the functional diagram for the ACO algorithm.

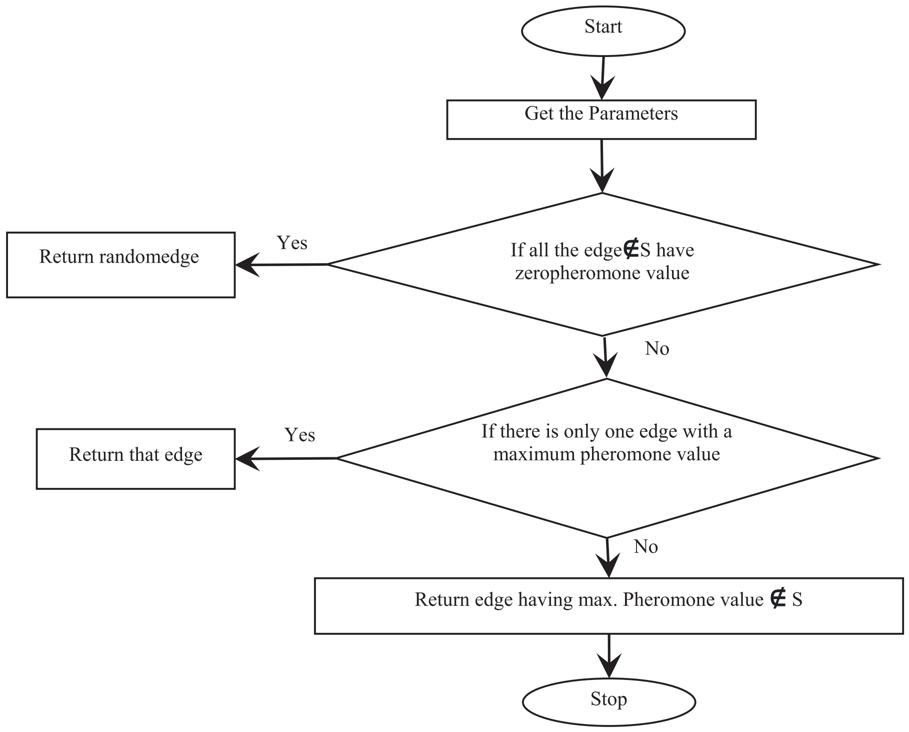
Start
Get the Parameters
If all the edge∉S have zeropheromone value
Yes
Return randomedge
No
If there is only one edge with a maximum pheromone value
Yes
Return that edge
No
Return edge having max. Pheromone value ∉ S
Stop

Figure 1: Flowchart for the selection of test cases

3.2.1 Assumptions for ACO-Based Algorithm

The terms used in ACO-based algorithms for optimizing the security test cases are described in Table 2.

Table 2: Description of ACO-based algorithm

Term	Description
T = {t1, t2, . . . , t49}	Original test suite
CTC = {CTC1, CTC2, . . . , CTC7}	Combined test cases
F = { $F_{lt\text{-}1}$, $F_{lt\text{-}2}$, . . . , $F_{lt\text{-}10}$}	Faults identified
MAX_TIME	The maximum time period after which the execution of the algorithm terminates. (set to 84 min)
An_1 to An_7	Ants (Search agents)
WT_i	Weight of the *i*th edge
K	Pheromone evaporation rate (set to 10%)

Each of the CTCs may cover some or all the faults from F. The minimum execution time of each ant is added to the current execution time after every iteration. The algorithm terminates once the current execution time exceeds MAX_TIME. The weight WT_i of an edge represents the amount of pheromone deposited on the edge. The rate of pheromone deposition is assumed to be 100% for each ant. The pheromone evaporation rate is assumed to be k%, which means that after the deposition of every iteration, the pheromone is reduced by k% of WT_i.

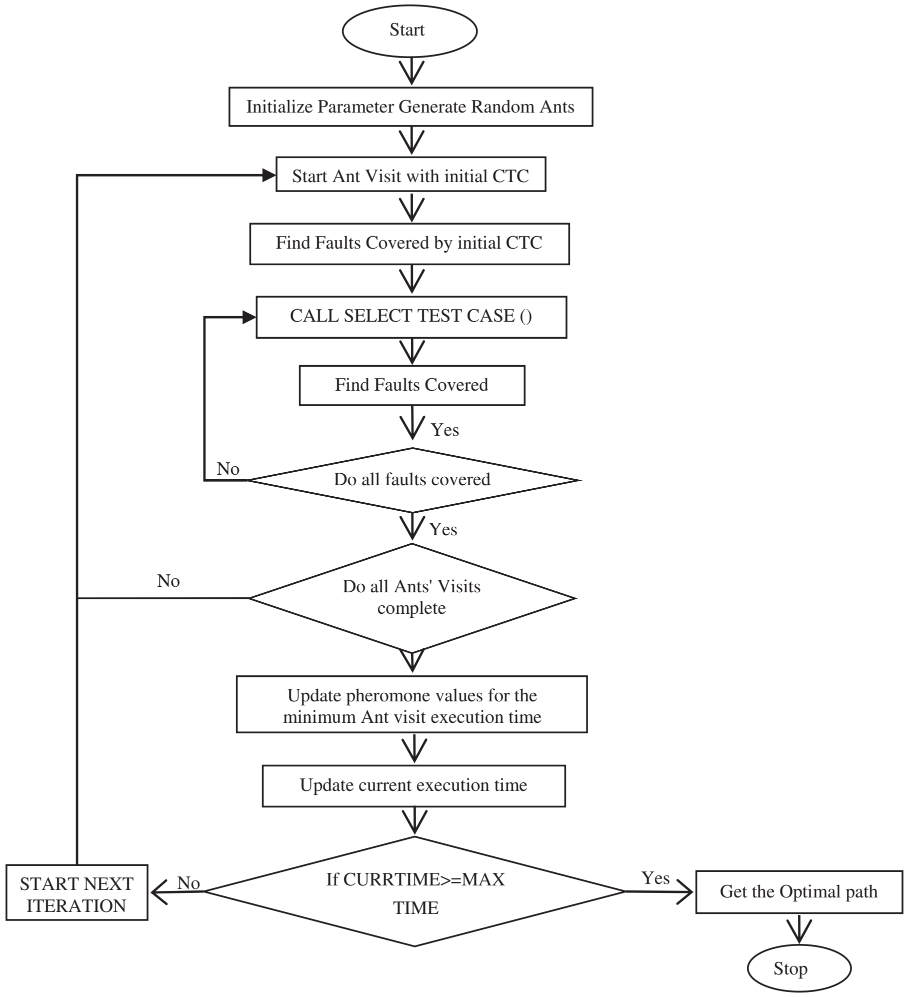
Start
Initialize Parameter Generate Random Ants
Start Ant Visit with initial CTC
Find Faults Covered by initial CTC
CALL SELECT TEST CASE ()
Find Faults Covered
Yes
No
Do all faults covered
Yes
No
Do all Ants' Visits complete
Update pheromone values for the minimum Ant visit execution time
Update current execution time
START NEXT ITERATION
No
If CURRTIME>=MAX TIME
Yes
Get the Optimal path
Stop

Figure 2: Flowchart of ACO-based proposed approach

3.2.2 *Selection of Test Cases*

The flowchart in Fig. 1 shows the process of selecting the test cases. Initially, when all the edges have zero pheromones deposited, a random edge is selected. Otherwise, an edge with a maximum pheromone value is selected. If there are multiple edges with maximum pheromone value, then an edge is selected which was previously not selected.

3.2.3 *Functional Diagram for the Proposed ACO Algorithm*

The proposed ACO-based algorithm, as shown in Fig. 2, is implemented step by step as follows:

Step 1: Initialize parameters and create an ant colony.

Step 2: Position an ant at the initial CTC.

Step 3: Identify faults explored by the first CTC.

Step 4: Choose a test case from SELECT_TEST_CASE and document covered faults.

Step 5: Repeat step 4 until each fault is explored.

Step 6: Deploy the next ant when all faults are covered, repeating until each ant completes a cycle.

Step 7: Adjust pheromone levels for the ant with the shortest run time.

Step 8: Keep track of the current execution time.

Step 9: If the MAX_TIME is greater than the present run time, repeat steps 2 to 8 for the next iteration.

Step 10: If the current run time surpass the MAX_TIME, obtain the optimal path and end the algorithm.

4 Case Study: Mobile Payment Wallet

Mobile payment wallets are now widespread, utilizing various communication technologies like Magnetic Secure Transmission (MST), Near Field Communication (NFC), Quick Recognition Code (QR), Bluetooth Low Energy (BLE), SMS, and the Internet to transmit payment data to merchant Point of Sale (POS) systems. The Mobile Payment Wallet case study is a mobile app comprising seven modules: Sign In (M1), Profile Update (M2), Bill Payments & Recharges, Add money, mPIN Change, Funds Transfer, and Transaction Details as shown in Fig. 3.

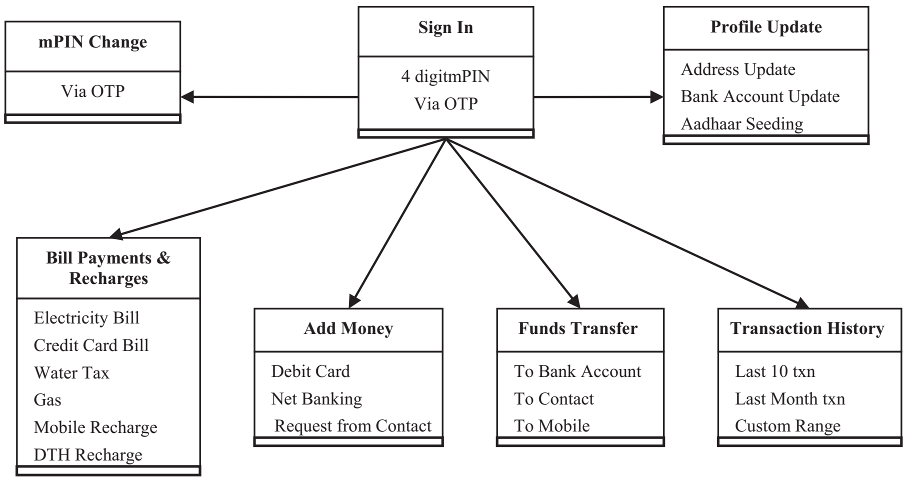

Figure 3: Diagrammatic illustration of the wallet application for mobile payments

In the Sign In module (M1), the users are required to input their login credentials, specifically consisting of a unique username and a confidential four-digit mPIN. These credentials serve as the key to access the application. In the subsequent module, Profile Update (M2), authenticated users possess the privilege to modify their account details. This includes updating personal information like addresses, bank accounts, and integrating Aadhaar details, among others. Notably, this information holds significant importance and is restricted from alteration or access by any unauthorized users, ensuring the security and integrity of the user's account.

In the third module (M3), the users can spend the added money from their online wallet to settle utility bills of credit card, electricity, house tax, etc. In case of a low wallet balance, the user can add money to the wallet through the fourth module (M4). This can be done either through the use of a debit/credit card or by net banking, or by request from a contact. The way in which these payments are processed is not altered through mobile payments by card merchant networks; they are based on traditional card-based payment systems in general.

In the fifth module (M5), the user can change the four-digit mPIN. The request for change is validated via an OTP (One Time Password) sent to the registered mobile number of the user, which will authenticate the transaction. In the sixth module (M6), the funds can be transferred to a specific bank account, to a contact or to a mobile number attached to the user's profile. Module seven (M7) furnishes comprehensive information pertaining to online transactions. This encompassing data encompasses details of the last ten transactions, an overview of transactions from the previous month, or the specifics of transactions for a user-defined period, as explicitly specified by the user.

5 Test Case Sampling and Analysis

Software security test cases encompass a collection of scenarios, as illustrated in Fig. 4, comprising both typical use cases and alternate cases. These meticulously constructed test cases serve the purpose of investigating potential system faults, conducting thorough analyses of various security threats and breaches that a software system may encounter, evaluating the reliability and dependability of the software in diverse environments, and assessing the system's resilience to faults.

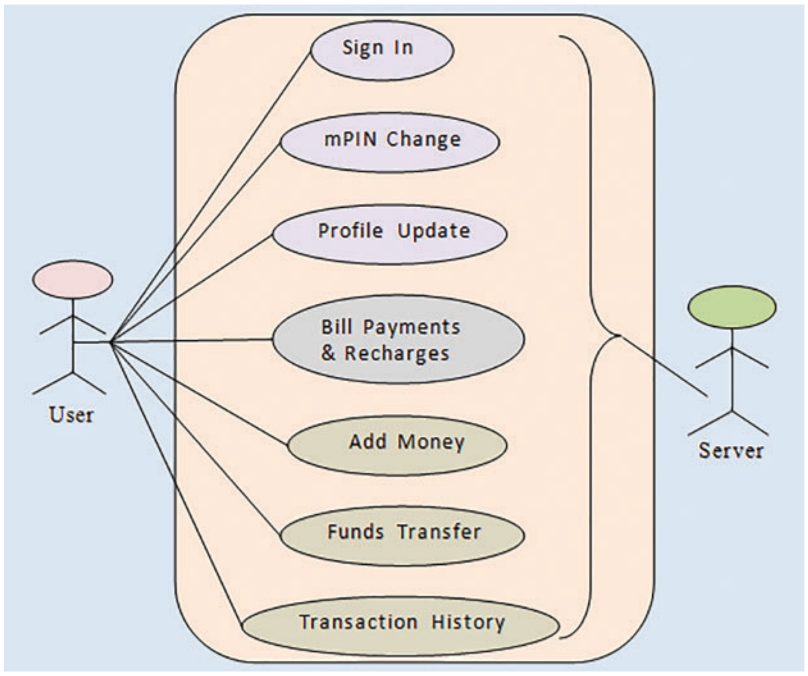

Figure 4: A diagram depicting the use cases within the mobile payment wallet system

In order to comprehensively assess the behaviour of software systems under alternate test case scenarios, the tester must meticulously examine and account for all potential vulnerabilities that pose security challenges to the software system. This rigorous analysis aims to identify any weak links or vulnerabilities within the software's structure.

To analyse the behaviour of the software systems in alternate paradigm, the tester should take all the enervations that compromise the system security. For the present study, seven test cases have been developed for each of the seven modules with respect to security attributes. Consequently, a comprehensive set of 49 test cases was developed. Table 3 presents a catalog of 10 potential faults that could manifest during the application's utilization, specifically concerning the seven vital security attributes: authentication, authorization, confidentiality, availability, integrity, non-repudiation, and resilience. The normal flow of use cases has been shown in Fig. 4, and the alternate flow of use cases is explained in their respective modules.

Table 3: Description of the faults

Faults	Description
$F_{lt\text{-}1}$	RNG (Random Number Generation) failure
$F_{lt\text{-}2}$	Network error
$F_{lt\text{-}3}$	Segmentation fault
$F_{lt\text{-}4}$	File/data not found
$F_{lt\text{-}5}$	Data loss
$F_{lt\text{-}6}$	Infinite loop
$F_{lt\text{-}7}$	Communication error
$F_{lt\text{-}8}$	Configuration error
$F_{lt\text{-}9}$	Service fault
$F_{lt\text{-}10}$	Coupling fault

5.1 Module-1: Sign In

This module serves as the user's gateway for accessing the system. It conducts user authentication by verifying login credentials, specifically the username and mPIN. In the event of incorrect credentials, the module triggers an error notification. Additionally, this module provides a platform for studying the alternate use case scenario of SignIn, encompassing test cases 1 to 7. In test case-1, if the system attempts to retrieve data from an incorrect memory segment, it results in the generation of $F_{lt\text{-}3}$. Test case-2 presents multiple potential scenarios. Firstly, an incorrect mPIN entry or a mismatch with the associated file triggers $F_{lt\text{-}2}$ and $F_{lt\text{-}4}$. Secondly, poor connectivity conditions can lead to $F_{lt\text{-}7}$, while unavailability of the service prompts $F_{lt\text{-}9}$. During the initial application startup phase, any erroneous user-entered parameters result in the detection of a configuration error, specifically flagged as $F_{lt\text{-}8}$ in test case-3. In test case-4, the occurrence of network interruptions or failures in file retrieval can potentially result in multiple faults, including $F_{lt\text{-}2}$, $F_{lt\text{-}4}$, and $F_{lt\text{-}9}$. Subsequently, following a successful application startup, test case-5 focuses on the transition of information to the next module. Any breakdown in this transfer process may lead to the emergence of $F_{lt\text{-}10}$. Moving on to test case-6, the inability of a user to authenticate their digital signature with the server gives rise to $F_{lt\text{-}1}$, causing a communication gap and consequently triggering $F_{lt\text{-}7}$. Finally, test case-7 encompasses two potential scenarios. Firstly, the application may encounter startup issues denoted by $F_{lt\text{-}8}$, or it may experience a halt marked by $F_{lt\text{-}6}$. Table 4 provides an overview of the specific faults covered by each test case.

Table 4: Sample test cases for each security attribute and associated faults identified in module-1

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_1 (Authorization)	_	_	X	_	_	_	_	_	_	_
TC_2 (Authentication)	_	X	_	X	_	_	X	_	X	_
TC_3 (Confidentiality)	_	_	_	_	_	_	_	X	_	_
TC_4 (Availability)	_	X	_	X	_	_	_	_	X	_
TC_5 (Integrity)	_	_	_	_	_	_	_	_	_	X
TC_6 (Non-repudiation)	X	_	_	_	_	_	X	_	_	_
TC_7 (Resilience)	_	_	_	_	_	_	_	X	_	_

5.2 Module-2: mPIN Change

This module is used by the users for changing their 4-digit mPIN in case of a suspected leak of mPIN or upon the expiry of the validity period. The user will initiate this request to the application server. The application server will authenticate it via OTP and old mPIN. Another scenario of mPIN change includes test cases 8 to 14 can be studied in this module (Table 5).

Table 5: Sample test cases for each security attribute and associated faults identified in module-2

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_8	X	_	_	_	_	_	_	_	_	_
TC_9	_	X	_	_	_	_	_	_	X	_
TC_10	_	_	_	_	_	_	_	_	_	_
TC_11	_	X	_	X	_	_	_	_	X	_
TC_12	_	_	_	_	_	X	_	_	_	_
TC_13	X	_	_	_	_	_	X	_	_	_
TC_14	_	_	_	_	_	X	_	_	_	_

In test case-8, the authorization may be affected by a corrupt digital signature because of F_{lt-1}. In test case-9, the authentication may fail due to F_{lt-2}, which consequently leads to F_{lt-9}. In this module, no fault was observed, which can affect test case-10. For test case-11, a break in the network or in file read could lead to F_{lt-2}, F_{lt-4} and F_{lt-9}. As in test case-7, the system may halt due to F_{lt-6} in test case-12. Test case-13 is similar to test case-6. Finally, in this module, the request may fail due to F_{lt-6} in test case-14. Table 5 shows the faults covered by each test case.

5.3 Module-3: Profile Update

This module allows users to update personal details and linked bank account details. It allows users to update KYC details like Aadhaar linking. This module facilitates the examination of an alternative use case scenario specifically focused on profile updates. This scenario includes 15–21 test scenarios that give significant insights and analyses. In test case-15, an authorization may fail due to a mismatch of the digital signatures because of F_{lt-1}. In test case-16, a user may not be able to complete the transactions because of F_{lt-2} leading to F_{lt-9}. For test case-17, due to F_{lt-7}, the user session may expire after no response from the user's side. F_{lt-2} and F_{lt-4} will affect the availability of the system in test case-18. Aadhaar linking could be failed to F_{lt-6}, or there may be some issues in linking with another module due to F_{lt-10} in test case-19. Test case-20 consists of F_{lt-7} due to no response from the user's side. Finally, at the last stage of this module, when the system reads external data, the user segment F_{lt-3} will occur in test case-21. Table 6 shows the faults coverage of the test cases.

Table 6: Sample test cases for each security attribute and associated faults identified in module-3

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_15	X	_	_	_	_	_	_	_	_	_
TC_16	_	X	_	_	_	_	_	_	X	_
TC_17	_	_	_	_	_	_	X	_	_	_
TC_18	_	X	_	X	_	_	_	_	_	_
TC_19	_	_	_	_	_	X	_	_	_	X
TC_20	_	_	_	_	_	_	X	_	_	_
TC_21	_	_	X	_	_	_	_	_	_	_

5.4 Module-4: Bill Payments & Recharges

This module is used for the payment of online bills like water tax, credit card bills, electricity bills, etc. The same module also allows users to pay bills for online shopping and prepaid/postpaid recharges. The users can pay these bills either by debit card and CVV or Internet Banking or via wallet balance. The user is required to enter the correct mPINin order to perform the transaction. The transaction will be declined if the user enters the wrong mPIN. The alternate use case scenario of bill payments & recharges that includes test cases-22 to 28 can be studied in this module (Table 7).

Table 7: Sample test cases for each security attribute and associated faults identified in module-4

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_22	X	_	_	_	_	_	_	_	_	_
TC_23	_	X	_	_	_	_	_	_	X	_
TC_24	_	_	_	_	X	_	_	_	_	_
TC_25	_	X	_	X	_	_	_	_	X	_
TC_26	_	_	X	_	_	_	_	_	_	X
TC_27	_	_	_	_	_	_	X	_	_	_
TC_28	_	_	_	_	_	_	_	_	_	_

Test case-22 recognized that F_{lt-1} might hinder the authorization if a unique key is not generated. In test case-23, poor network connectivity will result in F_{lt-2} and F_{lt-9}. In test case-24, a loophole in the system may result in F_{lt-5}. F_{lt-2}, F_{lt-4} and F_{lt-9} will occur in test case-25 if the user is denied service. In test case-26, F_{lt-3} and F_{lt-10} are raised due to a mismatch in file read and information passing between two modules, respectively. In test case-27, the transaction gets cancelled if the user cannot verify himself due to F_{lt-7}. No fault will occur for test case-28. Table 7 shows the faults coverage of the test cases.

5.5 Module-5: Add Money

This module is used to top up or add money to the wallet. The user may either add money via debit card or Internet Banking, or the user may also request money from a contact. The alternate use case scenario of adding money that includes test cases-29 to 35 can be studied in this module. All the test cases from test case-29 to 35 are similar to test case-22 to test case-28 in Module 4, respectively. Table 8 shows the faults covered by each test case.

Table 8: Sample test cases for each security attribute and associated faults identified in module-5

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_29	X	_	_	_	_	_	_	_	_	_
TC_30	_	X	_	_	_	_	_	_	X	_
TC_31	_	_	_	_	X	_	_	_	_	_
TC_32	_	X	_	X	_	_	_	_	X	_
TC_33	_	_	X	_	_	_	_	_	_	X
TC_34	_	_	_	_	_	_	X	_	_	_
TC_35	_	_	_	_	_	_	_	_	_	_

5.6 Module-6: Funds Transfer

This module is used to initiate funds transfers to bank accounts, contacts or mobile. The user will enter the amount, beneficiary detail and mPIN to complete the transaction. The alternate use case scenario of funds transfer that includes test cases-36 to 42 can be studied in this module. In test case-36, an incorrect key due to F_{lt-1} will halt the authorization process. In test case-37, F_{lt-2} and F_{lt-9} will occur because of poor connectivity. In test case-38, the beneficiary may not be verified due to F_{lt-1}, and in test case-39, the beneficiary details cannot be retrieved due to F_{lt-2}, F_{lt-4} or F_{lt-9}. In test case-40, bad file read, or miscommunication between modules will happen due to F_{lt-3} and F_{lt-10}, respectively. F_{lt-1} will occur as a result of an incorrect account number, and F_{lt-7} may occur due to loss of information during transfer in test case-41. Finally, in test case-42, no-fault will occur. Table 9 shows the faults coverage of the test cases.

Table 9: Sample test cases for each security attribute and associated faults identified in module-6

Security attribute/fault-specific test cases	F_{lt-1}	F_{lt-2}	F_{lt-3}	F_{lt-4}	F_{lt-5}	F_{lt-6}	F_{lt-7}	F_{lt-8}	F_{lt-9}	F_{lt-10}
TC_36	X	_	_	_	_	_	_	_	_	_
TC_37	_	X	_	_	_	_	_	_	X	_
TC_38	X	_	_	_	_	_	_	_	_	_
TC_39	_	X	_	X	_	_	_	_	X	_
TC_40	_	_	X	_	_	_	_	_	_	X
TC_41	X	_	_	_	_	_	X	_	_	_
TC_42	_	_	_	_	_	_	_	_	_	_

5.7 Module-7: Transaction History

This module is used to get the transaction details that are completed through the wallet. The alternate use case scenario of transaction history that includes test cases-43 to 49 can be studied in this module. In test case-43, $F_{lt\text{-}3}$ may occur due to improper reading. In test case-44, the poor or congested network connection may cause $F_{lt\text{-}2}$ and $F_{lt\text{-}9}$. In test case-45, $F_{lt\text{-}7}$ may occur due to connection loss. The transaction detail may not be found because of bad connectivity, thus leading to $F_{lt\text{-}2}$ and $F_{lt\text{-}9}$. Consequently, $F_{lt\text{-}4}$ may also occur in test case-46. If the user has specified an invalid range, then $F_{lt\text{-}6}$ may occur in test case-47. In test case-48, again, the network failure may result in $F_{lt\text{-}7}$. The $F_{lt\text{-}3}$ may occur during test case-49 because of miscommunication between modules. Table 10 shows the faults coverage of the test cases.

Table 10: Sample test cases for each security attribute and associated faults identified in module-7

Security attribute/fault-specific test cases	$F_{lt\text{-}1}$	$F_{lt\text{-}2}$	$F_{lt\text{-}3}$	$F_{lt\text{-}4}$	$F_{lt\text{-}5}$	$F_{lt\text{-}6}$	$F_{lt\text{-}7}$	$F_{lt\text{-}8}$	$F_{lt\text{-}9}$	$F_{lt\text{-}10}$
TC_43	_	_	X	_	_	_	_	_	_	_
TC_44	_	X	_	_	_	_	_	_	X	_
TC_45	_	_	_	_	_	_	X	_	_	_
TC_46	_	X	_	X	_	_	_	_	X	_
TC_47	_	_	_	_	_	X	_	_	_	_
TC_48	_	_	_	_	_	_	X	_	_	_
TC_49	_	_	X	_	_	_	_	_	_	_

The outcomes of all the test cases are checked for any redundancy that may have crept in the final result. The combined set of sample test cases is designed with respect to each security attribute, as shown in Table 11. These test cases aim to address the identified faults while considering the execution time associated with each test case.

Table 11: Combined test cases addressing security attributes, identified faults, and runtime

S. No.	Combined test cases	Total number of faults covered during security test	Respective module in which faults are found	Total execution time per unit
1	CTC1 (Authorization)	$F_{lt\text{-}1}$, $F_{lt\text{-}3}$	Mi,i = 1, 2, . . . ,7	7
2	CTC2 (Authentication)	$F_{lt\text{-}2}$, $F_{lt\text{-}4}$, $F_{lt\text{-}7}$, $F_{lt\text{-}9}$	Mi,i = 1, 2, . . . , 7	4
3	CTC3 (Confidentiality)	$F_{lt\text{-}1}$, $F_{lt\text{-}5}$, $F_{lt\text{-}7}$, $F_{lt\text{-}8}$	Mi,i = 1, 3, 4, 5, 6, 7	5
4	CTC4 (Availability)	$F_{lt\text{-}2}$, $F_{lt\text{-}4}$, $F_{lt\text{-}9}$	Mi,i = 1, 2, . . . , 7	4
5	CTC5 (Integrity)	$F_{lt\text{-}3}$, $F_{lt\text{-}6}$, $F_{lt\text{-}10}$	Mi,i = 1, 2, . . . , 7	4
6	CTC6 (Non-repudiation)	$F_{lt\text{-}1}$, $F_{lt\text{-}7}$	Mi,i = 1, 2, . . . , 7	5
7	CTC7 (Resilience)	$F_{lt\text{-}3}$, $F_{lt\text{-}6}$, $F_{lt\text{-}8}$	Mi,i = 1, 2, 3, 7	4

Where “CTC” refers to Combined test cases, “F_{lt}” signifies Faults, and M1, M2, . . . , M7 represent individual modules.

The organization’s paramount objective is to ascertain the optimal prioritization sequence for achieving comprehensive fault coverage. To address the challenges that have arisen during the optimization of test cases, we have initiated the application of the ACO technique. Previous studies done in this context have cited that early termination of testing could result in effective fault detection with reordered test suite [3,33]. In the current section, the execution of the algorithm with time-based prioritization is presented.

The security regression test suite, referred to as CTC, comprises a total of seven combined test cases, denoted as {CTC1, CTC2, CTC3, CTC4, CTC5, CTC6, CTC7}, each associated with specific execution times detailed in Table 9. During the execution of these combined test cases, the following fault-detection outcomes were observed: CTC1 identified two faults, namely {$F_{lt\text{-}1}$, $F_{lt\text{-}3}$}, within a time frame of seven minutes. CTC2 detected four faults, specifically {$F_{lt\text{-}2}$, $F_{lt\text{-}4}$, $F_{lt\text{-}7}$, $F_{lt\text{-}9}$}, in just four minutes. CTC3 revealed four faults, denoted as {$F_{lt\text{-}1}$, $F_{lt\text{-}5}$, $F_{lt\text{-}7}$, $F_{lt\text{-}8}$}, within a five-minute duration. Both CTC4 and CTC5 uncovered three faults each within four minutes, identifying {$F_{lt\text{-}2}$, $F_{lt\text{-}4}$, $F_{lt\text{-}9}$} and {$F_{lt\text{-}3}$, $F_{lt\text{-}6}$, $F_{lt\text{-}10}$}, respectively. CTC6 successfully pinpointed two faults, {$F_{lt\text{-}1}$, $F_{lt\text{-}7}$}, in five minutes. Lastly, CTC7 identified three faults, namely {$F_{lt\text{-}3}$, $F_{lt\text{-}6}$, $F_{lt\text{-}8}$}, in just four minutes. These findings provide valuable insights into the fault-detection capabilities and execution times associated with each of the combined test cases within the CTC test suite.

The combined test suite is derived by consolidating all seven test case scenarios while eliminating redundancy. The challenge at hand pertains to the selection and prioritization of security test cases for optimal fault coverage. This selection process revolves around maximizing the fault-detection capability of the test cases drawn from the set ‘T’, which represents the entire Test Suite comprising ‘n’ combined test cases. The objective is to identify a subset ‘S’ containing ‘m’ test cases (where ‘m’ is less than ‘n’) that are carefully chosen and optimized to achieve the desired testing objectives. The problem can be depicted as an undirected graph G (V, E), where V refers to the set of vertices, and E refers to the set of edges in the graph. ‘WTi’ is the weight of ‘ith’ edge. It describes the pheromone trail deposited on edge ei ϵ E, which indicates the fault coverage ‘Flt_i’ on the selected path under the time limit, ‘MAX_TIME’. The edges are initialized with the zero value.

6 Results

The sequence of the ACO-based algorithm’s execution is demonstrated across iterations in Tables 12 to 19. To ensure controlled termination, a maximum time limit of 84 min is set, beyond which the algorithm concludes the iteration and terminates. During each of the four iterations, every ant selects a random path initially. As soon as the fault coverage criteria are fulfilled along the chosen path, the ant halts its search. Ultimately, all the ants aim to identify the most favorable path that allows them to cover the distance within the minimum total time. This strategic approach aids in attaining the optimal path that effectively covers all the faults. The output of the ACO-based algorithm in the initial pass is presented in Table 12, displaying the faults covered by each ant and the time taken for the coverage. Additionally, Table 13 shows the adjacency matrix of the graph representing the path covered by each ant. Moving to the second iteration, Table 14 exhibits the output of the ACO-based algorithm, while the corresponding adjacency matrix is shown in Table 15. Similarly, for the third iteration, Table 16 showcases the algorithm’s output, and the corresponding adjacency matrix is presented in Table 17. Finally, in the fourth iteration, Table 18 displays the output of the ACO-based algorithm, and Table 19 shows the corresponding adjacency matrix.

Table 12: Execution of algorithm (1st iteration)

ACO			An.1			An.2			An.3			An.4			An.5			An.6			An.7		
First	Iteration	Observation	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$
			CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC5	4	3, 6, 10	CTC6	5	1, 7	CTC7	4	3, 6, 8
			CTC5	4	3, 6, 10	CTC6	5	1, 7	CTC4	4	2, 4, 9	CTC1	7	1, 3	CTC7	4	3, 6, 8	CTC1	7	1, 3	CTC2	4	2, 4, 7, 9
			CTC2	4	2, 4, 7, 9	CTC4	4	2, 4, 9	CTC5	4	3, 6, 10	CTC3	5	1, 5, 7, 8	CTC1	7	1, 3	CTC5	4	3, 6, 10	CTC6	5	1, 7
			CTC3	5	1, 5, 7, 8	CTC1	7	1, 3				CTC2	4	2, 4, 7, 9	CTC6	5	1, 7	CTC3	5	1, 5, 7, 8	CTC1	7	1, 3
						CTC5	4	3, 6, 10				CTC7	4	3, 6, 8	CTC3	5	1, 5, 7, 8	CTC2	4	2, 4, 7, 9	CTC5	4	3, 6, 10
						CTC7	4	3, 6, 8				CTC6	5	1, 7	CTC2	4	2, 4, 7, 9				CTC4	4	2, 4, 9
						CTC3	5	1, 5, 7, 8				CTC5	4	3, 6, 10							CTC3	5	1, 5, 7, 8
		Total execution time (units)		20			33			13			33			29			25			33	

Note: Where PC: Path Covered, ET: Execution Time, Flt.C: Faults Covered.

Table 13: Adjacency matrix of pheromone deposited after first iteration

	An.1	An.2	An.3	An.4	An.5	An.6	An.7
An.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.3	0.0	0.0	0.0	0.9	0.0	0.0	0.0
An.4	0.0	0.0	0.0	0.0	0.9	0.0	0.0
An.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 14: Execution of algorithm (2nd iteration)

ACO			An.1			An.2			An.3			An.4			An.5			An.6			An.7		
Second	Iteration	Observation	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$
			CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC5	4	3, 6, 10	CTC6	5	1, 7	CTC7	4	3, 6, 8
			CTC3	5	1, 5, 7, 8	CTC3	5	1, 5, 7, 8	CTC5	4	3, 6, 10	CTC5	4	3, 6, 10	CTC3	5	1, 5, 7, 8	CTC3	5	1, 5, 7, 8	CTC6	5	1, 7
			CTC5	4	3, 6, 10	CTC5	4	3, 6, 10	CTC1	7	1, 3	CTC1	7	1, 3	CTC7	4	3, 6, 8	CTC1	7	1, 3	CTC5	4	3, 6, 10
			CTC4	4	2,4,9				CTC7	4	3, 6, 8	CTC3	5	1, 5, 7, 8	CTC2	4	2, 4, 7, 9	CTC4	4	2, 4, 9	CTC1	7	1, 3
									CTC6	5	1, 7							CTC7	4	3, 6, 8	CTC4	4	2, 4, 9
									CTC4	4	2, 4, 9							CTC5	4	3, 6, 10	CTC3	5	1, 5, 7, 8
	Total execution time (units)			20			13			29			20			17			29			29	

Table 15: Adjacency matrix of pheromone deposited after the second iteration

	An.1	An.2	An.3	An.4	An.5	An.6	An.7
An.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.2	0.0	0.0	0.9	0.0	0.0	0.0	0.0
An.3	0.0	0.0	0.0	0.81	0.9	0.0	0.0
An.4	0.0	0.0	0.0	0.0	0.81	0.0	0.0
An.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 16: Execution of algorithm (3rd iteration)

ACO			An.1			An.2			An.3			An.4			An.5			An.6			An.7		
Third	Iteration	Observation	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$	PC	ET	$F_{lt.}C$
			CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC5	4	3, 6, 10	CTC6	5	1, 7	CTC7	4	3, 6, 8
			CTC6	5	1, 7	CTC4	4	2, 4, 9	CTC6	5	1, 7	CTC6	5	1, 7	CTC1	7	1, 3	CTC1	7	1, 3	CTC5	4	3, 6, 10
			CTC4	4	2, 4, 9	CTC7	4	3, 6, 8	CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC4	4	2, 4, 9
			CTC5	4	3, 6, 10	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC3	5	1, 5, 7, 8	CTC6	5	1, 7	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8
			CTC3	5	1, 5, 7, 8	CTC1	7	1, 3	CTC7	4	3, 6, 8	CTC7	4	3, 6, 8	CTC7	4	3, 6, 8	CTC5	4	3, 6, 10			
						CTC6	5	1, 7	CTC5	4	3, 6, 10	CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC5	5	1, 5, 7, 8			
						CTC5	4	3, 6, 10				CTC5	4	3, 6, 10									
	Total execution time (units)			25			33			29			33			29			29			17	

Table 17: Adjacency matrix of pheromone deposited after the third iteration

	An.1	An.2	An.3	An.4	An.5	An.6	An.7
An.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.2	0.0	0.0	0.81	0.0	0.0	0.0	0.0
An.3	0.0	0.0	0.0	0.729	0.81	0.0	0.0
An.4	0.0	0.0	0.9	0.0	0.729	0.0	0.0
An.5	0.0	0.0	0.0	0.9	0.0	0.0	0.0
An.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An.7	0.0	0.0	0.0	0.0	0.9	0.0	0.0

Table 18: Execution of algorithm (4th iteration)

ACO			An_1			An_2			An_3			An_4			An_5			An_6			An_7		
Fourth	Iteration	Observation	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$	PC	ET	$F_{It.}C$
			CTC1	7	1, 3	CTC2	4	2, 4, 7, 9	CTC3	5	1, 5, 7, 8	CTC4	4	2, 4, 9	CTC5	4	3, 6, 10	CTC6	5	1, 7	CTC7	4	3, 6, 8
			CTC5	4	3, 6, 10	CTC5	4	3, 6, 10	CTC7	4	3, 6, 8	CTC6	5	1, 7	CTC1	7	1, 3	CTC4	4	2, 4, 9	CTC1	7	1, 3
			CTC4	4	2, 4, 9	CTC1	7	1, 3	CTC6	5	1, 7	CTC2	4	2, 4, 7, 9	CTC2	4	2, 4, 7, 9	CTC7	4	3, 6, 8	CTC5	4	3, 6, 10
			CTC7	4	3, 6, 8	CTC3	5	1, 5, 7, 8	CTC2	4	2, 4, 7, 9	CTC1	7	1, 3	CTC4	4	2, 4, 9	CTC1	7	1, 3	CTC3	5	1, 5, 7, 8
			CTC2	4	2, 4, 7, 9				CTC1	7	1, 3	CTC3	5	1, 5, 7, 8	CTC7	4	3, 6, 8	CTC5	4	3, 6, 10	CTC2	4	2, 4, 7, 9
			CTC6	5	1, 7				CTC4	4	2, 4, 9	CTC7	4	3, 6, 8	CTC3	5	1, 5, 7, 8	CTC2	4	2, 4, 7, 9			
			CTC3	5	1, 5, 7, 8				CTC5	4	3, 6, 10	CTC5	4	3, 6, 10				CTC3	5	1, 5, 7, 8			
	Total execution time (units)			33			20			33			33			28			33			24	

Table 19: Adjacency matrix of pheromone deposited after the fourth iteration

	An_1	An_2	An_3	An_4	An_5	An_6	An_7
An_1	0.0	0.0	0.9	0.0	0.0	0.0	0.0
An_2	0.0	0.0	0.729	0.0	0.9	0.0	0.0
An_3	0.0	0.0	0.0	0.6561	0.729	0.0	0.0
An_4	0.0	0.0	0.81	0.0	0.6561	0.0	0.0
An_5	0.9	0.0	0.0	0.81	0.0	0.0	0.0
An_6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
An_7	0.0	0.0	0.0	0.0	0.81	0.0	0.0

7 Comparison

The proposed framework, founded upon the ACO technique, has been subjected to a rigorous comparative analysis against well-established traditional methods, which include No ordering, Reverse ordering, Random ordering, and Optimal ordering of the combined test cases. The results of this comparative evaluation are presented in Table 20.

Table 20: Ordering of test cases through different techniques

S. No.	N_o	Re_o	Rm_o	Ot_o	ACO
1	CTC1	CTC7	CTC5	CTC2	CTC2
2	CTC2	CTC6	CTC7	CTC3	CTC5
3	CTC3	CTC5	CTC2	CTC5	CTC1
4	CTC4	CTC4	CTC3	CTC7	CTC3
5	CTC5	CTC3	CTC4	CTC4	CTC7
6	CTC6	CTC2	CTC1	CTC6	CTC4
7	CTC7	CTC1	CTC6	CTC1	CTC6

The Average Percentage of Fault Detected (APFD), developed by Elbaum, quantifies the test suite's fault detection rate per percentage of test suite execution [34,35]. The goal of validation is to find the defect as quickly as possible via the optimal and shortest path, and this can be accomplished by applying optimised prioritisation criteria to the test suite. The APFD measure is used to compute and compare the outcomes of different approaches, as given in Eq. (1).

$$APFD = 1 - \left(\frac{(Tf1 + Tf2 + \ldots\ldots\ldots + Tfm)}{(mn)}\right) + (1/2n) \tag{1}$$

where 'n' represents the total number of test cases, and 'm' denotes the number of faults under consideration. The variables (Tf1, ..., Tfm) denote the positions of the initial test case 'T' in the sequence that manifest the identified faults.

Given: Total number of Combined Test Cases (CTC) = 7;

Total number of faults (Flt) = 10

CTC Flti denotes the position of the first test in CTC that exposes fault Flti

APFD metric for No ordering is given below:

CTC1 CTC2 CTC3 CTC4 CTC5 CTC6 CTC7

APFD = 1 − {(1 + 2 + 1 + 2 + 3 + 5 + 2 + 3 + 2 + 5)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 − {26/70} + {1/14}

= 1 − {0.37142857142} + {0.07142857142}

= 1 − {0.371} + {0.071}

= 1.071 − 0.371

= 0.700

Therefore, the results obtained through APFD metric for No ordering (in percentage) = 70%.

APFD metric for Reverse ordering is given below:

CTC7 CTC6 CTC5 CTC4 CTC3 CTC2 CTC1

APFD = 1 − {(2 + 4 + 1 + 4 + 5 + 1 + 2 + 1 + 4 + 3)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 − {27/70} + {1/14}

= 1 − {0.38571428571} + {0.07142857142}

= 1 − {0.386} + {0.071}

= 1.071 − 0.386

= 0.685

Therefore, the results obtained through the APFD metric of Reverse ordering (in percentage) = 68.5%.

APFD metric for Random ordering is given below:

CTC5 CTC7 CTC2 CTC3 CTC4 CTC1 CTC6

APFD = 1 − {(6 + 3 + 6 + 3 + 4 + 1 + 3 + 4 + 3 + 1)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 − {34/70} + {1/14}

= 1 − {0.48571428571} + {0.07142857142}

= 1 − {0.486} + {0.071}

= 1.071 − 0.486

= 0.585

Therefore, the results obtained through the APFD metric of Random ordering (in percentage) = 58.5%.

APFD metric for Optimal ordering is given below:

CTC2 CTC3 CTC5 CTC7 CTC4 CTC6 CTC1

APFD = 1 − {(7 + 1 + 2 + 1 + 2 + 3 + 1 + 2 + 1 + 3)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 − {23/70} + {1/14}

= 1 − {0.32857142857} + {0.07142857142}

= 1 − {0.329} + {0.071}

= 1.071 − 0.329

= 0.742

Therefore, the results obtained through the APFD metric of Optimal ordering (in percentage) = 74.2%.

APFD metric for ACO-based ordering is as given below:

CTC2 CTC3 CTC5 CTC7 CTC4 CTC6 CTC1

APFD = 1 − {(3 + 1 + 3 + 1 + 4 + 2 + 1 + 4 + 1 + 2)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 − {22/70} + {1/14}

= 1 − {0.31428571428} + {0.07142857142}

= 1 − {0.314} + {0.071}

= 1.071 − 0.314

$= 0.757$

Therefore, the results obtained through the APFD metric of ACO based ordering (in percentage) = 75.7%. Finally, the outcomes from these techniques are presented in Table 21.

Table 21: Results of test case optimization employing various traditional techniques

	N_o	Re_o	Rm_o	Ot_o	ACO
APFD metric results	0.700 70.00%	0.685 68.50%	0.585 58.50%	0.742 74.20%	0.757 75.70%

Now, the proposed approach with the other state-of-the-art optimization algorithm for prioritization of security test cases: Ant Lion Optimization (ALO), Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are compared. The values of parameters for ALO, GA and PSO are given in Table 22. The combined test case ordering obtained from the state-of-the-art algorithm and the proposed ACO algorithm is shown in Table 23. The APFD score for the ordering obtained using GA, PSO and GLA algorithms are given in Table 24.

Table 22: Parameter values of the state-of-the-art algorithm

Parameter	Value
Mutation rate r in ALO	Min = 0 and Max = 0.9
Crossover ratio in GA	0.9
Mutation ratio in GA	0.1
Selection mechanism in GA	Roulette wheel
Inertia w in PSO	[0.9, 0.6]
Acceleration constants in PSO	[2, 2]

Table 23: Ordering of test cases through state-of-the-art algorithm

S. No.	ALO	GA	PSO	ACO
1	CTC4	CTC3	CTC2	CTC2
2	CTC1	CTC4	CTC3	CTC5
3	CTC7	CTC7	CTC5	CTC1
4	CTC5	CTC1	CTC7	CTC3
5	CTC2	CTC2	CTC4	CTC7
6	CTC6	CTC6	CTC6	CTC4
7	CTC3	CTC5	CTC1	CTC6

Table 24: Results of test case optimization through state-of-the-art evolutionary techniques

	ALO ordering	GA ordering	PSO ordering	ACO based ordering
APFD metric results	0.642 64.20%	0.685 68.50%	0.742 74.20%	0.757 75.70%

APFD metric for ALO ordering is given below: CTC4 CTC1 CTC7 CTC5 CTC2 CTC6 CTC3

APFD= 1 – {(2+1+2+1+7+4+5+3+1+4)/(7 ∗ 10)} + {1/(2 ∗ 7)} = 1 – {30/70} + {1/14}

= 1 – {0.429} + {0.071}

=0.642

Therefore, the results obtained through the APFD metric of ALO ordering (in percentage) = 64.2%.

APFD metric for GA ordering is given below: CTC3 CTC4 CTC7 CTC1 CTC2 CTC6 CTC5

APFD= 1 – {(1+2+3+2+1+7+1+1+2+7)/(7 ∗ 10)} + {1/(2 ∗ 7)} = 1 – {27/70} + {1/14}

= 1 – {0.386} + {0.071}

=0.685

Therefore, the results obtained through the APFD metric of ALO ordering (in percentage) = 68.5%.

APFD metric for PSO ordering is given below:

CTC2 CTC3 CTC5 CTC7 CTC4 CTC6 CTC1

APFD = 1 – {(7+1+2+1+2+3+1+2+1+3)/(7 ∗ 10)} + {1/(2 ∗ 7)}

= 1 – {23/70} + {1/14}

= 1 – {0.32857142857} + {0.07142857142}

= 1 – {0.329} + {0.071}

= 1.071 – 0.329

=0.742

Therefore, the results obtained through the APFD metric of PSO ordering (in percentage) = 74.2%.

The value obtained from APFD metrics shows that the ACO-based algorithm outperforms the other state-of-the-art optimization algorithm. Table 25 shows their total results improvements. APFD measures reveal that ACO-based ordering outperforms all others. The above comparisons and APFD metric results suggest that the ACO technique is optimal for early defect detection. As a result, the ACO method is clearly superior to other methods.

Since the ACO technique provides the best possible order, we used this methodology for beforehand fault detection in security test suites and further mitigates the issues. The ACO-based algorithm gives an edge to the proposed testing techniques over traditional techniques as it is more result-oriented and less time-consuming. This will ultimately cut down the development cost of the application system. Hence, by applying the ACO-based algorithm, the security attributes, according to their weightage, are used to detect faults and provide the shortest path to mitigate those faults.

Table 25: Comparing ACO based ordering with other approaches

ACO-based ordering *vs.* others		Improvements	Percentage
ACO *vs.* N_o	0.757–0.700	0.057	8.14%
ACO *vs.* Re_o	0.757–0.685	0.072	10.51%
ACO *vs.* Rm_o	0.757–0.585	0.172	29.40%
ACO *vs.* Ot_o	0.757–0.728	0.029	3.9%
ACO *vs.* *ALO*	0.757–0.642	0.115	17.91%
ACO *vs.* *GA*	0.757–0.685	0.072	10.51%
ACO *vs.* *PSO*	0.757–0.742	0.015	2.02%

8 Discussion

The demand for security test case optimization for developing secure software is increasing day by day. Besides, the presence of vulnerabilities not only wreaks financial losses and causes time delays but can also be a threat to life. Modern software systems should ensure the reliability and security of the operation whenever they are employed. Improvised software security is the key differentiator between the products in the present-day environment of digitally enabled services. In a competitive marketplace, the significance of providing a secure end product is not an advantage but more of a necessity for the success of the organization. While there is uniform consensus among researchers and industry persons for the high need for secure software, the question of how, when and where the security should be measured is far from settled issues. The present study employed the ACO algorithm for prioritizing the security test cases. The ACO algorithm has been used in several existing studies for solving the problem of optimization with good performance [36–38]. Here, in our work, the ACO algorithm has outperformed the conventional techniques of ordering the test cases with significant improvement in terms of APFD metric. The ACO based has nearly 30% improvement over the random ordering technique. Moreover, the ACO has also produced better results than other evolutionary algorithms like GA, ALO and PSO. Since the graph generated by the test cases may dynamically change, contrary to the GA, the ACO algorithm can effectively handle such changes to produce better results. Moreover, the positive feedback among the ants involved helps in finding the optimal solutions. By using ACO, the fault detection of the security test cases may reduce the overall test cases. This would further help in reducing the computational cost. Thus, ACO based technique could be beneficial during regression testing where new test cases need to be generated continuously during the maintenance phase of the software.

9 Conclusion

Security experts are consistently working on inventive mechanisms to develop software systems with enhanced levels of security that meet the users' requirements. Software testing is a vital contributor to determining the efficacy of the systems. Testing procedures need to be realizable as well as efficacious. In this league, the present study undertook a novel experiment of evaluating the test cases impact on application security through an Ant Colony Optimization algorithm from the perspective of design. The APFD for the order obtained using the proposed ACO-based technique was better than the results obtained on order from other traditional and state-of-the-art optimization algorithms. The ACO-based ordering generated considerable improvement as per the APFD metric. The generated ordering can be utilized to detect security faults before they cause any serious problems in the working of the software. The conclusive results of the present study will be an emphatic contribution in the domain of security test case optimization. In the future, we will attempt to reduce fault detection loss to near zero and validate the algorithm in real-world applications. We will employ the improved ACO algorithm proposed in the literature, as well as the testing procedure. With the use of ACO, developers have the ability to establish the prioritizing of security test cases and utilize the approach as the foundation upon which to see the probable outcomes and impacts. In addition, it would be interesting to investigate the possibility of using other metrics, such as ANOVA and ANCOVA, in place of APFD in the approaches used to prioritize test cases.

Acknowledgement: The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through Large Group Research Project.

Funding Statement: The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through Large Group Research Project under Grant Number RGP2/249/44.

Author Contributions: Study conception and design: A. Attaallah, K. al-Sulbi, M. W. Khan, M. Faizan, D. Pandey; data collection: A. Alasiry, M. Marzougui, M. W. Khan, A. Agrawal, D. Pandey; analysis and interpretation of results: A. Attaallah, K. al-Sulbi, M. W. Khan, A. Agrawal, D. Pandey; draft manuscript preparation: A. Attaallah, K. al-Sulbi, A. Alasiry, M. Marzougui, M. W. Khan, M. Faizan, A. Agrawal, D. Pandey. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The authors confirm that the data supporting the findings of this study are available within the article.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. A. Khan, "From threat to security indexing: A casual chain," *Computer Fraud and Security*, vol. 5, no. 5, pp. 9–12, 2009.

[2] L. Zhou, L. Tang and Z. Zhang, "Extracting and ranking product features in consumer reviews based on evidence theory," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, pp. 9973–9983, 2022.

[3] U. Waheed, "Security regression testing framework for web application development," MS Thesis, Department of Informatics, University of Oslo, Norway, 2014. [Online]. Available: https://www.duo.uio.no/handle/10852/43442 (accessed on 02/02/2023)

[4] Y. Singh, A. Kaur and B. Suri, "Test case prioritization using ant colony optimization," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 4, pp. 1–7, 2010.

[5] A. Agrawal, M. Alenezi, S. A. Khan, R. Kumar and R. A. Khan, "Multi-level fuzzy system for usable-security assessment," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 3, pp. 657–665, 2022.

[6] Z. Zhang, J. Guo, H. Zhang, L. Zhou and M. Wang, "Product selection based on sentiment analysis of online reviews: An intuitionistic fuzzy TODIM method," *Complex & Intelligent Systems*, vol. 8, pp. 3349–3362, 2022.

[7] K. Ayari, S. Bouktif and G. Antoniol, "Automatic mutation test input data generation via ant colony," in *Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation*, New York, USA, pp. 1074–1081, 2007.

[8] M. Dorigo, V. Maniezzo and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transaction on Systems, Man and Cybernetics*, vol. 26, no. 5, pp. 29–41, 1996.
[9] J. H. Holand, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.
[10] E. Emary, H. M. Zawbaa and A. E. Hassanien, "Binary ant lion approaches for feature selection," *Neurocomputing*, vol. 213, pp. 54–65, 2016.
[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN'95—Int. Conf. on Neural Networks*, Perth, WA, Australia, vol. 4, pp. 1942–1948, 1995.
[12] Q. Y. Bo, W. Cheng and M. Khishe, "Evolving chimp optimization algorithm by weighted opposition-based technique and greedy search for multimodal engineering problems," *Applied Soft Computing*, vol. 132, pp. 109869, 2023.
[13] L. Liu, M. Khishe, M. Mohammadi and A. H. Mohammed, "Optimization of constraint engineering problems using robust universal learning chimp optimization," *Advanced Engineering Informatics*, vol. 53, pp. 101636, 2022.
[14] E. D. Demircioğlu and O. Kalipsiz, "API message-driven regression testing framework," *Electronics*, vol. 11, no. 17, pp. 1–16, 2022.
[15] P. Jung, S. Kang and J. Lee, "Efficient regression testing of software product lines by reducing redundant test executions," *Applied Sciences*, vol. 10, no. 23, pp. 1–21, 2020.
[16] Y. Lou, J. Chen, L. Zhang and D. Hao, "A survey on regression test case prioritization," *Advances in Computers, Elsevier*, vol. 113, pp. 1–46, 2019.
[17] M. Taghavi and M. Khishe, "A modified grey wolf optimizer by individual best memory and penalty factor for sonar and radar dataset classification," *Journal of Marine Science University of Imam Khomeini*, vol. 15, pp. 122–132, 2019.
[18] 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, pp. 1–22, 2022.
[19] S. Saju Sankar and S. S. Vinod Chandra, "An ant colony optimization algorithm based automated generation of software test cases," in *Advances in Swarm Intelligence: 11th Int. Conf., ICSI 2020*, Belgrade, Serbia, Cham, Springer, vol. 12145, pp. 231–239, 2020.
[20] W. Zhang, Y. Qi, X. Zhang, B. Wei, M. Zhang *et al.,* "On test case prioritization using ant colony optimization algorithm," in *21st Int. Conf. on High-Performance Computing and Communications, IEEE 17th Int. Conf. on Smart City, IEEE 5th Int. Conf. on Data Science and Systems*, Zhangjiajie, China, pp. 2767–2773, 2019.
[21] J. Ning, C. Zhang, P. Sun and Y. Feng, "Comparative study of ant colony algorithms for multi-objective optimization," *Information*, vol. 10, no. 1, pp. 1–19, 2018.
[22] S. Nayak, C. Kumar and S. Tripathi, "Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection," *Arabian Journal for Science and Engineering*, vol. 42, no. 8, pp. 3307–3323, 2017.
[23] R. K. Sahoo, D. Ojha, D. P. Mohapatra and M. R. Patra, "Automated test case generation and optimization: A comparative review," *International Journal of Information Technology and Computer Science*, vol. 8, no. 5, pp. 19–32, 2016.
[24] S. A. A. Hridoy, F. Ahmed and M. S. Hossain, "Regression testing based on hamming distance and code coverage," *International Journal of Computer Applications*, vol. 120, no. 14, pp. 1–5, 2015.
[25] T. Muthusamy and K. Seetharaman, "Effectiveness of test case prioritization techniques based on regression testing," *International Journal of Software Engineering & Applications*, vol. 5, no. 6, pp. 113–123, 2014.
[26] A. Pravin and S. Srinivasan, "Effective test case selection and prioritization in regression testing," *Journal of Computer Science*, vol. 9, no. 5, pp. 654–659, 2013.
[27] G. Rothermel, R. J. Untch and C. Chu, "Prioritizing test cases for regression testing," *IEEE Transaction on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.
[28] T. L. Graves, M. J. Harrold, M. J. Kim, A. Porter and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transaction Software Engineering and Methodology*, vol. 10, no. 2, pp. 145–159, 2001.

[29] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. thesis, Politecnico di Milano, Milano, 1992.

[30] M. Dorigo and S. Krzysztof, "An Introduction to ant colony optimization," in *Proc. of Approximation Algorithms and Metaheuristics*, pp. 1–19, Bruxelles, Belgium: Chapman and Hall/CRC, 2006.

[31] L. Li, S. Ju and Y. Zhang, "Improved ant colony optimization for the travelling salesman problem," in *Int. Conf. on Intelligent Computation Technology and Automation*, Changsha, China, IEEE, pp. 76–80, 2008.

[32] C. A. Silva and T. A. Runkler, "Ant colony optimization for dynamic traveling salesman problems," in *ARCS 2004–Organic and Pervasive Computing*, Munich, Germany, pp. 259–266, 2009.

[33] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer and R. S. Robert, "Time-aware test suite prioritization," in *ISSTA'06: Proc. of Int. Symp. of Software Testing and Analysis*, Portland, USA, pp. 1–12, 2006.

[34] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Prioritizing test cases for regression testing," in *ISSTA'00: Proc. of ACM SIGSOFT Int. Symp. of Software Testing and Analysis*, Portland, USA, pp. 102–112, 2000.

[35] S. Elbaum, G. Rothermel, S. Kanduri and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185–210, 2004.

[36] A. Vescan, C. M. Pintea and P. C. Pop, "Test case prioritization—ANT algorithm with faults severity," *Logic Journal of the IGPL*, vol. 30, no. 2, pp. 277–288, 2022.

[37] G. Shaheamlung and K. Rote, "A comprehensive review for test case prioritization in software engineering," in *2020 Int. Conf. on Intelligent Engineering and Management*, IEEE, pp. 331–336, 2020.

[38] A. Vescan, C. M. Pinteaand and P. C. Pop, "Solving the test case prioritization problem with secure features using ant colony system," in *12th Int. Conf. on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th Int. Conf. on European Transnational Education (ICEUTE 2019)*, Cham, Springer, pp. 67–76, 2019.