JSCDM

JOURNAL OF SOFT COMPUTING AND DATA MINING
e-ISSN: 2716-621X

Vol. 7 No. 2 (2026) 1-16
https://publisher.uthm.edu.my/ojs/index.php/jscdm

Comparative Analysis of Test Case Prioritization Using Swarm Intelligence Algorithms to Enhance Fault Detection Rate and Execution Time

Nur Atiqah Wahidah Sulaiman[1], Nurezayana Zainal[1]*

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

*Corresponding Author: nurezayana@uthm.edu.my
DOI: https://doi.org/10.30880/jscdm.2026.07.02.001

Article Info

Received: 29 September 2025
Accepted: 16 April 2026
Available online: 30 June 2026

Keywords

Swarm intelligence algorithm, test case prioritization, particle swarm optimization, whale optimization algorithm, fault detection, optimization

Abstract

Test Case Prioritization (TCP) is an essential process in software testing that improves fault-detection rates and reduces execution time by determining optimal test ordering. However, traditional manual testing struggles to manage large-scale test suites, while automated methods often demand significant resources. To overcome these challenges, most researchers use metaheuristic algorithms for their efficiency. In particular, Swarm Intelligence (SI), inspired by the behavior of natural systems, has emerged as a powerful tool for solving optimization problems and offers promising solutions for TCP. Therefore, this study aims to compare two promising SI algorithms, namely the Whale Optimization Algorithm (WOA) and the Particle Swarm Optimization Algorithm (PSO), to improve fault detection rate and execution time. Using sample test suites containing 8 test cases with 10 detected faults and 15 test cases with 15 detected faults as datasets. Further, this research analyzes the performance of these algorithms using measures such as Average Percentage of Fault Detection (APFD) and execution time. The results reveal that the algorithms prioritize performance differently, with WOA achieving higher APFD than PSO in two case studies, at 0.813 and 0.789, respectively. In contrast with PSO, it minimizes execution time compared with WOA, achieving 0.026 and 0.103 seconds in two case studies.

1. Introduction

Competent testing procedures are necessary in contemporary software development to ensure software quality and meet user expectations. As software systems become increasingly complex, comprehensive testing becomes vital for managing large test suites and uncovering potential defects. Software testing (ST) can be categorized into two types: manual and automated. Manual testing entails carrying out test cases by hand using methods like white box and black box testing. Commonly used in both functional and non-functional testing settings, black box testing assesses application behavior without looking at internal structures [1]. However, since manual testing takes a lot of time and effort, it becomes impractical as system complexity rises.

To overcome these difficulties, automated testing uses software tools to automate repetitive activities, reduce human involvement, and more accurately identify errors [2]. Acceptance testing, performance testing, regression testing, and unit testing are common forms of automated testing. Regression testing (RT) is vital for software to remain stable following updates or alterations. Regression test efficiency is improved by the use of

optimization strategies like test case minimization (TCM), Test Case Selection (TCS), and TCP. Among these, TCP is acknowledged for its cost-effectiveness and robust fault detection, as it reorders test cases to identify errors earlier in the process [3][4].

Despite its advantages, the huge solution space of potential test case orders might make it computationally demanding to implement TCP effectively. Metaheuristic algorithms offer effective, flexible, and problem-independent methods for navigating intricate optimization problems in order to overcome this challenge [5]. Techniques like Tabu Search (TS) [6], Genetic Algorithm (GA)[7], PSO [8], Simulated Annealing (SA) [6], and Differential Evolution (DE) [6] are examples of algorithms that draw inspiration from natural processes [9]. The three main categories of metaheuristics are swarm-based, physics-based, and evolution-based approaches [10].

A subclass of metaheuristics known as swarm intelligence (SI) algorithms is especially good at solving multidimensional problems because they imitate cooperative behavior found in nature. Among these are the Firefly Algorithm (FA)[11], Artificial Fish School (AFS)[12], Artificial Bee Colony (ABC)[13], Ant Colony Optimization (ACO)[14], PSO[15], WOA[8]. In order to efficiently explore and exploit the solution space, these algorithms use decentralized communication among agents [16].

In the context of test case prioritization, this paper compares two promising SI algorithms which were WOA and PSO. WOA has demonstrated great exploratory capabilities but is still underutilized in TCP applications. It was inspired by the humpback whale's bubble-net hunting strategy [17]. Bird flocking behavior is modeled by PSO, which is renowned for its ease of use, quick convergence, and robust performance across a range of optimization tasks [18][19]. Applying WOA and PSO to datasets from [20] and [21] will allow this study to evaluate the effectiveness as well as efficiency of each algorithm in terms of execution time and fault detection rate. Which algorithm provides a more accurate and efficient solution for TCP will be determined by the findings, which will ultimately lead to quicker and more dependable software development and delivery.

2. Literature Review

This section provides an overview of the domain related to this study and summarizes the existing literature on TCP. A review of the acknowledged literature indicates that various researchers have implemented various SI algorithms to improve TCP.

2.1 Regression Testing

Throughout the Software Development Life Cycle (SDLC), ST is essential to produce software that satisfies functional requirements and is error-free. Through numerous methods, techniques, and processes, it guarantees the software's accuracy, dependability, security, and general quality [22]. Testing can be partitioned into a number of areas, such as requirements-based testing, usability testing, developmental testing, and RT [23].

In order to make sure that bug fixes or newly introduced features do not adversely affect already-existing functionalities, RT is one of the most dominant testing types during the maintenance phase [24]. However, this procedure can be costly and time-consuming since it often entails running the complete test suite [25]. There are a few optimization strategies that are commonly used to overcome these issues, which have been presented earlier: TSM, TCS, and TCP. TSM seeks to minimize the number of test cases while preserving the effectiveness of fault detection [4]. Conversely, only test cases relevant to current code changes should be identified and run as part of TCS [26]. In contrast, TCP allocates a ranking to test cases, ensuring that the ones with the best chance of identifying errors are run first [27]. These techniques improve RT efficiency and effectiveness through resource optimization and increased fault detection rates in a constrained amount of testing time.

2.2 Test Case Prioritization

TCP is the process of optimizing one or more test goals in line with a particular strategy to determine the test case set of either the whole or part of the test case priority order of execution [12]. Test cases are prioritized based on pre-established testing criteria, including fault detection, maximum coverage, and decreased execution time [34]. According to [41], TCP can help lower the cost of carrying out the testing procedure.

The APFD and execution time are the performance metrics regularly evaluated based on the pre-established testing criteria. Therefore, even with limited resources, TCP can improve the software's attributes, making repeating the testing procedure more effective and efficient [25].

Additionally, many researchers have employed TCP to decrease execution time and boost the fault detection rate. The majority of researchers today use swarm intelligence algorithms to tackle TCP in order to use metaheuristic techniques. Numerous algorithms have been presented. Examples of algorithms influenced by nature-behavior include WOA [8], PSO [15], ACO [14], ABC [13], FA [11], and numerous others. Some researchers even expand certain algorithms in order to optimize the outcome, such as the Multi-Objective Particle Swarm Optimization algorithm (MOPSO) for TCP, which combines the multi-objective technique with PSO.

J. of Soft Computing and Data Mining Vol. 7 No. 2 (2026) p. 1-16

2.3 Implementation on Test Case Prioritization Using Metaheuristic Algorithms

Table 1 indicates the comparison of the algorithm used in TCP within several studies from previous research.

Table 1 *Comparison of algorithms used in TCP*

Author(s)	Algorithm(s)	Summary
Khatibsyarbini M. et al.[27]	Firefly Algorithm (FA)	The APFD score for FA is higher than that of other algorithms, where FA (0.9517), Local Beam Search (0.9461). The same goes for the execution time, where FA slightly outperformed Local Beam Search (LBS), with LBS getting 400 seconds while FA got 390 seconds.
Nayak S. et al.[28]	Artificial Bee Colony (ABC) algorithm with fuzzy rule base	BCO got the highest APFD results for both projects 1 and 2, where it got 85% and 82.5%, and the other algorithms got lower than 80%, excluding the prioritized order by Tyagi and Malthora (82%) and the prioritized order by Nayak et al. (81%) for P1.
Ariffin M. et al.[29]	Ant Colony Optimization (ACO) and Firefly Algorithm (FA)	The APFD values for ACO and FA are the same for both case studies. As for execution time and fault coverage, FA outperformed ACO. However, ACO can solve the TCP problem directly, as FA needs an objective function to solve the problem that led to ACO getting better prioritization performance than FA.
Alkawaz M. et al.[30]	Modified Ant Colony Optimization Algorithm (m-ACO)	The days taken to complete software RT are 45 days, while without using the m-ACO is 90 days.
Mugilan A. et al.[31]	Ant Colony Optimization (ACO) and multi-objective particle swarm optimization algorithm (MOPSO)	MOPSO is better in TCP because it finds test cases that cover every fault first, then uses a method to rank them according to priority.
Bajaj A. et al.[32]	Improved novel bat algorithm (iBAT)	The algorithm was compared with several algorithms, including the whale optimization algorithm, general algorithm, bat algorithm, random search, improved bat algorithm, and novel bat algorithm. But iBAT and WA have better results compared to others in TCP.
Mohan K. et al.[13]	Artificial Bee Colony (ABC) and Ant Colony Optimization (ACO)	ABC got the highest value in fault detection rate, but ACO took the shortest time in TCP. Thus, both algorithms are equally good in terms of both performance metrics.
Singhal S. et al.[33]	African Buffalo Optimization (ABO) and Ant Colony Optimization (ACO)	The time and size drop for both algorithms are the same; meanwhile, the APFD ACO is slightly better than the ABO, but the value is close and near optimal.
Attallah A. et al.[35]	Ant Colony Optimization (ACO)	The ACO has a higher APFD compared to Ant Lion Optimization and GA, but is just slightly higher than PSO.
Yang B. et al. [45]	Improved Whale Optimization Algorithm (REM-WOA)	The proposed algorithm (REM-WOA) was compared with a few algorithms, such as WOA, PR-WOA, CAW- WOA, AFSA, and IA. The proposed algorithm showed better performance compared to other algorithms, as it has strong global search ability and optimization balance. However, it may reduce the efficiency of the TCP and increase the iterations needed.
Nazir M. et al.[20]	Multi-Goal Particle Swarm Optimizer for Test Case Prioritization (MOPSO)	MOPSO got the lowest execution time compared to other algorithms (No Order, Random Order, Reverse Order, and GA), while the APFD of MOPSO got more coverage compared to GA.

Anuar M. et al.[37]	Ant Colony Optimization (ACO) and Genetic Algorithm (GA)	Both algorithms' performance is almost the same. However, GA shows a bigger value for APFD and a shorter amount of time to execute. As for Big-O notation, both algorithms have the same notation, which is $O(n^2)$.
Kumari S. et al. [43]	Grey Wolf Algorithm (GWO)	GWO has outperformed a few traditional approaches, Actual Testcase Prioritization (ATP), Random Form Testcase Prioritization (RFTP), and Highest Rate Testcase Prioritization (HRTP) for APFD value, whereby the values are 94.24%, 86.44%, 80.03% and 78.88%, respectively.
Assiri M. [44]	Dragon Boat Optimization Algorithm (DBOA)	Four (4) datasets were used in this study, which are GZIP, GREP, TCAS, and CS-TCAS, while the algorithm was being compared with several techniques, such as MHHO-TCP, FA techniques, PSD techniques, LBS techniques, and Greedy. The results showed that DBOA outperformed all techniques used for all datasets, 96.62%, 96.90%, for GZIP and GREP, while TCAS and CS-TCAS got the same value, 94.80%.

Khatibsyarbini M. et al. utilized the FA to challenge the TCP issue, assessing its performance using the APFD and the time taken for execution. The results indicated that FA surpassed several other algorithms, including PSO, LBS, the Greedy Algorithm, and the GA, on both performance metrics. Nevertheless, their research focused mainly on fault detection, recommending that subsequent studies should investigate improving test coverage efficiency within the TCP framework.

In a similar investigation, [28] introduced an ABC algorithm enhanced with a fuzzy rule base to optimize and prioritize regression test suites. The main performance metric examined was APFD, which was compared to baseline methods such as no prioritization, random prioritization, and reverse prioritization. The introduced approach achieved an APFD of 82.5%, the highest among the methods evaluated. However, the study had some drawbacks in that the assessment was limited to effectiveness in ordering without reflecting other significant aspects like scalability and robustness. Additionally, the algorithm showed instability when applied to bulkier datasets and, due to its irregularity characteristics, sometimes failed to reach an optimal solution.

Ariffin M. et al. employed both FA and ACO to the TCP problem in two case studies, evaluating their efficiency and effectiveness based on APFD, execution time, and fault coverage. The findings proposed that both algorithms produced similar APFD scores across the case studies. However, FA revealed improved execution efficiency and achieved complete fault coverage, compared to ACO's 90% coverage. Despite these benefits, the researchers concluded that ACO presented superior overall prioritization performance, mainly because it addressed the TCP problem more directly, whereas FA depended on an externally defined objective function. In conclusion, while FA provided swifter execution and comprehensive fault detection, ACO was considered more effective in terms of prioritization accuracy due to its precise design for the problem.

In a research study by [30], a modified version of the ACO (m-ACO) was proposed to improve the optimization of software regression test cases. The main performance metric assessed was the total time required to finish the process, applied within the Zasta Billing Web Application. The results showed a marked enhancement in efficiency, which, without m-ACO, the SRT process took 90 days, while implementing m-ACO reduced the time to 45 days. Additionally, the researchers used the APFD metric to assess fault-detection effectiveness during the optimization phase.

In a similar line, [31] examined TCP through two algorithms, ACO and MOPSO. Test cases were arranged according to their prioritization, and both algorithms achieved the same Average Percentage of Test Cases (APTC) coverage, recorded at 0.1875. Nonetheless, MOPSO was considered superior due to its two-phase TCP strategy, which first determined test cases that collectively identified all known defects and then applied a prioritization formula. For future studies, the authors proposed assessing the method with larger test suites and looking into enhanced algorithms like e-ACO or other metaheuristics that could improve the efficiency and cost-effectiveness of test case prioritization.

Bajaj A. et al. presented the iBAT for both TCP and TSM. The study compared iBAT against various algorithms, including the WOA, the original novel bat algorithm, GA, the bird swarm algorithm, the standard bat algorithm, and random search. The findings revealed that all nature-inspired algorithms outperformed random search, with iBAT consistently demonstrating the best performance in TCP. Interestingly, WOA showed comparable effectiveness to iBAT in terms of statement coverage. Experiments carried out on three software systems, which covered Jtopas, Ant, and JMeter, showed that iBAT achieved the highest APFD scores across all applications, with values of 95.621, 93.937, and 93.696, respectively, reflecting its superior fault detection ability.

Mohan K. et al. employed ABC and ACO algorithms for TCP with the objective of enhancing fault detection and minimizing execution time in web applications. The evaluation was conducted across two case studies using

APFD and execution time as performance metrics. In Case Study One, ABC demonstrated superior fault detection capabilities, achieving an APFD of 0.80 compared to ACO's 0.71. However, ACO exhibited faster execution, completing the test in 0.69 seconds versus ABC's 8.64 seconds. A similar tendency was observed in Case Study Two, where ABC got a higher APFD of 0.81, while ACO attained 0.64. Thus, it showed that ACO outperformed ABC in execution time, completing the process in 1.22 seconds as opposed to ABC's 8.83 seconds. These findings propose that ABC is more effective in detecting faults, while ACO offers advantages in execution efficiency.

Similarly, [33] utilized the ABO algorithm in both TCP and TCS, comparing it in contradiction of ACO to assess its capability. The comparison applied metrics such as time reduction, size reduction, and APFD. Both algorithms returned identical reductions in execution time (48.57%) and test suite size (62.5%). Nevertheless, ACO slightly outperformed ABO in fault detection, recording an APFD of 82.5% compared to ABO's 80%. These outcomes showed that ABO performs comparably to ACO, with only minimal differences in fault detection effectiveness.

Attallah A. et al. utilized ACO in the context of security-focused TCP, with APFD as the primary evaluation metric. The algorithm was benchmarked against conventional ordering strategies such as no ordering, random ordering, reverse ordering, and optimal ordering. ACO achieved an APFD of 75.7%, surpassing these traditional techniques. Furthermore, when compared to other nature-inspired algorithms, Ant Lion Optimization (ALO), GA, and PSO. Nonetheless, ACO demonstrated improved performance, notably exceeding PSO, which achieved an APFD of 74.2%.

Yang B. et al. presented an enhanced WOA for TCP, comparing its performance to the AFSA and the Immune Algorithm (IA), both of which are recognized for their global search abilities and resistance to local optima. The study evaluated performance using APTC and execution time. In addition to AFSA and IA, the proposed WOA variant was benchmarked against standard WOA, as well as three enhanced versions, which were Population Redistribution-based WOA (PR-WOA), Convergence Adaptive Weighting-based WOA (CAW-WOA), and Reinforced Exploration Mechanism WOA (REM-WOA). The proposed algorithm achieved higher APTC values than AFSA and IA and required less execution time than all other compared algorithms. In real-world implementation, REM-WOA yielded the highest APTC of 0.97, while AFSA and IA yielded lower values, confirming the efficiency of the enhanced WOA approach.

Nazir M. et al. suggested a MOPSO algorithm for TCP, improving upon the traditional PSO. MOPSO achieved an APFD of 85%, outperforming traditional strategies such as random (81.25%), reverse (80%), and no ordering (81.25%). Additionally, MOPSO surpassed the GA, which achieved an APFD of 83.75%, and proved better execution time relative to all other compared algorithms.

On the other hand, [37] conducted a comparative study of ACO and GA within the TCP domain, using APFD, execution time, and Big-O complexity as evaluation criteria across two case studies. In Case Study One, ACO slightly outperformed GA with an APFD of 0.72 compared to 0.71, while in Case Study Two, ACO and GA achieved APFDs of 0.70 and 0.67, respectively. In contrast, GA demonstrated faster execution: 0.21 seconds versus 0.54 seconds for Case Study One, and 0.19 seconds versus 0.35 seconds for Case Study Two. Both algorithms exhibited similar theoretical efficiency, with time complexity classified as $O(n^2)$ in both case studies.

Kumari S. et al. indicated that the test case optimization method based on GWO regularly reduces the size of the test suite while outperforming conventional prioritization strategies in terms of APFD. The suggested approach yields APFD values between 85% and 98% across 17 programs, with an average of 94.24%. In contrast, ATP, RFTP, and HRTP have APFD values of 86.44%, 80.03%, and 78.88%, respectively. According to these findings, Grey Wolf Optimization is better at identifying errors early on without sacrificing execution performance, which emphasizes how appropriate it is for prioritizing regression test cases.

As a final point, [44] has demonstrated that by simultaneously optimizing APFD and execution time, the TCP-DBOA technique shows strong efficacy in test case prioritization. DBOA consistently outperforms current methods, according to experimental assessments on four benchmark datasets, with APFD values of 96.62% and 96.90% for GZIP and GREP and 94.80% for both TCAS and CS-TCAS. In comparison to MHHO-TCP, FA, PSD, LBS, and Greedy techniques, these results show improved early defect detection capacity. However, the author also stated that the method is constrained by its dependence on a predetermined set of benchmarks, lack of scalability analysis, and lack of real-time adaptation, despite its encouraging performance. Therefore, even though TCP-DBOA offers compelling empirical proof of the potential of swarm-based optimization for TCP, more research is necessary to confirm its efficacy in extensive and dynamic software testing.

Swarm intelligence algorithms are useful for enhancing fault detection and optimization efficiency in test case prioritization, according to the summary shown in Table 1. Nevertheless, most of the previous research emphasizes a limited set of algorithms or optimizes a single performance metric without taking execution time into account. Newer algorithms like the WOA are still relatively unexplored, and there are few direct comparison studies conducted in consistent research settings, despite the broad use of techniques like PSO and ACO. PSO is selected due to its proven convergence accuracy and robustness, particularly in the context of test case prioritization, where it has been extensively adopted in prior studies [19]. In contrast, the WOA is chosen because it requires fewer control parameters and exhibits a strong independent search mechanism with a fast convergence rate, making it a promising alternative for optimization tasks in this domain [44]. These gaps motivate this study,

which conducts an orderly comparison of PSO and WOA using both fault detection effectiveness and execution time, thereby providing precise findings into their relative performance in test case prioritization. Moreover, the theoretical complexity of both algorithms is established using Big-O notation by using an asymptotic analysis. This research emphasizes the underlying trade-offs between PSO and WOA by combining algorithmic complexity with empirical performance data, offering a theoretically supported approach to adopting swarm intelligence algorithms in software quality assurance.

2.4 Overview of WOA

The encircling of prey, bubble-net attacking (which involves spiral updating and shrinking encircling based on a random value p ∈ [0,1]) [35], and random exploration [36] are the three main actions of humpback whales that motivated the WOA. It has an easy structure, a quick convergence rate, and a logical structure that makes it effective in both local and global searches [38]. Although there are fewer studies that only concentrate on WOA, many researchers improve it or use it in conjunction with other methods. WOA, however, continues to be a widely used benchmark, outperforming both ACO and the enhanced Bat Algorithm (iBAT). Fig. 1 shows the WOA process.

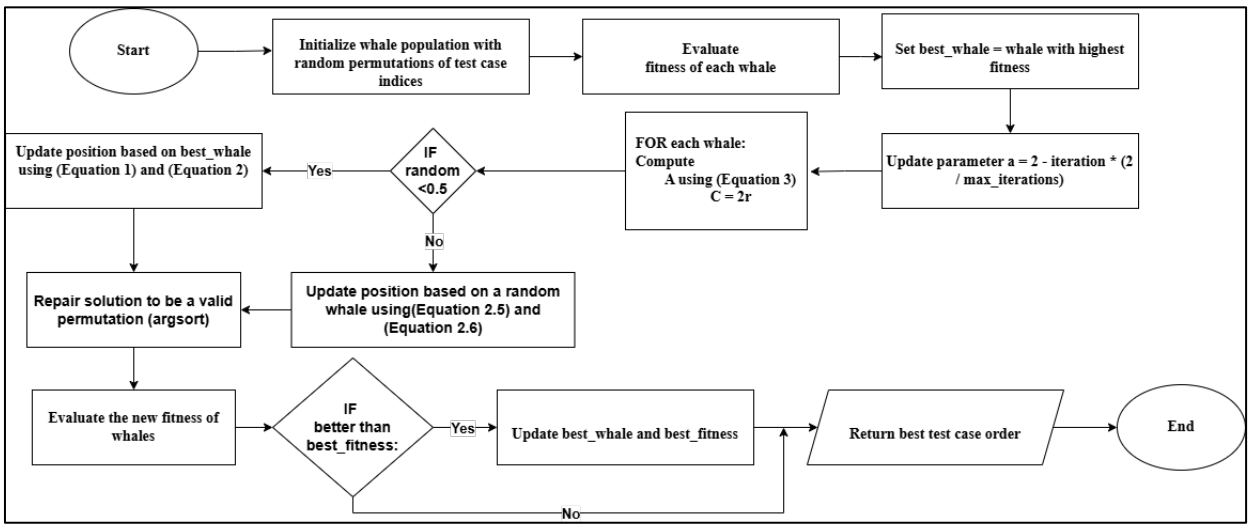

Fig. 1 *Flowchart for WOA process [15]*

According to [15], the whale will encircle the prey, but since it cannot tell where the ideal target is, it will approach it by treating the present optimal target as the global optimal target. The mathematical model for this phase is as in Equations 1 and 2:

$$D = \left| C \cdot x^{*(t)} - x^{(t)} \right| \tag{1}$$

$$x^{(t+1)} = x^{*(t)} - A \cdot D \tag{2}$$

$x^{*(t)}$ is the current local optimal solution while A and C being the coefficient vectors and A is being calculated in Equation 3:

$$A = 2a \,.r - a \tag{3}$$

where r is a random vector that falls between 0 and 1, and a has a linear decline from 2 to 0. The second phase is bubble-net attacking which comprises two methods which are shrinking encircling and spiral movement. The whales use the spiral movement to get closer to the most well-known solution [38]. This phase can also be called the exploitation phase. The mathematical equation for shrinking encircling is the same in equation 3 after updating the position of the new best whale. The last phase of WOA is random exploration by mimicking their haphazard hunt for prey. The whales contribute diversity to the population by moving toward random locations in the search space. The equation is as in Equations 4 and 5:

$$D = \left| C \cdot x^{*} - x^{t} \right| \tag{4}$$

J. of Soft Computing and Data Mining Vol. 7 No. 2 (2026) p. 1-16

$$x^{t+1} = x_{rand} - A \cdot D \tag{5}$$

where x_{rand} is the position of the randomly selected whale, and D is the distance to the global optimal target.

2.5 Overview of PSO

PSO refers to particles that interact in the solution space to save energy, with each particle continuously refining its search strategy based on the population's experience [40]. Fig 2 shows the flowchart of the PSO algorithm.

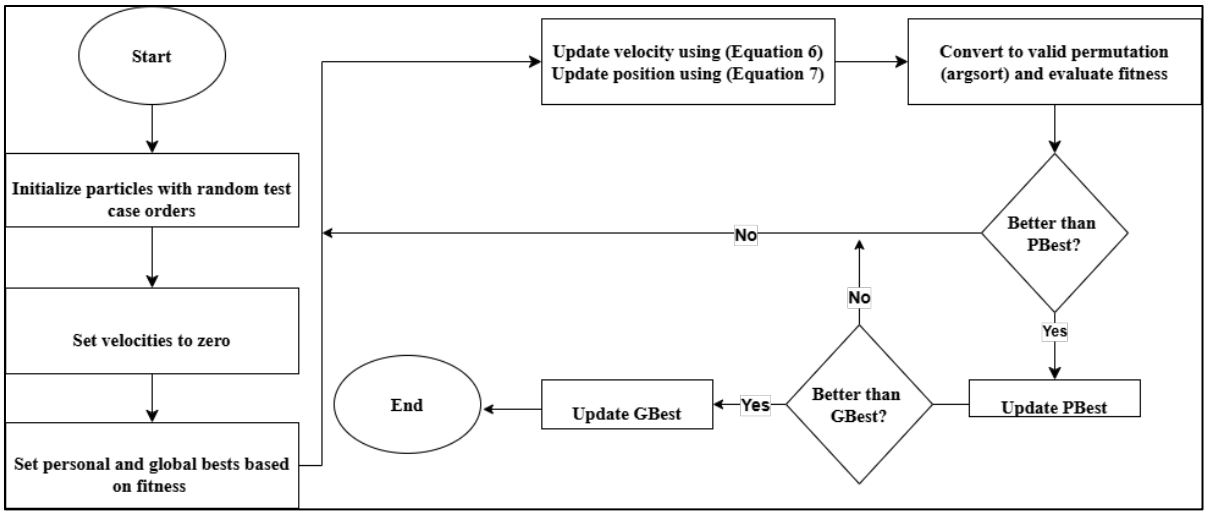

Fig. 2 *Flowchart for PSO process [25]*

Each particle's position is random, moving in various directions [41]. Each particle iteratively approaches the desired result by changing itself in response to its own and the group's experiences. It contains two characteristics, which are the position vector and velocity vector. The mathematical equations for both velocity and position are in Equations 6 and 7.

$$v_i^{(t)} = \omega \cdot v_i^{(t)} + c_1 \cdot rand_1() \cdot (pBest_i^{\,d} - x_i^{\,d}) + c_2 \cdot rand_2() \cdot (gBest_i^{\,d} - x_i^{\,d}) \tag{6}$$

$$x_i^{(t)} = x_i^{(t)} + v_i^{(t)} \tag{7}$$

where ω is the inertia weight, while c_1 and c_2 are acceleration coefficients. For $rand_1()$ and $rand_2()$ are any random numbers between 0 and 1. For $pBest$, it indicates the position for each bird and $gBest$ is the current optimal position. v indicates the velocity and x indicates the position vector.

3. Methodology

This section outlines the research flowchart used in this study, as shown in Fig 3. There are four important phases to achieve the objectives: problem definition, data definition, implementation, and analysis of results. The study began by identifying the research gap through a review of previous research work. In the next phase, two datasets were defined and collected, sourced from studies conducted by [20] and [21]. Then, PSO and WOA algorithms will be implemented in both datasets. Finally, the results obtained from the implementation will be analyzed, focusing on APFD and execution time to evaluate the effectiveness and efficiency of the algorithms.

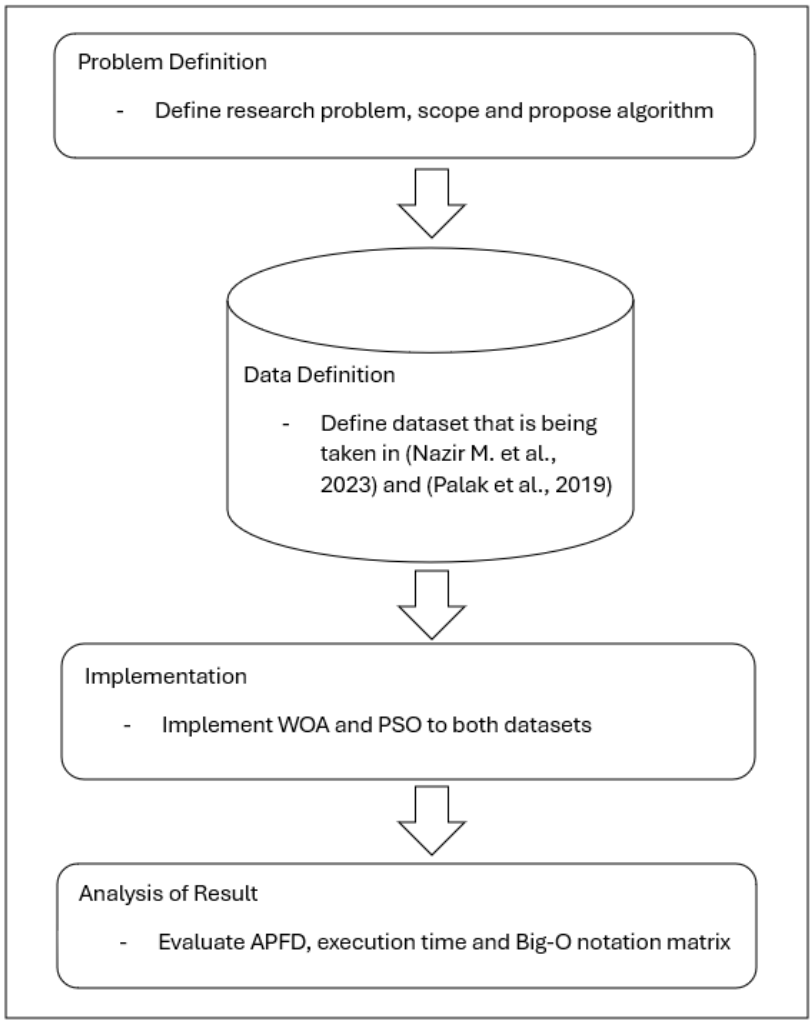

Fig. 3 *Research flowchart*

3.1 Problem Definition

During this phase, TCP techniques implemented by previous researchers are thoroughly reviewed, including existing algorithms, their limitations, and the challenges in improving fault detection rates and execution speed. In addition, the research objectives, scopes, and the two algorithms selected for this study, WOA and PSO, are clearly outlined. Specific research goals are defined, focusing on the application of metaheuristic algorithms and the identification of key performance metrics to evaluate their effectiveness.

The main problem this study addresses is the inadequacy of traditional TCP techniques in dealing with complex software systems. As software systems become increasingly complex, conventional manual testing methods often fail to detect faults efficiently within an acceptable timeframe. Moreover, existing automated solutions tend to require meaningful resources for setup and maintenance. There is also an inconsistency in how current algorithms perform in terms of competence. Therefore, this study aims to enhance TCP performance by employing swarm intelligence (SI) approaches, namely WOA and PSO, as these methods have shown strong potential for solving complex optimization problems such as TCP.

3.2 Data Definition

This study uses two datasets: Nazir M. et al. [20] and Palak et al. [21]. The datasets consist of sample test suites labeled as $TC1, TC2, TC3, \ldots, Tn$. For each test case, the number of detected faults represented by $F1, F2, F3, \ldots, Fn$ and the execution time (in seconds) is provided.

The first dataset is taken from [20], consisting of 8 test cases and 10 detected faults. Table 2 summarizes the test cases, including their detected faults and execution times for this test suite.

J. of Soft Computing and Data Mining Vol. 7 No. 2 (2026) p. 1-16

Table 2 *Case Study One*

Detected Faults	Test Cases							
	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
F1	/		/	/			/	
F2					/	/		/
F3	/		/				/	
F4				/	/			/
F5	/	/	/			/		/
F6	/		/	/			/	
F7		/				/		
F8		/		/	/			/
F9			/	/				
F10	/				/	/	/	
Total Detected Faults	5	3	5	5	4	4	4	4

The second dataset is taken from [21], consisting of 15 test cases and 15 detected faults. Table 3 provides a summary of the test cases, including the detected faults and execution times for this dataset.

Table 3 *Case Study Two*

Detected Faults	Test Cases														
	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11	TC12	TC13	TC14	TC15
F1	/		/		/	/			/					/	
F2		/					/			/			/		/
F3	/					/		/	/			/			
F4							/	/		/	/		/		/
F5											/				
F6				/	/		/	/		/		/	/		
F7		/												/	
F8			/	/						/		/			
F9	/	/							/						
F10											/				
F11				/	/			/		/			/		
F12		/									/				
F13			/			/						/			/
F14					/					/					
F15	/														
Total Detected Faults	4	4	3	3	4	3	3	4	3	6	4	4	4	2	3

3.3 Implementation

The third phase involves implementing the WOA and PSO algorithms on both datasets. The implementation was conducted using Google Colaboratory (Colab) on an Acer Nitro laptop with an Intel Core i5 10th Gen processor.

3.3.1 Implementation Using WOA

WOA begins by randomly initializing the whale population within the search space. Each whale's fitness is then evaluated to identify the current best solution, known as the leader. The algorithm proceeds through three main phases. The first phase involves updating each whale's position relative to the leader's, a process known as the encircling mechanism, where whales move closer to the leader. In the second phase, the position is refined using the bubble-net attacking strategy, which simulates the whale's spiral movement toward the target. This phase typically continues until a specified number of iterations is reached. The final phase is the exploration stage, where random searches are conducted to explore new areas of the search space. At each step, the algorithm ensures that all whale positions remain within the defined boundaries and if a whale discovers a better solution, the leader will be updated accordingly. This process is repetitive until the optimal solution is found or the maximum number of

iterations is reached. In this study, the algorithm executes for up to 100 iterations, using 8 whales for Case Study One and 15 whales for Case Study Two.

3.3.2 Implementation Using PSO

PSO follows an iterative process, but with different mechanisms. It starts by generating a swarm of particles, each with a random initial position. As the algorithm progresses, each particle updates its position and velocity based on its own experience and the position of the global best solution found by the swarm. If a particle meets the stopping criteria, such as convergence or maximum iteration, it records its final path and performance. Otherwise, its position and velocity continue to be updated relative to the global best until the termination condition is met. For consistency with the WOA setup, PSO also uses 8 particles for Case Study One and 15 particles for Case Study Two, with a maximum of 100 iterations.

3.4 Analysis of Result

The last phase involves the evaluation of results after running the algorithms. Two performance metrics will be assessed in this research, which are APFD and execution time. APFD calculates the fault detection rate for each algorithm where the higher the value of APFD, the better the fault detection rate. It will be calculated as in Equation 8.

$$APFD = 1 - \frac{TC1 + TC2 + \cdots + TCm}{nm} + \frac{1}{2n} \tag{8}$$

where TCm indicates the position of the first test case that exposed the fault first while n is the total number of the test cases and m is the total number of detected faults.

The second performance metric that will be assessed is execution time. It measures the time taken to run the algorithm and the one with the lowest time will be classified as a better algorithm. The unit for the time measured is seconds (s). The calculation is as in Equation 9,

$$\text{Execution time (s)} = \text{End time (s)} - \text{Start time(s)} \tag{9}$$

Big-O notation will be the third assessed performance metric. It measures the algorithm's worst-case time complexity in an algebraic term [46]. The article also indicated that assessing the time and memory needs of algorithms is a standard method for determining which algorithm is more effective in solving a specific problem. Big-O notation encompasses several classifications which are constant time ($O(1)$), logarithmic ($O(log\ n)$), linear ($O(n)$), log-linear ($O(n\ log\ n)$), quadratic ($O(n^2)$), polynomial ($O(np)$), and factorial ($O(n!)$).

4. Results and Discussion

For this case study, PSO took a shorter amount of time to execute compared with WOA, according to Figures 4 and 5. Figure 4 indicates the new test case order after implementing WOA which TC6 – TC7 – TC5 – TC3 – TC2 – TC1 – TC4 – TC8 meanwhile after implementing PSO in Figure 5, the order is as follows, TC5 – TC2 - TC8 – TC4 – TC6 – TC1 – TC3 – TC7 where both iterations are the same which are 100 iterations. A comparative analysis of Figures 4 and 5 shows significant differences in fault detection among the various test cases. In Figure 4, TC6 identified faults F2, F5, F7, and F10, whereas TC7 found F1, F3, and F6. TC5 was tasked with discovering F4 and F8, and TC3 uncovered F9. In contrast, Figure 5 shows that TC5 achieved wider fault coverage by detecting F2, F4, F8, and F10, while TC2 recognized F5 and F7. Faults F1, F6, and F9 were detected by TC4, and F3 was detected by TC1.

```
Prioritized Test Case Order: [6, 7, 5, 3, 2, 1, 4, 8]
Execution Time: 0.111583 seconds
```

Fig. 4 *WOA result*

```
Prioritized Test Case Order: [5, 2, 8, 4, 6, 1, 3, 7]
Execution Time: 0.026367 seconds
```

Fig. 5 *PSO result*

Similar results were obtained for case study two, where PSO took the shortest period of time to complete the iterations. It can be proved using Figures 6 and 7 below. Using WOA in Figure 6, the order begins with TC13 followed by TC2 – TC3 – TC14 – TC8 – TC10 – TC11 – TC4 – TC15 – TC7 – TC1 – TC12 – TC5 – TC9 and ended with TC6. As for Figure 7 which by using PSO, the order begins with TC12 followed by TC11 – TC15 – TC4 – TC13 – TC1 – TC9 – TC10 – TC14 – TC7 – TC2 – TC5 – TC6 – TC8 and ended with TC3. Figure 6 indicated that TC13 initiated the fault detection process by recognizing F2, F4, F6, and F11, with TC2 following closely behind by identifying F7, F9, and F12. TC3 was tasked with uncovering F1, F8, and F13, while TC8 was in charge of detecting F3. Moreover,

J. of Soft Computing and Data Mining Vol. 7 No. 2 (2026) p. 1-16

F14 was recognized by TC10, and F15 was detected by TC1. TC11 played a role in identifying both F5 and F10. In contrast, Figure 7 presents a different pattern of fault detection. The detection process began with TC12, which found F3, F6, F8, and F13. TC11 identified F4, F5, F10, and F12, indicating a relatively broad coverage of faults. TC15 solely detected F2, whereas TC4 was responsible for detecting F1. F7 was recognized by TC14, and F14 was noted by TC10. Finally, TC1 was involved in the detection of F1, F9, and F15.

Fig. 6 *WOA result*

Fig. 7 *PSO result*

Using Equation 8 in Subsection 3.4, the APFD was computed to evaluate the performance of both algorithms; the results are shown in Figure 8 for Case Study One and Figure 9 for Case Study Two. In comparison to the PSO algorithm, which produced a lower APFD value of 0.738, the WOA demonstrated superior fault detection performance for Case Study One, with an APFD value of 0.813. However, Case Study Two showed an opposite trend, with PSO achieving a slightly higher value of 0.793 and WOA's APFD slightly declining to 0.789.

These findings suggest that WOA exhibits stronger performance in case studies with characteristics similar to Case Study One, highlighting its potential in determining an optimal test case execution sequence. However, the improved result of PSO in Case Study Two indicates that algorithm effectiveness may vary depending on the dataset size, structure, or complexity. Therefore, while WOA showed superior fault detection capability in the first dataset, PSO's modest advantage in the second dataset reflects the importance of context-specific algorithm selection in TCP tasks.

Overall, WOA performed better in Case Study One than PSO, despite PSO's modest advantage in Case Study Two. Consequently, WOA can be considered as being more effective in figuring out the best sequence for test cases, particularly in settings like those shown in Case Study One. These results imply that no single algorithm consistently outperforms the others across different problem instances. Instead, each algorithm's effectiveness may vary based on the characteristics of the dataset and the optimization landscape of the TCP problem.

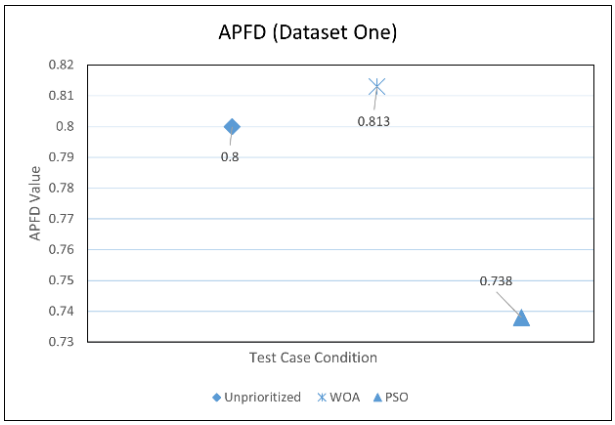

Fig. 8 *Case Study One result*

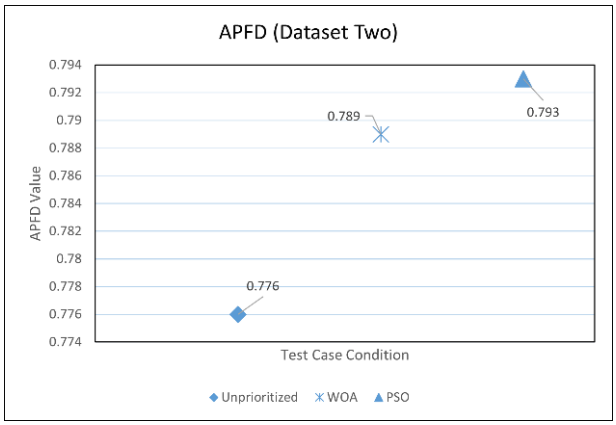

Fig. 9 *Case Study Two result*

The analysis of execution times indicates that Case Study One took less time to process than Case Study Two, mainly because the former contained fewer test cases. In particular, Case Study Two includes seven additional test cases, which leads to a longer execution time. As shown in Table 3, the PSO outperformed the WOA regarding execution time, especially when applied to a larger case study. However, the differences in execution time between PSO and WOA for both case studies are slight, likely due to the relatively small size of the test suites used in this research, which is presented in Figure 10 below. PSO's quicker convergence can be attributed to its simple method of updating positions based on velocity, which improves its runtime efficiency. Nonetheless, because of its limited ability to search locally, PSO is susceptible to premature convergence, which might result in subpar solutions within the discrete search space of TCP. Therefore, even though PSO completes the prioritization task more quickly, the quality of the resulting test case order may be inferior, as evidenced by a lower APFD value compared to algorithms with more effective exploratory capabilities.

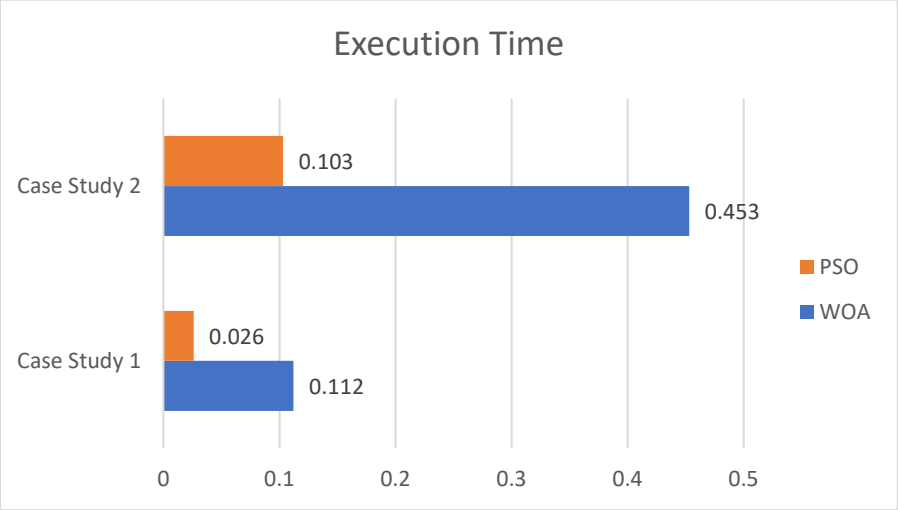

Fig. 10 *Comparison of execution time*

To validate the empirical findings, a computational complexity analysis was conducted using Big O notation. Both algorithms demonstrated an identical complexity of $\boldsymbol{O(n\ log\ n)}$. Given that the observed variations in execution time across both datasets and algorithms were statistically negligible, the results align with theoretical expectations, as the uniform Big O complexity accounts for the comparable performance profiles. The expression for WOA is $\boldsymbol{n + n + 1 + n + n\ log\ n + n}$ which will then be simplified algebraically to $\boldsymbol{n\ log\ n + 4n + 1}$. In the context of asymptotic analysis, the constant term is typically disregarded, reducing the expression to $\boldsymbol{n\ log\ n + n}$. Furthermore, when applying Big-O notation, only the most dominant term is retained; consequently, the complexity is simplified to $\boldsymbol{O(n\ log\ n)}$. In parallel, the computational complexity for the PSO algorithm is initially expressed as $\boldsymbol{n + n + n + n + 1 + n + n + n\ log\ n + n + 1}$, which simplifies to $\boldsymbol{n\ log\ n + 7n + 2}$. Following the principles of asymptotic analysis, constant terms and lower-order coefficients are disregarded, reducing the expression to $\boldsymbol{n\ log\ n + n}$. Ultimately, Big-O notation identifies only the most dominant term, resulting in a complexity of $\boldsymbol{O(n\ log\ n)}$, which aligns with the Big-O classification of the WOA. Based on the experimental findings and literature review, Table 4 presents a summarized comparison of the strengths and weaknesses of both algorithms.

Table 4 *Comparative of both algorithms' strengths and weaknesses*

Criterion	WOA	PSO
APFD	Consistently higher in both case studies	Inconsistent, which is significantly lower than WOA in Case Study 1
Execution Time	Slower due to encircling and spiral mechanisms	Faster due to simple velocity-position updates
Big-O Notation	$O(n\ log\ n)$	$O(n\ log\ n)$
Convergence Behavior	Strong exploration and exploitation behavior	Fast convergence
Parameter Sensitivity	Fewer control parameters	Requires inertia and acceleration tuning
Fitness for Large Datasets	Efficient when fault detection is the top priority	Efficient when time is critical

According to Table 4, PSO exhibits superior execution efficiency, which qualifies it for testing situations with time constraints. In contrast, WOA consistently achieves higher APFD values, suggesting greater robustness in fault detection. Despite having the same asymptotic complexity, both algorithms perform differently when implemented because of different search techniques. These advantageous strengths imply that testing objectives such as fault detection efficacy or execution efficiency ought to influence algorithm selection.

5. Conclusion

To sum up, WOA and PSO each offer unique advantages in solving the TCP issue. WOA is better at setting test case priorities and achieving higher APFD values, whereas PSO is more efficient within the context of execution time.

With the creation of newly prioritized test case sequences for each dataset, the first objective in applying WOA and PSO to two datasets of varying sizes was accomplished. Comparing the efficiency and effectiveness of the two algorithms was the second objective, and it was also accomplished. With an APFD difference of 0.075, Figure 8's results demonstrate that WOA performed better than PSO in Case Study One. Both improved in Case Study Two, with PSO slightly outperforming WOA by 0.004. Table 3 demonstrates that PSO performed more quickly in both settings, with Case Study One executing faster because of its smaller size. From a practical perspective, PSO is superior when execution time is the top concern, where immediate response is essential, whereas WOA is better suited for improving fault detection to reduce debugging and testing costs. Depending on the testing objectives, both algorithms have significant advantages. Nonetheless, each algorithm has its own restrictions, which prompts further investigation into hybrid methods or optimized algorithms paired with larger datasets. One example of employing hybrid methods is the combination of these algorithms. Besides, in practical software testing, such hybrid approaches might potentially be helpful to balance the efficacy of fault detection and execution efficiency, especially in large-scale or dynamic systems. Additionally, incorporating the traveling salesman problem can further improve the APFD and TCP execution times when using the same algorithms, since TCP lacks distance metrics between test cases. Future research might also explore alternative algorithms, such as FA, or examine current algorithms across different RT techniques to evaluate their versatility and effectiveness in diverse testing contexts.

Acknowledgement

Communication of this research is made possible through monetary assistance by Universiti Tun Hussein Onn Malaysia and the UTHM Publisher's Office via Publication Fund E15216.

Conflict of Interest

We declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

We confirm the contributions to the paper as follows: ***study conception and design; data collection; analysis and interpretation of results; and draft manuscript preparation:*** *Nur Atiqah Wahidah binti Sulaiman. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Kurmaku, T., & Kumrija, M. (2020). A Systematic Literature Review and Meta-Analysis Comparing Automated Test Generation and Manual Testing.

[2] Gadwal, A. S., & Prasad, L. (2020). Comparative review of the literature of automated testing tools. ResearchGate, 10,

[3] Khaleel, S. I., & Anan, R. (2023). A Review Paper: Optimal Test Cases for Regression Testing using Artificial Intelligent Techniques. *International Journal of Electrical and Computer Engineering (IJECE),* https://doi.org/10.11591/ijece.v13i2.

[4] Naheed, M., Nadeem, A., & Zaman, Q. u. (2022). A Requirement Based Approach to Test Case Prioritization for Regression Testing. 2022 *19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. http://doi.org/10.1109/IBCAST54850.2022.9990042.

[5] Nandal, D. (2022). Metaheuristics for software test case optimization, Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT). *IEEE.* https://doi.org/10.1109/CCiCT56684.2022.00091.

[6] Wang, T., Noori, M., Altabey, W. A., Wu, Z., Ghiasi, R., Kuok, S. C., & Farsangi, E. N. (2023). From model-driven to data-driven: A review of hysteresis modeling in structural and mechanical systems. *Mechanical Systems and Signal Processing, 204, p 110785.* https://doi.org/10.1016/j.ymssp.2023.110785.

[7] Priya, T., & Prasanna, M. (2022). Fault-Based Test Case Prioritization of Regression Testing Using Genetic Algorithm. International Journal of e-Collaboration (IJeC), *18(2), 1-16.* https://doi.org/10.4018/IJeC.304032.

[8] Yang, Y., Wang, L., Cha, N., & Li, H. (2023, June). A test case prioritization based on genetic algorithm with ant colony and reinforcement learning improvement. I*n 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, *1588-1593*. http://doi.org/10.1109/COMPSAC57700.2023.00245.

[9] Eker, E., Kayri, M., Ekinci, S. et al. (2021). A New Fusion of ASO with SA Algorithm and Its Applications to MLP Training and DC Motor Speed Control. Arab J *Sci Eng 46, 3889–3911*. https://doi.org/10.1007/s13369-020-05228-5.

[10] Zhao, W., Zhang, Z., & Wang, L. (2020). Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. *Engineering Applications of Artificial Intelligence, 87, 103300.* https://doi.org/10.1016/j.engappai.2019.103300.

[11] Augad, N. R., & Gutte, V. S. (2023, December). Swarm intelligence for AI problem solving. *In 2023 International Conference on Advanced Computing & Communication Technologies (ICACCTech), 590-596*. IEEE. http://doi.org/10.1109/ICACCTech61146.2023.00101.

[12] Xing, Y., Wang, X., & Shen, Q. (2021). Test case prioritization based on artificial fish school algorithm. Computer Communications, *180, 295-302.* https://doi.org/10.1016/j.comcom.2021.09.014.

[13] Mohan, K. K., & Zainal, N. (2023). Test Case Prioritization Using Swarm Intelligence Algorithm to Improve Fault Detection and Time for Web Application. *Journal of Soft Computing and Data Mining, 4(2), 59-66.* https://doi.org/10.30880/jscdm.2023.04.02.006.

[14] Lu, C., Zhong, J., Xue, Y., Feng, L., & Zhang, J. (2020). Ant Colony System with Sorting-Based Local Search for Coverage-Based Test Case Prioritization. IEEE. https://doi.org/10.1109/TR.2019.2930358.

[15] Yuan Z., Li J., Yang H., Zhang B. (2021). A Hybrid Whale Optimization and Particle Swarm Optimization Algorithm. 2021 IEEE International Conference on Progress in Informatics and Computing (PIC), *260-264.* https://doi.org/10.1109/PIC53636.2021.9687017.

[16] Chopra, D., & Arora, P. (2022). Swarm intelligence in data science: challenges, opportunities and applications. Procedia computer science, *215, 104-111.* https://doi.org/10.1016/j.procs.2022.12.012.

[17] Chen, J., Rong, H., Zhang, Z., & Luo, R. (2021, May). An adaptive evolutionary whale optimization algorithm. In 2021 33rd Chinese Control and Decision Conference (CCDC), *IEEE, 4610-4614.* https://doi.org/10.1109/CCDC52312.2021.9601898.

[18] Deng, L., Chen, H., Liu, H., Zhang, H., & Zhao, Y. (2021). Overview of UAV path planning algorithms. 2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI), *IEEE, 520-521.* https://doi.org/10.1109/ICETCI53161.2021.9563566.

[19] Yuan, Z., Li, J., Yang, H., & Zhang, B. (2021, December). A hybrid whale optimization and particle swarm optimization algorithm. In 2021 IEEE International Conference on Progress in Informatics and Computing (PIC), *IEEE, 260-264*. https://doi.org/10.1109/PIC53636.2021.9687017.

[20] Nazir, M., Mehmood, A., Aslam, W., Park, Y., Choi, G. S., & Ashraf, I. (2023). A multi-goal particle swarm optimizer for test case prioritization. IEEE Access, 11, 90683-90697. https://doi.org/10.1109/ACCESS.2023.3305973.

[21] Palak, P., & Gulia, P. (2019). Hybrid swarm and GA based approach for software test case selection. International Journal of Electrical and Computer Engineering, 9(6), 4898. https://doi.org/10.11591/ijece.v9i6.pp49898-4903.

[22] Putra, S. J., Sugiarti, Y., Prayoga, B. Y., Samudera, D. W., & Khairani, D. (2023, November). Analysis of Strengths and Weaknesses of Software Testing Strategies: Systematic Literature Review. In 2023 11th International Conference on Cyber and IT Service Management (CITSM) (pp. 1-5). IEEE. https://doi.org/10.1109/CITSM60085.2023.10455226.

[23] Job, M. A. (2021). Automating and optimizing software testing using artificial intelligence techniques. International Journal of Advanced Computer Science and Applications, *12(5)*. https://doi.org/10.14569/IJACSA.2021.0120571.

[24] Zhyhulin, D., Kasian, K., & Kasian, M. (2022, February). Combined method of prioritization and automation of software regression testing. In 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), *IEEE, 751-755.* https://doi.org/10.1109/TCSET55632.2022.9767034.

[25] Yue, L. Z., & Ibrahim, R. (2023). Comparative Analysis for Test Case Prioritization Using Particle Swarm Optimization and Firefly Algorithm. Journal of Soft Computing and Data Mining, *4(2), 67-77*.

[26] Ghani, I., Kadir, W. M. N. W., Arbain, A. F., & Ghani, I. (2024). A Detection-Based Multi-Objective Test Case Selection Algorithm to Improve Time and Efficiency in Regression Testing. IEEE Access, *12, 114974-114994*. https://doi.org/10.1109/ACCESS.2024.3435678.

[27] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., Hamed, H. N. A., & Suffian, M. D. M. (2019). Test case prioritization using firefly algorithm for software testing. IEEE access, *7, 132360-132373*. https://doi.org/10.1109/ACCESS.2019.2940620.

[28] Nayak, S., Kumar, C., Tripathi, S., Mohanty, N., & Baral, V. (2021). Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base. Soft Computing, 25(15), 9925-9942. https://doi.org/10.1007/s00500-020-05428-z.

[29] Ariffin, M. A., Ibrahim, R., Ibrahim, I. S., & Wahab, J. A. (2022). Test Cases Prioritization Using Ant Colony Optimization and Firefly Algorithm. International Journal of Engineering Trends and Technology. https://doi.org/10.14445/22315381/IJETT-V70I3P203.

[30] Alkawaz, M. H., Silvarajoo, A., Mohammad, O. F., & Raya, L. (2022). A System for Optimizing Software Regression Test Cases using Modified Ant Colony Optimization Algorithm. 2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA).

[31] Mugilan A., Totla, T., Renwa, Y., R, C., Subbiah, S., Dharmaraj, T. B., & Prakash, S. O. (2022). Comparative Analysis of Ant Colony Optimization and Particle Swarm Optimization for Test Case Prioritization. 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), (p. 680).

[32] Bajaj, A., Sangwan, O. P., & Abraham, A. (2022). Improved novel bat algorithm for test case prioritization and minimization. Soft Computing, *26(22), 12393-12419*. https://doi.org/10.1007/s00500-022-07121-9.

[33] Singhal, S., Jatana, N., Subahi, A. F., Gupta, C., Khalaf, O. I., & Alotaibi, Y. (2023). Fault Coverage-Based Test Case Prioritization and Selection Using African Buffalo Optimization. Computers, Materials & Continua, *74(3)*. http://doi.org/10.32604/cmc.2023.032308.

[34] Nadimi-Shahraki, M. H., Zamani, H., Asghari Varzaneh, Z., & Mirjalili, S. (2023). A systematic review of the whale optimization algorithm: theoretical foundation, improvements, and hybridizations. Archives of Computational Methods in Engineering, *30(7), 4113-4159*. https://doi.org/10.1007/s11831-023-09928-7.

[35] Alkawaz, M. H., Silvarajoo, A., Mohammad, O. F., & Raya, L. (2022). A System for Optimizing Software Regression Test Cases using Modified Ant Colony Optimization Algorithm. 2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA).

[36] Qu, S., Liu, H., Xu, Y., Wang, L., Liu, Y., Zhang, L., ... & Li, Z. (2024). Application of spiral enhanced whale optimization algorithm in solving optimization problems. Scientific Reports, *14(1), 24534*. https://doi.org/10.1038/s41598-024-74881-9.

[37] Attaallah, A., al-Sulbi, K., Alasiry, A., Marzougui, M., Khan, M. W., Faizan, M., . . . Pandey, D. (2023). Security Test Case Prioritization through Ant Colony Optimization Algorithm. Computer Systems Science and Engineering.

[38] Ouyang, C., Gong, Y., Zhu, D., & Zhou, C. (2023). Improved Whale Optimization Algorithm Based on Fusion Gravity Balance. Axioms, *12(7), 664*. https://doi.org/10.3390/axioms12070664.

[39] Yab, L. Y., Wahid, N., & Hamid, R. A. (2024). Impact of balanced exploration and exploitation on high-dimensional feature selection with hierarchical whale optimisation algorithm. Journal of Information and Communication Technology, *23(4), 593-626*. https://doi.org/10.32890/jict2024.23.4.2.

[40] Tang, J., Liu, G., & Pan, Q. (2021). A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends. IEEE/CAA Journal of Automatica Sinica, *8(10), 1627-1643.* https://doi.org/10.1109/JAS.2021.1004129.

[41] Xu, J., Xu, S., Zhang, L., Zhou, C., & Han, Z. (2023). A particle swarm optimization algorithm based on diversity-driven fusion of opposing phase selection strategies. Complex & Intelligent Systems, *9(6), 6611-6643*. https://doi.org/10.1007/s40747-023-01069-5.

[42] Samad, A., Mahdin, H., Kazmi, R., & Ibrahim, R. (2021). Regression test case prioritization: a systematic literature review. International Journal of Advanced Computer Science and Applications, 12(2).655 - 663.

[43] Kumari, S., Jindal, S., & Sharma, A. (2025). Test case optimization using grey wolf algorithm. Software Quality Journal, *33(2), 20*. https://doi.org/10.1007/s11219-025-09717-4.

[44] Assiri, M. (2025). Test case prioritization using dragon boat optimization for software quality testing. Electronics, *14(8), 1524*. https://doi.org/10.3390/electronics14081524.

[45] Yang, B., Li, H., Xing, Y., Zeng, F., Qian, C., Shen, Y., & Wang, J. (2023). Directed search based on improved whale optimization algorithm for test case prioritization. International Journal of Computers Communications & Control, 18(2). https://doi.org/10.15837/ijccc.2023.2.5049.

[46] Phalke, S., Vaidya, Y., & Metkar, S. (2022, August). Big-O time complexity analysis of algorithm. In 2022 international conference on signal and information processing (IConSIP) (pp. 1-5). IEEE. doi: 10.1109/ICoNSIP49665.2022.10007469.