Implementation of Sand cat Swarm Optimization for Uniform T-Way Test Suite Generation

Submitted: 10th June 2025; accepted: 24th November 2025

Muhammad Aiman bin Mohd Asyraf, Rozmie Razif Bin Othman, Mohd Zamri Bin Zahir Ahmad, Ahmad Ashraf Abdul Halim, Kentaro Go, Nuraminah binti Ramli, R. Badlishah Ahmad, Latifah Munirah Kamarudin, Murad Muhammad Hasan Salih Al-Walidi

DOI: 10.14313/jamris-2026-028

Abstract:
T-way combinatorial testing is an essential approach for optimizing test suite generation by systematically covering parameter interactions while minimizing test cases. Various metaheuristic strategies have been introduced to improve test suite generation, with an increasing focus on balancing exploration and exploitation for efficient test selection. This study investigates the sand cat swarm optimization (SCSO) algorithm as a metaheuristic strategy for t-way test suite generation. Inspired by the hunting behavior of sand cats, SCSO dynamically adjusts sensitivity factors to improve test suite generation efficiency. To evaluate SCSO's performance, 30 benchmark experiments were conducted across four groups of t-way configurations, with t varying from 2 to 6 and v ranging from 2 to 10. Each configuration was executed five times, and the smallest test suite size was selected for analysis. Experimental results demonstrate that SCSO outperforms 15.79% of competing strategies, achieves comparable performance in 42.11% of cases, and is outperformed in 42.11% of benchmark comparisons. These findings highlight SCSO's capability of generating competitive test suites, particularly in t-way interaction testing. The statistical evaluations, including Wilcoxon Rank and Friedman Mean Rank tests, further validate SCSO's performance in comparison to other metaheuristic approaches. Although SCSO effectively reduces test suite size while maintaining interaction coverage, further enhancements are necessary to improve its adaptability and computational efficiency across diverse configurations. Future work should focus on refining SCSO's exploration mechanisms to optimize search efficiency and extend its applicability in combinatorial test generation.

Keywords: *T-way combinatorial testing, test suite generation, sand cat swarm optimization (SCSO), metaheuristic algorithm*

1. Introduction

Software systems are deeply embedded in various domains, including finance, health care, aviation, and critical infrastructure, making their reliability a fundamental concern. Demand for software applications has raised the bar for developed software quality assurance [1]. Software products must be dependable and high quality to fulfill every customer's expectations and requirements [2]. A single software failure can lead to catastrophic consequences, ranging from financial losses and security breaches to safety risks and operational disruptions. Software testing serves as a quality assurance mechanism, systematically identifying defects and vulnerabilities that could compromise system integrity. With an estimated 40% of total development expenses, software testing is an especially expensive portion of the software development process [3]. However, as modern software applications become increasingly complex, traditional method or testing approaches face significant challenges in achieving comprehensive validation within practical time and resource constraints [4].

Ensuring software reliability requires efficient test design strategies that balance coverage and efficiency. Interaction-related faults are among the crucial errors that must be disclosed because any breakdown in these interactions would impact main functionalities of the system [5]. Exhaustive testing is impractical due to the exponential growth of test cases, making combinatorial testing a preferred approach for systematically covering parameter interactions because selecting at least one test case must be covered for each t-way (t is the interaction strength) combination of input parameters of a configuration system that cover all t-way interactions among input parameters [6, 7]. The process of combinatorial testing involves defining the input space (sampling the input configuration), constructing optimized test cases that ensure interaction coverage, and executing the test suite to detect faults [8].

Generating optimal combinatorial test suites remains computationally challenging, making difficult combinatorial testing a challenge and a significant research limitation [9]. Metaheuristic algorithms, such as genetic algorithms (GAs), particle swarm optimization (PSO), ant colony optimization (ACO), and whale optimization algorithm (WOAs) have been applied to optimize test suite size and address the minimum covering array generation (MCAG) problem [10]. However, the nondeterministic polynomial (NP) hard nature of test suite optimization, where no single strategy can guarantee that it always produces the best test suite size for all configurations, presents challenges for existing algorithms, creating opportunities for

new metaheuristic approaches to improve efficiency and coverage [11]. Current methods struggle with selective pressure, premature convergence, inefficient exploration, and high computational costs, limiting their effectiveness in large-scale configurations [12].

Nonetheless, according to the no free lunch (NFL) theorem, no single optimization algorithm can solve all optimization domains [13]. This suggests that an algorithm's performance is highly dependent on the nature of the problem, making the integration of metaheuristic algorithms into specific applications, such as t-way combinatorial test generation, an ongoing research challenge. Motivated by this, the present study explores the application of SCSO in t-way testing.

SCSO is a recently introduced metaheuristic inspired by the adaptive hunting behavior of sand cats, which attack or search for prey according to the sound frequency [14]. Unlike traditional swarm-based optimization techniques, SCSO incorporates a stochastic movement strategy that dynamically adjusts its search behavior, effectively balancing exploration and exploitation. It has demonstrated strong performance in finding optimal solutions with fewer parameters and reduced computational overhead [15]. This adaptability makes it particularly suitable for optimization problems that require efficient navigation of large search spaces. While SCSO has been successfully applied to various optimization domains, its potential for combinatorial test suite generation remains unexplored. Given its unique search mechanism, SCSO could serve as an alternative approach for minimizing test suite size while ensuring comprehensive parameter interaction coverage.

Motivated by its reported advantages, this study aims to evaluate the performance of SCSO in t-way combinatorial test suite generation and compare it against existing metaheuristic-based approaches. The remainder of this paper is structured as follows. Section 2 presents a comprehensive review of combinatorial testing strategies and metaheuristic optimization techniques. Section 3 details the methodology, including the adaptation of SCSO for test suite generation. Section 4 discusses the experimental setup, data sets, and evaluation criteria. Section 5 presents the results and analysis, followed by Section 6, which concludes the study and outlines potential future research directions.

2. Combinatorial T-way Testing Background

Software testing plays a crucial role in ensuring the quality and reliability of modern software systems [16]. As applications become increasingly complex, the number of configurable parameters expands, leading to an exponential growth in possible input combinations. Exhaustive testing (a method that tests all combinations), while theoretically ensuring complete accuracy, becomes impractical due to excessive computational costs and time constraints, as the combinatorial explosion problem makes it unfeasible for a large number of input parameters [12]. To address this challenge, combinatorial testing has emerged as an effective approach, focusing on covering t-way interactions between input parameters, rather than testing every possible combination. This method provides a structured way that effectively reduces the number of test cases needed compared to exhaustive testing while maintaining adequate coverage of parameter interactions [17].

The core principle behind t-way testing is based on empirical observations that show most software failures are caused by the interaction of a small subset of input parameters rather than the entire input space that can reach an incorrect result [18]. Studies have shown that pairwise (2-way) testing can detect a large percentage of defects while increasing the interaction strength (t) enhances fault detection. By systematically covering all possible t-way interactions, combinatorial testing ensures that critical parameter interactions are tested, improving software quality without requiring an infeasible number of test cases. The effectiveness of this approach depends on selecting an appropriate t, where higher values provide better fault detection at the cost of increased test suite size.

Combinatorial test suite generation relies on the construction of covering arrays (CAs), which are mathematical structures that ensure that all t-way combinations appear in at least one test case. A covering array is represented as CA(N; t, k, v), where N denotes the number of test cases, t is the interaction strength, k is the number of parameters, and v represents the possible values each parameter can take [10]. The goal of test suite generation is to minimize N while maintaining full t-way coverage, as a smaller test suite reduces execution time and testing costs. Constructing an optimal CA array is computationally challenging, requiring efficient strategies to generate minimal test suites while maintaining complete interaction coverage.

Exhaustive testing involves evaluating every possible combination of input parameters, which quickly becomes infeasible due to the combinatorial explosion. T-way testing can cover all of the important interaction components at least once, and the size of the test suite is reduced in proportion to the interaction strength, "t" [19]. It can improve the effectiveness of software testing from different configuration systems while simultaneously lowering the anticipated cost and time [20]. It also makes combinatorial testing highly efficient, especially for large-scale software systems where time and cost are major concerns. It allows teams to detect potential issues early, optimize resources, and accelerate product delivery without getting stuck in a never-ending cycle of exhaustive testing [21].

Combinatorial testing can be categorized into two variations based on interaction strength: uniform interaction strength and variable interaction strength [10]. Uniform strength interaction means that all input parameters share the same level of interaction where the CA maintains a consistent interaction strength across all parameters, ensuring uniform

test coverage [22]. Variable strength alone guarantees and has more than one interaction strength for generating test cases [22]. It assigns different t-way levels to parameters based on system criticality. This approach optimizes test coverage without unnecessarily increasing test suite size [10].

Metaheuristic algorithms have gained significant attention in combinatorial testing due to their ability to explore large solution spaces and optimize test suite size while ensuring full coverage. It is because it has capable to escape from local optima and perform a robust search of a search space [23]. Algorithms such as GA, ACO, and WOA leverage heuristic-based search techniques are able to find near-optimal test suites under several conditions which are balancing exploration (global search) and exploitation (local refinement) to minimize the number of test cases. These methods have proven particularly useful in large-scale and highly configurable software systems, whereas traditional techniques struggle to provide efficient solutions.

As combinatorial testing continues to evolve, its integration with optimization techniques remains a key area of research. The challenge of generating optimal test suites efficiently requires a balance between computational feasibility and maximizing interaction coverage. Metaheuristic approaches provide promising solutions to this problem by leveraging intelligent search mechanisms to minimize test suite size while maintaining high defect detection efficiency. It can exploit stochastic behavior to search for the best test cases through only several iterations [17]. With the growing complexity of modern software systems, combinatorial testing is becoming an indispensable methodology in ensuring robust and efficient software validation.

2.1. Problem Definition Model

Figure 1 illustrates the concept of t-way testing using an "online learning system" application. This system filters learning content based on five parameters: user type, device used, level course difficulty, content format, and interaction mode. The user type is categorized into two groups: student and teacher. The device used is classified as mobile phone or laptop, while the course difficulty falls into three levels: beginner, intermediate, and advanced. The content format is video and text, and the interaction mode is categorized as self-paced, live, and hybrid. To simplify the explanation, each parameter is assigned a distinct notation (e.g., U1 and U2 for user type, D1 and D2 for device used, L1, L2, and L3 for level course difficulty). Table 1 provides a simplified representation of the parameters and their possible values for this system.

To test such a system, exhaustive testing, which involves generating all possible combinations of parameter interactions to achieve complete coverage, could be applied. For the given system, full-strength interaction testing (where t = 5) results in 72 test cases (2 × 3 × 2 × 3 × 2 = 72). While exhaustive testing theoretically ensures complete accuracy,

Figure 1. Filter for online learning system

Table 1. Classification Representation of Input Parameters

Input parameter	U	L	D	M	C
Possible Value	U1	L1	D1	M1	C1
	U2	L2	D2	M2	C2
		L3		M3	

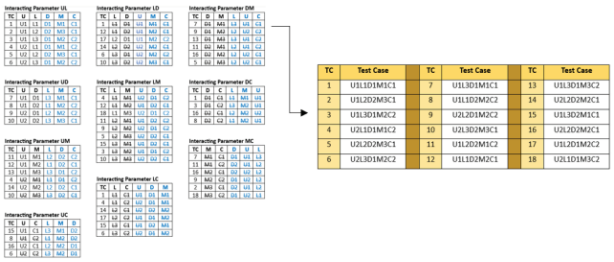
TC	Test Case	TC	Test Case	TC	Test Case	TC	Test Case
1	U1L1D1M1C1	19	U1L2D2M1C1	37	U2L1D1M1C1	55	U2L2D2M1C1
2	U1L1D1M1C2	20	U1L2D2M1C2	38	U2L1D1M1C2	56	U2L2D2M1C2
3	U1L1D1M2C1	21	U1L2D2M2C1	39	U2L1D1M2C1	57	U2L2D2M2C1
4	U1L1D1M2C2	22	U1L2D2M2C2	40	U2L1D1M2C2	58	U2L2D2M2C2
5	U1L1D1M3C1	23	U1L2D2M3C1	41	U2L1D1M3C1	59	U2L2D2M3C1
6	U1L1D1M3C2	24	U1L2D2M3C2	42	U2L1D1M3C2	60	U2L2D2M3C2
7	U1L1D2M1C1	25	U1L3D1M1C1	43	U2L1D2M1C1	61	U2L3D1M1C1
8	U1L1D2M1C2	26	U1L3D1M1C2	44	U2L1D2M1C2	62	U2L3D1M1C2
9	U1L1D2M2C1	27	U1L3D1M2C1	45	U2L1D2M2C1	63	U2L3D1M2C1
10	U1L1D2M2C2	28	U1L3D1M2C2	46	U2L1D2M2C2	64	U2L3D1M2C2
11	U1L1D2M3C1	29	U1L3D1M3C1	47	U2L1D2M3C1	65	U2L3D1M3C1
12	U1L1D2M3C2	30	U1L3D1M3C2	48	U2L1D2M3C2	66	U2L3D1M3C2
13	U1L2D1M1C1	31	U1L3D2M1C1	49	U2L2D1M1C1	67	U2L3D2M1C1
14	U1L2D1M1C2	32	U1L3D2M1C2	50	U2L2D1M1C2	68	U2L3D2M1C2
15	U1L2D1M2C1	33	U1L3D2M2C1	51	U2L2D1M2C1	69	U2L3D2M2C1
16	U1L2D1M2C2	34	U1L3D2M2C2	52	U2L2D1M2C2	70	U2L3D2M2C2
17	U1L2D1M3C1	35	U1L3D2M3C1	53	U2L2D1M3C1	71	U2L3D2M3C1
18	U1L2D1M3C2	36	U1L3D2M3C2	54	U2L2D1M3C2	72	U2L3D2M3C2

Figure 2. List of generated test cases for exhaustive testing

it becomes impractical when dealing with a large number of input parameters due to the combinatorial explosion problem [12]. Figure 2 shows test cases for exhaustive testing based on the online learning system application.

To address this issue, the number of test cases can be significantly reduced by applying t-way testing with a lower interaction strength (t). For instance, pairwise testing (t = 2) ensures that all pairs of parameter interactions are tested, thereby reducing the number of test cases while maintaining sufficient coverage. T-way testing focuses on testing parameter interactions up to a defined strength (t). For example, 2-way testing ensures that all pairwise interactions are covered, significantly reducing the number of test cases compared to exhaustive testing. Techniques such as uniform strength (pairwise) testing provide flexibility in prioritizing critical interactions.

In pairwise testing (t = 2), only specific pairs of parameters are tested for interactions, while the remaining noninteracting parameters are assigned random valid values (denoted as DX) to form complete test cases. For the given system, the interacting pairs include UL, UD, UM, UC, LD, LM, LC, DM, DC, and MC. Figure 2 demonstrates the construction of test cases using t-way testing with t= 2. Any duplicate test



TC	Test Case	TC	Test Case	TC	Test Case
1	U1L1D1M1C1	7	U1L3D1M1C1	13	U1L3D1M3C2
2	U1L2D2M3C1	8	U1L1D2M2C2	14	U2L2D2M2C1
3	U1L3D1M2C2	9	U2L2D1M2C2	15	U1L3D2M1C1
4	U2L1D1M1C2	10	U2L3D2M3C1	16	U2L2D2M2C1
5	U2L2D2M3C1	11	U1L2D2M1C2	17	U1L2D1M2C2
6	U2L3D1M2C2	12	U1L1D2M2C1	18	U2L1D1M3C2

Figure 3. Test case generation for t = 2 and list of test cases for t-way

cases are removed, and only unique combinations are retained as final test cases.

As shown in Figure 3, t-way testing significantly reduces the number of test cases from 72 (in exhaustive testing) to 18, achieving a reduction of over 75%. Despite this reduction, the test cases maintain adequate coverage of parameter interactions, ensuring effective fault detection. This approach not only reduces the number of test cases but also minimizes resource consumption, such as time and cost, making the testing process more efficient.

3. Related Work

Research in t-way combinatorial testing has significantly evolved, transitioning from traditional computational approaches to modern metaheuristic methods. Computational approaches, rooted in algebraic techniques, have been fundamental in t-way test suite generation. These methods employ heuristic rules and greedy strategies to cover uncovered test combinations [24]. However, while computational methods offer flexibility and support for complex configurations, they struggle with efficiency, as longer computational times are required to handle extensive test combinations [25]. This limitation highlights the need for more scalable solutions. To address these challenges, metaheuristic algorithms have emerged as an effective alternative for t-way test suite generation. Metaheuristics typically begin with a random solution and iteratively apply search techniques to improve fitness. Though not perfectly accurate, these methods efficiently produce near-optimal solutions in less execution time than computational methods. Metaheuristic algorithms enable a more robust and adaptive learning process, ultimately enhancing the network's ability to solve various challenging problems involving t-way interactions [26].

Within this context, t-way testing strategies are broadly categorized into two fundamental approaches: one-test-at-a-time (OTAT) and one-parameter-at-a-time (OPAT). OTAT starts with an empty test suite, where test cases are added one by one until all interactions are covered. Whenever a test case is chosen, it is included in the final suite (vertical extension). OTAT strategies have been extensively explored in research due to their ease of implementation and ability to generate effective test suites efficiently [19]. On the other hand, OPAT begins with an initial test suite and progressively adds parameters one at a time until all parameters have been incorporated. This method is referred to as horizontal extension, where test cases evolve gradually with additional parameters. After completing the horizontal extension, further test cases may need to be added through vertical extension to ensure full interaction coverage. OPAT provides an alternative approach to test suite generation but requires more complex handling of parameter dependencies, making it less commonly adopted in research [19].

Due to its structured nature and incremental test case construction, OTAT is often considered a more convenient and practical approach for t-way testing. Its systematic process simplifies test generation and reduces computational complexity, making it easier to implement compared to OPAT. As a result, OTAT strategies have gained more attention as one of the most promising research areas in t-way combinatorial testing [10].

Among the early computational methods developed for t-way interaction testing was the Test Configuration Generator (TConfig), introduced by A. W. Williams in the late 1990s [27] for web-based interaction testing. It is classified as a computational approach, since it relies on systematic algorithms. TConfig follows the OPAT approach, constructing test configurations in a structured, step-by-step manner rather than generating all possible test combinations at once. The tool primarily employs combinatorial testing techniques, specifically using CAs to ensure that every pair-wise (or higher-order) combination of parameter values is represented efficiently in the test set. It leverages the recursive block algorithm (Williams) and the in-parameter-order (IPO) greedy algorithm (Lei and Tai) to construct these test configurations, focusing on minimizing test cases while maintaining high interaction coverage.

In 1998, Lei and Tai introduced the IPO strategy [28], which was a groundbreaking computational method for pairwise interactions. It also applies the OPAT approach. Enhancements such as in-parameter-order-general (IPOG) and IPOG-D extended its capabilities to support variable-strength testing up to t = 6, optimizing test suite generation through horizontal and vertical extensions [29, 30]. Subsequent variants like IPOG-F2 and SCIPOG integrated lightweight heuristics and advanced constraint-handling techniques, ensuring efficient and precise test case generation for modern systems [20, 31].

At the beginning of the new decade, in 2010, the test vector generator (TVG), developed by Arshem, was introduced as a public domain tool for generating GUI tests using computational t-way interaction testing. It supports three interaction strength types: input-output relationship (IOR), variable strength, and uniform strength. Although it is claimed that TVG can support up to six strengths, practical execution has only achieved up to five [32]. TVG employs a greedy method and OTAT for test case generation and utilizes

three algorithms: t-reduced, plus-one, and random set. Among these, t-reduced produces the most optimized test suites, though limited details are available on the workings of each algorithm.

In contrast to these computational approaches, metaheuristic-based t-way interaction testing strategies have gained significant attention and have been widely explored in recent years.

Among the earliest breakthroughs, the particle swarm test generator (PSTG), introduced by Ahmed and Zamli [33] in 2010, utilized PSO as a metaheuristic method for generating t-way combinatorial test suites. Initially designed for uniform interaction strengths, PSTG was improved by the same authors in 2011 [34] to support variable-strength interaction as well. PSTG follows an OTAT approach, incrementally generating test cases to optimally cover interactions through particle swarm operations. PSTG effectively balances global search (exploration) and local search (exploitation) to produce compact test suites. Advantages include simpler implementation, fewer parameters to tune compared to other metaheuristics, and strong performance in generating smaller test sets. However, PSTG may experience higher computational overhead from repeated particle evaluations and typically requires careful parameter adjustments to consistently perform well across different test scenarios.

A year later, in 2011, the harmony search strategy (HSS) [35] was introduced by Alsewari and Zamli as a metaheuristic approach specifically designed for generating t-way test suites. It employs an OTAT method, incrementally building individual test cases to optimally cover interactions. Initially supporting only uniform interaction strengths, HSS was enhanced in 2012 to include support for variable-strength interactions. Its search mechanism balances exploration and exploitation effectively using harmony memory operations. The advantages of HSS include producing comparatively compact test suites, simple implementation, and minimal parameter tuning. Nevertheless, it may incur higher computational costs due to frequent harmony evaluations and can require careful parameter calibration to maintain optimal results across various testing configurations.

Subsequently, in 2015, the cuckoo search (CS) strategy for combinatorial test suite generation was introduced by Ahmed et al. [36]. It is a metaheuristic approach employing the OTAT method, where each iteration produces one complete test case optimized through a stochastic global search. CS leverages Lévy flights to efficiently balance exploration (global search) and exploitation (local search), allowing it to effectively generate combinatorial test suites. The strategy supports both uniform and mixed interaction strengths, making it suitable for variable-strength combinatorial testing. Key advantages include minimal parameter tuning, robustness in escaping local optima, and competitive performance with simpler tuning requirements compared to other metaheuristics like GA and PSO. Nevertheless, CS can incur higher computational overhead due to the iterative Lévy flight operations, particularly for complex or large-scale test generation scenarios, and it may still experience issues with convergence speed in some contexts.

In the same year, the swarm intelligent test generator (SITG) was introduced by Rabbi et al. [37]. The strategy applies PSO to generate t-way test suites by treating each particle as a complete test case rather than a numerical solution. It follows the OTAT approach, where test cases are added incrementally to maximize interaction coverage. The process begins with a randomly initialized swarm, where each particle represents a candidate test case. Instead of using traditional PSO velocity updates, SITG evaluates each test case based on its ability to cover previously uncovered t-way interactions, adjusting positions accordingly. The best test cases are iteratively selected and added to the final test suite (FTS), ensuring that all required interactions are covered. SITG supports both uniform and variable-strength interactions, with testing conducted up to t = 6. The approach efficiently balances exploration and exploitation, leading to compact test suites that effectively cover interactions, especially for higher-strength scenarios ($t \geq 4$). However, it introduces computational overhead due to frequent velocity evaluations and requires careful parameter tuning to prevent premature convergence to suboptimal solutions. Despite these challenges, SITG remains an effective and adaptive strategy for combinatorial test case generation.

In 2016, the high-level hyper-heuristic (HHH) strategy using tabu search was introduced by Zamli et al. [38] for t-way combinational test suite generation. It employs the OTAT approach and explicitly supports both uniform and variable interaction strengths, with reported experimental validations up to t = 6. In its implementation, HHH iteratively generates each test case by dynamically selecting among four low-level metaheuristics: teaching learning-based optimization (TLBO), the global neighborhood algorithm (GNA), PSO, and CS. Differing from conventional single-method strategies, HHH utilizes tabu search to intelligently switch among these metaheuristics using adaptive operators for improvement, diversification, and intensification, effectively balancing exploration and exploitation. Advantages of HHH include flexibility in adapting its search strategy according to problem dynamics, reduced likelihood of becoming trapped in local optima, and improved test suite compactness. However, employing multiple metaheuristics through tabu search significantly increases algorithm complexity, computational overhead, and sensitivity to parameter tuning, potentially limiting its practical applicability without careful calibration.

Following this, in 2017, the adaptive teaching-learning-based optimization (ATLBO) method was introduced by Din and Zamli [39] to enhance the conventional TLBO by integrating fuzzy logic for adaptive combinational t-way test suite generation. ATLBO

applies an OTAT approach, incrementally constructing each test case through iterative optimization. In this implementation, candidate solutions represent individual test cases that evolve based on the TLBO mechanism, comprising a global search (teacher phase) and local search (learner phase). Unlike traditional TLBO, ATLBO dynamically selects between these two search phases using a Mamdani fuzzy inference system, which continuously evaluates solution quality, intensification, and diversification measures to decide the optimal search direction at each iteration. ATLBO explicitly supports uniform and variable interaction strengths, with experimental validation reported up to t = 6. Its main advantages over conventional TLBO include increased adaptability, improved convergence speed, and better robustness against premature convergence. Nevertheless, the fuzzy logic integration introduces additional complexity, computational overhead, and the need for carefully designed fuzzy inference rules, potentially complicating practical applications.

Building upon reinforcement learning principles, in 2018, the Q-learning sine cosine algorithm (QLSCA) was introduced by Zamli et al. [40] as a metaheuristic approach specifically tailored for generating combinational t-way test suites using an OTAT methodology. QLSCA constructs test suites by iteratively generating each individual test case through a dynamic search guided by reinforcement learning (Q-learning). In each iteration, candidate solutions are updated by adaptively selecting among four distinct search operations: sine search, cosine search, Lévy flight, and elitism. Unlike traditional SCA, where the search operations are determined through fixed parameters, QLSCA dynamically chooses these operations based on a learned Q-value, promoting balanced exploration and exploitation. The method explicitly supports uniform interaction strength and has been experimentally validated up to t = 4. Advantages of QLSCA include improved adaptability, reduced reliance on manually tuned parameters, and enhanced capability to avoid premature convergence compared to traditional SCA. However, incorporating the reinforcement learning mechanism introduces additional complexity, computational overhead, and increased sensitivity related to the learning parameters, which must be tuned carefully to maintain consistent performance.

In 2019, the artificial bee colony for variable strength (ABCVS) strategy was introduced by Alazzawi et al. as a metaheuristic approach derived from the artificial bee colony (ABC) algorithm for combinatorial t-way test suite generation [41]. ABCVS employs an OTAT approach and explicitly supports both uniform and variable interaction strengths, validated up to t = 6. Unlike conventional ABC, ABCVS enhances the balance between exploration and exploitation by dynamically adjusting the number of employed, onlooker, and scout bees based on test case coverage and diversity. This adaptive mechanism ensures optimal interaction coverage while minimizing test suite size. The advantages of ABCVS include improved optimization efficiency, high scalability, and better adaptability in handling complex t-way interactions. Additionally, its swarm intelligence approach helps prevent premature convergence while maintaining test diversity. However, ABCVS requires careful tuning of parameters such as colony size and search limits, and its iterative nature introduces computational overhead, which may impact execution time in large-scale test configurations.

Continuing this trend, in 2020, the ant colony optimization algorithm using fuzzy logic (ACOF) strategy was introduced by Ahmad et al. [42] as an advanced metaheuristic derived from Dorigo's original ACO. ACOF employs an OTAT approach and supports both uniform and variable interaction strengths, explicitly tested up to t = 6. Unlike traditional ACO, ACOF integrates a Mamdani fuzzy inference system to dynamically adjust two critical parameters: the pseudo-random proportional selection rule and the number of ants utilized per iteration. Specifically, fuzzy logic determines the optimal selection probability and dynamically allocates ant resources, enhancing exploration and exploitation balance. Advantages include generating compact test suites and significantly improving execution time compared to conventional ACO variants, due to its adaptive mechanisms. However, integrating fuzzy logic increases algorithm complexity, requires carefully designed fuzzy rules, and may add computational overhead during the inference process.

In the same year, WOA for t-way test suite generation was introduced by Hassan et al. [43]. This metaheuristic strategy employs the OTAT approach and explicitly supports both uniform and variable-strength interactions, with successful tests conducted for interaction strengths up to t = 6. The WOA method initializes multiple candidate solutions ("whales") within the combinational search space and iteratively updates their positions using whale-inspired mechanisms such as encircling prey (exploitation) and bubble-net attacking (exploration). This adaptive process helps achieve a balanced exploration-exploitation trade-off, enhancing diversity and efficiency in generating optimized test suites. However, the approach faces potential computational overhead due to iterative candidate updates and can experience excessive exploration, which may slow convergence.

Advancing further, in 2021, the gravitational search test generator (GSTG) strategy was introduced by Htay et al. [11], based on the gravitational search algorithm (GSA), which is a population-based metaheuristic inspired by Newtonian gravity. GSTG uses the OTAT approach, iteratively selecting the best test case by mimicking gravitational interactions between candidate solutions, with each candidate represented as a test case. The algorithm uniquely employs gravitational forces to guide less optimal solutions (lighter masses) toward optimal ones (heavier masses), thus effectively balancing exploration and exploitation. GSTG supports both uniform and

variable interaction strengths, explicitly tested up to t = 10. Its advantages include strong exploration capabilities, effective avoidance of local optima due to its gravity-inspired search mechanism, and competitive performance in generating optimal or near-optimal test suites. However, GSTG has disadvantages, such as higher computational complexity, especially with increasing interaction strength or parameters, and sensitivity to parameters like gravitational constants, requiring careful tuning for optimal results. These issues are consistent with typical challenges noted in the literature regarding GSAs.

More recently, in 2022, the improved particle swarm optimization (improved PSO) strategy for t-way test suite generation was introduced by Prasad et al. [44]. It is a metaheuristic method extending traditional PSO, specifically adapted for generating t-way combinatorial test cases. Improved PSO follows the OTAT approach, uniquely simplifying the particle-update mechanism by directly adjusting particle positions based on coverage of uncovered interactions. Unlike conventional PSO, it eliminates reliance on typical parameters such as inertia weight, acceleration coefficients, and complex velocity calculations, significantly reducing the need for extensive parameter tuning. The strategy supports both uniform and variable interaction strengths, explicitly tested up to t = 6. Its advantages include easier implementation, fewer parameters to tune, and notably improved performance and efficiency compared to conventional PSO. However, disadvantages include computational overhead due to iterative particle evaluations and sensitivity to parameter adjustments, potentially causing premature convergence and trapping in local optima, consistent with common challenges highlighted in PSO-related literature.

Most recently, in 2024, the wingsuit flying search (WFS) optimization algorithm for t-way test suite generation was introduced by Rose et al. [45] as a metaheuristic, parameter-free approach inspired by wingsuit flying. WFS employs a unique three-phase implementation: generating initial points via Halton sequences, adaptively adjusting neighborhood sizes, and progressively narrowing the search space through decreasing discretization steps. It follows the OTAT approach, iteratively building each test case. WFS supports both uniform and variable interaction strengths, explicitly tested up to interaction strength t = 10. Its main advantages include ease of use due to its parameter-free nature, efficient performance, and the capability of producing compact test suites without extensive parameter tuning. However, it tends to heavily emphasize exploitation in later stages of optimization, increasing the risk of convergence to local optima, especially in complex test configurations [46].

This continuous evolution from computational methods to modern metaheuristic-based strategies highlights the ongoing advancements in t-way interaction testing techniques, significantly improving efficiency and adaptability over time.

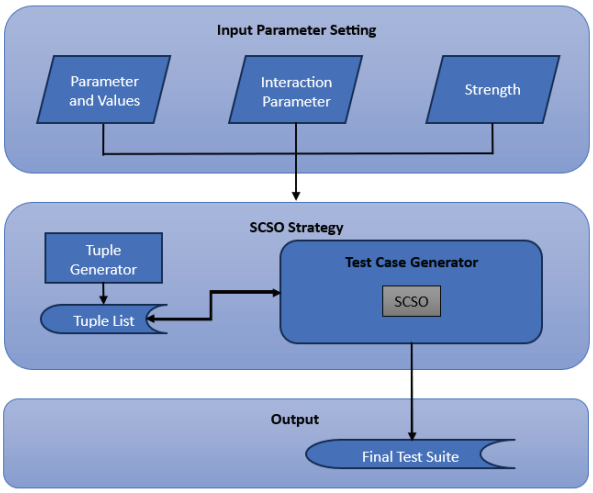


Figure 4. SCSO's framework

Figure 5. Exploration and exploitation of SCSO

4. Proposed Strategy of the Sand Cat Swarm Optimization (SCSO) Algorithm

Figure 4 shows the recommended framework for SCSO's approach. To achieve the SCSO method, a number of components are created and used, including the test case generator (TCG) and tuple generator (TG) where TG is derived from [24, 47]. The framework consists of three main sections. The first section, input parameter setting, defines the test parameters, their values, interaction parameters, and interaction strength, which serve as the foundation for generating test cases. The SCSO framework support only uniform strength, which are implemented using the OTAT test case generation approach. The second section, SCSO strategy, involves the TG, producing a tuple list (TL) representing possible parameter combinations. These tuples are then processed by the TCG, which incorporates the SCSO algorithm to generate optimized test cases. The final section, output, delivers the final test suite, ensuring that the generated test cases comprehensively cover the defined interactions.

SCSO is a bio-inspired optimization algorithm modeled after the hunting and survival strategies of sand cats, called "Felis margarita." The SCSO concept was first introduced by Seyyedabbasi and Kiani to solve optimization problems [15]. This novel approach combines unique behavioral patterns of sand cats to solve optimization problems efficiently. Exploration and exploitation phases based on the hunting behavior of sand cats is shown in Figure 5.

The process begins by initializing the population, determining the population size required for the SCSO to achieve the best outcome. Based on the requirements, a population size of 10 seems sufficient for the iterations, as it can yield the best possible results

efficiently. Furthermore, it will take a long time if the set is more than 10. The associated structure is a vector, and SCSO is thus a population-based technique. Each possible solution is determined by evaluating a predetermined fitness function. The SCSO will determine the optimal values of the parameters based on this function's definition of the problem's parameters.

4.1. Global Sensitivity Factor, $\overrightarrow{r_G}$

$$\overrightarrow{r_G} = SM - \left(\frac{SM \times iter_c}{iter_{Max}}\right) \quad (1)$$

The SCSO algorithm introduces a dynamic global sensitivity factor from Eq. (1) to balance exploration and exploitation effectively. The sensitivity is calculated, where SM = 2 inspired by sand cats' 2 kHz hearing ability. High initial sensitivity ensures broad exploration during early iterations, covering diverse regions of the search space. As iterations progress, the global sensitivity factor decreases gradually, shifting the algorithm's focus to exploitation by refining solutions. Toward the final iterations, it approaches zero, minimizing exploration and emphasizing intensification. This adaptive behavior mirrors sand cats' transitioning from scanning for prey to capturing targets, ensuring efficient navigation through the search space.

4.2. Sensitivity for Each Sand Cat

$$\vec{r} = \overrightarrow{r_G} \times rand(0,1) \quad (2)$$

Each sand cat's individual sensitivity ***r*** is dynamically adjusted using Eq. (2), where ***r*** scales the agent's responsiveness and rand(0,1) introduces randomness. This stochastic variability allows agents to explore diverse regions of the search space effectively while maintaining adaptability. The sensitivity mechanism ensures a balanced search dynamic, enabling agents to avoid local optima while still refining promising solutions. By mimicking sand cats' natural adaptability to environmental cues during hunting, this mechanism enhances the robustness and diversity of the optimization process.

4.3. Decision Parameter (R)

$$R = 2 \times \overrightarrow{r_G} \times rand\ (0,1) - \overrightarrow{r_G} \quad (3)$$

The decision parameter R from Eq. (3) determines whether agents explore new areas (R > 1) or exploit known solutions (R ≤ 1) during each iteration. It is calculated, where $\overrightarrow{r_G}$ is the global sensitivity factor.

If $R > 1$, the agent enters the exploration phase, where it moves to less-visited regions of the search space to promote diversity and avoid premature convergence. In this phase, the position of the agent is updated using the formula below.

$$\vec{r} \cdot (\overrightarrow{Pos_{bc}}(t) - rand\,(0,1) \cdot \overrightarrow{Pos_c}(t)) \quad (4)$$

Here, $\vec{r}$ represents the sensitivity factor that scales the agent's movement, ensuring its adjustments align with the global search dynamics. $\overrightarrow{Pos_{bc}}(t)$ refers to the position of the best candidate solution at iteration t, serving as a reference for guiding exploration. rand(0,1) introduces randomness to the movement, adding stochastic variability, while $\overrightarrow{Pos_c}(t)$ represents the current position of the agent. The subtraction $\overrightarrow{Pos_{bc}}(t) - rand\,(0,1) \cdot \overrightarrow{Pos_c}(t)$ creates a directional vector that leads the agent toward unexplored areas of the search space, with the random term ensuring unpredictable movement patterns. This calculated randomness enables the agent to explore a broader region effectively, increasing the probability of discovering better solutions while avoiding local optima. This formula encapsulates the essence of balancing guidance from the best solutions and randomness, making exploration robust and efficient in SCSO.

On the other hand, if $R \leq 1$, the agent enters the exploitation phase, which focuses on refining its current position by moving toward promising solutions. The agent's position is updated using Eq. (5).

$$\overrightarrow{Pos_b}(t) - \overrightarrow{Pos_{rnd}} cos\,(\theta) \cdot \vec{r} \quad (5)$$

The position update mechanism leverages both deterministic and stochastic components to guide the agent's movement during the search process. Here, $\overrightarrow{Pos_b}(t)$ represents the best-known position at iteration t, serving as a key anchor point for the agent to refine its search toward optimal solutions. $\overrightarrow{Pos_{rnd}}$ introduces an element of randomness by selecting a random position within the search space, ensuring variability in movement. The cosine function, *cos(θ)*, incorporates the angle θ, a randomly chosen value between 0° and 360°, to determine the direction of the agent's movement. This angle is further multiplied by $\vec{r}$, the sensitivity factor, which scales the movement magnitude. Together, these terms ensure that the agent's motion is both guided by the best solutions and randomized to maintain diversity and avoid convergence on suboptimal regions.

The random angle θ is selected using the roulette wheel selection algorithm, a probabilistic approach that enables dynamic and unbiased selection of movement directions. By randomly selecting θ, the agent's path is less predictable, reducing the likelihood of being trapped in local optima. The cosine function applied to θ introduces controlled oscillations in the movement trajectory, allowing the agent to explore more refined areas of the search space without losing diversity. R dynamically regulates the ratio of exploration to exploitation, making sure that the search agent both investigates novel possibilities and converges on the best answers. Achieving an ideal search procedure in optimization issues requires striking this balance.

Figure 6 describes the SCSO algorithm for selecting an optimal test suite based on test coverage. The process begins by initializing a population of search agents and calculating their fitness using a test coverage-based function. Parameters r and R are

Pseudocode for Sand Cat Swarm Optimization Algorithm

```
Input: Tuple List, TL;
Output: Final Test Suite, FTS
Begin
Initialize population of search agents
Calculate the fitness function based on test coverage
Initialize parameters r, R
While TL ≠ Ø {
 For each search agent {
  Get a random angle based on Roulette Wheel Selection (0° ≤ θ ≤ 360°)
   If (abs(R) ≤ 1) {
    Update the search agent position using Eq.5 (exploration phase)
   } Else {
    Update the search agent position using Eq.4 (exploitation phase)
   }
 }
 Evaluate fitness of all search agents
 Select the best test case candidate
 If best candidate improves coverage {
  Add candidate to FTS
  Remove covered interactions from TL
 }
 Update population based on best solution
}
End
```

Figure 6. Pseudocode for Sand Cat Swarm Optimization Algorithm

also initialized. The algorithm iterates while the TL is not empty. Within each iteration, every search agent selects a random angle using a roulette wheel selection mechanism. Based on the absolute value of R, the search agent updates its position using either the exploration equation [Eq. (5)] if $|R| \leq 1$ or the exploitation equation [Eq. (4)] otherwise. After updating positions, the algorithm evaluates the fitness of all search agents and selects the best test case candidate. If the selected candidate improves coverage, it is added to the FTS, and the covered interactions are removed from TL. The population is then updated based on the best solution. This process continues until all interactions in TL are covered, and the algorithm terminates, returning the optimized FTS.

5. Results and Discussion

The SCSO algorithm is developed and compiled using the Java programming language within the Eclipse 2024 (4.34.0) software. The running environment for the project is a desktop PC operating on Windows 11, equipped with a 2.19 GHz Intel® Core™ i7-12700F CPU and 32 GB of RAM. The three groups of experiments are as follows:

Group 1: CA(t, v^7), t varied from 2 to 6 and v varied from 2 to 5 based on [38, 43, 45, 47].

Group 2: CA(t, 2^{10}) based on [38, 44, 45, 47].

Group 3: CA(t, 3^7) based on [41, 44].

This project focuses solely on uniform strength configurations, as SCSO is a newly introduced approach to t-way testing. To evaluate its performance in generating test suite sizes for uniform strength, three groups of experiments were conducted, based on configurations derived from two separate journal papers. These experiments benchmarked the SCSO against results from various existing strategies discussed in Section 2. The experimental configurations were adapted from prior works by [38, 41, 43–45, 47], where the SCSO has been executed and benchmarked with those various experiments results. The results of these experiments are documented and summarized in Table 2, Table 3, and Table 4. Detailed experimental configurations for each group are presented accordingly.

The results of all experiments are presented in the provided tables, with the best test suite sizes highlighted in bold and cells are darkened for clarity. Cells marked as “X” indicate that results are unavailable or not reported in the respective articles. Based on the data in Table 2, SCSO delivers good competitive performance across all configurations of t and v.

The metaheuristic-based techniques generally outperform computation-based strategies in Table 2. Among these techniques, the HHH strategy demonstrated the highest performance, achieving the best results in 40% (8 out of 20) of the test cases. Following closely, ACOF secured 30% (6 out of 20), while QLSCA performed well with 25% (5 out of 20). Strategies such as GSTG, WOA, and HSS each accounted for 20% (4 out of 20) of the top results, whereas ATLBO followed with 15% (3 out of 20). Meanwhile, WFS, PSTG, CS, TConfig, and IPOG each contributed to 10%.

(2 out of 20) of the best-performing cases. However, SCSO, t-way test suite generation strategy based on ant colony algorithm (TTSGA), and TVG recorded

Table 2. Result Test Suite Size Performance for Group 1

CA(t,v^7)		Metaheuristic-based Strategies												Computational-based Strategies		
t	v	2026	2024	2023	2022	2020	2019	2018	2017	2016	2011	2010	2009	2010	2007	1990s
		SCSO	WFS	ACOF	GSTG	WOA	TTSGA	QLSCA	ATLBO	HHH	HSS	PSTG	CS	TVG	IPOG	TConfig
2	2	7	7	6	6	6	7	7	7	7	7	6	6	7	8	7
	3	15	15	15	15	14	15	15	15	14	14	15	15	15	17	15
	4	26	27	26	26	25	25	23	23	23	25	26	25	27	28	28
	5	40	40	37	40	36	38	34	34	35	35	37	37	42	42	40
3	2	14	13	12	12	12	12	15	15	15	12	13	12	15	19	16
	3	51	51	48	49	49	49	49	49	49	50	50	49	55	57	55
	4	124	123	117	121	116	118	112	111	112	121	116	117	134	208	112
	5	240	242	230	240	223	228	215	216	216	223	225	223	260	275	239
4	2	30	25	30	26	27	29	31	31	31	29	29	27	31	48	36
	3	158	156	148	155	152	152	149	151	148	155	155	155	167	185	166
	4	497	513	485	499	484	485	477	480	482	500	487	487	559	509	568
	5	1220	X	1174	1217	NA	1175	1150	1166	1153	1174	1176	1171	1385	1349	1320
5	2	52	51	53	52	52	55	X	X	58	53	53	53	59	128	56
	3	434	442	430	430	432	433	X	X	435	437	441	439	464	608	477
	4	1833	1861	1825	1822	1815	1821	X	X	1805	1831	1826	1845	2010	2560	1792
	5	5535	X	5450	X	X	5457	X	X	5413	5468	5474	5479	6257	8091	X
6	2	70	66	64	64	64	68	X	X	64	64	64	66	78	64	64
	3	949	955	970	971	945	957	X	X	853	916	977	973	1016	1281	921
	4	5621	5610	5450	5611	5567	5487	X	X	5478	4096	5599	5610	5978	4096	X
	5	21580	X	21145	X	X	21148	X	X	21107	21748	21595	21597	23218	28513	X
Total	***0***	***2***	***6***	***4***	***4***	***0***	***5***	***3***	***8***	***4***	***2***	***2***	***0***	***2***	***2***	
Optimal	***0%***	***10%***	***30%***	***20%***	***20%***	***0%***	***25%***	***15%***	***40%***	***20%***	***10%***	***10%***	***0%***	***10%***	***10%***	

Table 3. Result Test Suite Size Performance for Group 2

CA(t, 2^{10})	Metaheuristic-based Strategies											Computational-based Strategies		
	2026	2024	2023	2022	2022	2019	2016	2015	2011	2010	2009	2010	2007	1990s
t	SCSO	WFS	ACOF	GSTG	ImprovedPSO	TTSGA	HHH	SITG	HSS	PSTG	CS	TVG	IPOG	TConfig
2	8	8	8	8	9	8	8	9	7	8	8	10	10	9
3	17	16	16	16	16	16	16	16	16	17	16	17	19	20
4	34	38	39	27	39	36	36	44	37	37	36	41	49	45
5	84	84	74	74	84	76	79	87	81	82	79	84	128	95
6	159	160	153	156	168	155	153	174	158	158	157	168	352	183
Total	***0***	***1***	***3***	***3***	***1***	***1***	***2***	***1***	***2***	***0***	***1***	***0***	***0***	***0***
Optimal	***0%***	***20%***	***60%***	***60%***	***20%***	***20%***	***40%***	***20%***	***40%***	***0%***	***20%***	***0%***	***0%***	***0%***

the lowest performance, failing to achieve the best results in any of the test cases (0%).

The analysis of Table 3 reaffirms the dominance of ACOF and GSTG, both achieving the highest performance with 60% (3 out of 5) of the test cases, setting a strong benchmark. Meanwhile, HHH and HSS demonstrated balanced performance, each securing 40% (2 out of 5) of the best test cases. In contrast, WFS, CS, improved PSO, SITG, and TTSGA trailed behind, contributing to only 20% (1 out of 5) of the test cases. The lowest-performing strategies in this group were SCSO, PSTG, IPOG, TConfig, and TVG, all of which failed to secure any top-ranking test cases (0%).

Table 4 highlights the dominance of improved PSO, which achieves the highest performance with 80% (4 out of 5) of the test cases, outperforming all other strategies. Meanwhile, SITG, ABCVS, and TConfig each secure 20% (1 out of 5) of the test cases, demonstrating a more limited impact. SCSO, despite not leading the group, manages to contribute 40% (2 out of 5) of the best test cases, showcasing moderate performance and competitive potential. In contrast, IPOG fails to secure any top-ranking test cases (0%), making it the least effective strategy in this configuration. This pattern of results reinforces improved PSO's dominance, the balanced yet limited success of SITG and TConfig, and the moderate competitiveness of SCSO, while IPOG remains at the bottom, struggling to deliver optimal results.

SCSO demonstrates consistent competitiveness across multiple groups, showing its potential in diverse test scenarios. While it performs moderately and has yet to surpass the top-ranked metaheuristic strategies, its stability across configurations highlights its strength. This suggests that with further refinements or hybridization with other metaheuristic techniques, SCSO could achieve even greater optimization and emerge as a strong contender among the top-performing strategies.

The performance of SCSO has been evaluated using statistical analyses to assess its effectiveness in generating optimal test suite sizes for uniform strength interaction test. Two nonparametric tests, the Wilcoxon Rank test and the Friedman test, were employed due to the small sample size and nonnormal distribution of results. The Wilcoxon Rank test analyzed paired strategies with SCSO to determine statistically significant differences at a 95% confidence

Table 4. Result Test Suite Size Performance for Group 3

CA(t, 3^7)	Metaheuristic-based Strategies				Computational-based Strategies	
	2026	2022	2019	2015	2007	1990s
t	SCSO	Improved PSO	ABCVS	SITG	IPOG	TConfig
2	15	15	15	15	17	15
3	51	45	49	52	57	55
4	158	152	157	154	185	166
5	434	444	442	436	608	477
6	949	832	944	848	1281	921
Total	*2*	*4*	*1*	*1*	*0*	*1*
Optimal	*40%*	*80%*	*20%*	*20%*	*0%*	*20%*

level (α = 0.05). A null hypothesis was used, where a *p*-value ≤ α indicated significant differences, leading to the null hypothesis being rejected. Conversely, a *p*-value > α retained the null hypothesis, suggesting no significant difference. The Friedman test ranked strategies based on mean rank, with smaller values indicating better performance in generating test suite sizes when the null hypothesis was rejected.

A total of 30 results from three groups of experiments were analyzed across Table 5 to Table 6, although some configurations were excluded due to unavailable data for certain strategies. These analyses highlight SCSO's competitive performance compared to other strategies, particularly in scenarios where it demonstrated statistically significant advantages or achieved better mean rankings in test suite generation.

Table 5 presents the results of testing conducted using the Wilcoxon and Friedman tests, involving data from Groups 1 and 2. A total of 25 test types were analyzed, with the results offering insights into the comparative performance of different strategies. Table 5 also provides insights into whether the null hypothesis was retained or rejected for each paired strategy involving SCSO. For strategies like ACOF, TTSGA, HHH, HSS, and GSTG, the null hypothesis is rejected, meaning these strategies significantly outperform SCSO. For instance, the pairing with ACOF shows a $p = 0.002$, confirming a clear advantage for ACOF. Similarly, the null hypothesis is rejected when SCSO is compared to TVG, IPOG, WOA, QLSCA, and ATLBO, but in these cases, it is SCSO that demonstrates superior performance, as indicated by its lower mean rank.

In comparisons with PSTG, CS, TConfig, and WFS, the null hypothesis is retained, signifying no statistically significant differences. This implies that SCSO performs similarly to these strategies. For example, against PSTG ($p = 0.055$) and CS ($p = 0.083$), SCSO neither significantly outperforms nor underperforms, reflecting comparable performance levels.

When the null hypothesis is rejected, it means one strategy is statistically better than the other based on the test results. For SCSO, this occurs when it shows clear advantages over strategies like TVG, IPOG, and others, or when it is outperformed by stronger strategies like ACOF and TTSGA. On the other hand, retaining the null hypothesis, as seen in its comparisons with PSTG, CS, and WFS, suggests that SCSO is on par with these strategies, offering neither significantly better nor worse performance.

The majority of the paired strategies that outperformed SCSO, such as ACOF, TTSGA, HSS, improved PSO, and GSTG achieved better rankings primarily because they integrate additional techniques, such as hybridization with other optimization algorithms, rule-based enhancements, or ensemble approaches. These combinations allow them to improve search efficiency, balance exploration and exploitation, and adapt to problem-specific constraints more effectively. For example, ACOF benefits from the self-organizing behavior of ACO while incorporating fuzzy logic to handle uncertainty and improve decision-making. The fuzzy rules help refine the solution space more effectively, reducing randomness and increasing convergence speed, making ACOF superior to SCSO.

Another reason these strategies appear superior is that HHH inherently carries four different metaheuristics, which are TLBO, GNA, PSO, and CS, while SCSO operates as a standalone approach. HHH's adaptive mechanism selects the most suitable metaheuristic at different stages of optimization, ensuring a well-balanced search process. This multi-metaheuristic structure naturally provides an advantage in performance, as it allows HHH to adapt to different problem complexities dynamically. However, this also makes comparisons with SCSO unfair, as HHH benefits from the combined strengths of multiple algorithms, whereas SCSO is evaluated based on a single optimization framework.

ATLBO holds a significant advantage due to its adaptive teacher and learner phases, which dynamically adjust using a fuzzy inference system. Unlike SCSO, which follows a fixed optimization structure, ATLBO intelligently adapts its search strategy based on real-time conditions, allowing it to balance exploration and exploitation more effectively. The teacher phase drives global improvements by guiding learners toward better solutions, while the learner phase enhances local refinement through peer-to-peer learning. This adaptability not only improves solution quality but also reduces the risk of premature convergence. However, this advantage also makes direct comparisons with SCSO less fair, as ATLBO's superior flexibility comes from its additional fuzzy rule-based decision-making system, which SCSO does not incorporate. Essentially, ATLBO benefits from an

Table 5. Wilcoxon and Friedman Test for Group 1 and Group 2

No.	Paired Strategy	Test Statistic							Null Hypothesis	Conclusion
		Comparison for SCSO			Total Samples	*p*-value (Asymp. Sig. (2-tailed))	Friedman Mean Rank Test			
		<	=	>			Mean Rank	Rank		
1.	SCSO vs ACOF	18	4	3	25	0.002	SCSO – 5.76	7	reject	ACOF outperforms
							ACOF – 3.32	2		
2.	SCSO vs TTSGA	19	3	3		0.002	SCSO – 5.76	7	reject	TTSGA outperforms
							TTSGA – 3.88	3		
3.	SCSO vs HHH	18	2	5		0.01	SCSO – 5.76	7	reject	HHH outperforms
							HHH – 2.84	1		
4.	SCSO vs HSS	19	1	5		0.005	SCSO – 5.76	7	reject	HSS outperforms
							HSS – 3.92	4		
5.	SCSO vs PSTG	16	4	5		0.055	SCSO – 5.76	7	retain	no significant difference
							PSTG – 4.86	6		
6.	SCSO vs CS	17	2	6		0.083	SCSO – 5.76	7	retain	no significant difference
							CS – 4.16	5		
7.	SCSO vs TVG	0	4	21		0.001	SCSO – 5.76	7	reject	SCSO outperforms
							TVG – 7.92	8		
8.	SCSO vs IPOG	2	0	23		0.001	SCSO – 5.76	7	reject	SCSO outperforms
							IPOG – 8.34	9		
9.	SCSO vs WFS	8	6	8	22	0.532	SCSO – 2.27	3	retain	no significant difference
							WFS – 2.23	2		
10.	SCSO vs GSTG	14	6	2		0.013	SCSO – 2.27	3	reject	GSTG outperforms
							GSTG – 1.50	1		
11.	SCSO vs TConfig	14	3	5	22	0.099	SCSO – 1.30	1	retain	no significant difference
							TConfig – 1.70	2		
12.	SCSO vs WOA	16	1	0	17	0.001	SCSO – 1.97	2	reject	WOA outperforms
							WOA – 1.03	1		
13.	SCSO vs QLSCA	8	2	2	12	0.012	SCSO – 2.50	3	reject	QLSCA outperforms
							QLSCA – 1.63	1		
14.	SCSO vs ATBLO	8	2	2		0.012	SCSO – 2.50	3	reject	ATBLO outperforms
							ATBLO – 1.88	2		

Table 6. Wilcoxon and Friedman Test for Group 2 and Group 3

No.	Paired Strategy	Test Statistic							Null Hypothesis	Conclusion
		Comparison for SCSO			Total Samples	*p*-value (Asymp. Sig. (2-tailed))	Friedman Mean Rank Test			
		<	=	>			Mean Rank	Rank		
1	SCSO vs Improved PSO	4	2	4	10	0.944	SCSO – 2.00	2	retain	no significant difference
							PSO – 1.85	1		
2	SCSO vs TConfig	1	1	8		0.086	SCSO – 2.00	2	retain	no significant difference
							TConfig –3.75	4		
3	SCSO vs SITG	3	1	6		0.513	SCSO – 2.00	2	retain	no significant difference
							SITG – 2.50	3		
4	SCSO vs IPOG	0	0	10		0.005	SCSO – 2.00	2	reject	SCSO outperforms
							IPOG – 4.90	5		
5	SCSO vs ABCVS	1	1	3	5	0.715	SCSO – 1.70	2	retain	no significant difference
							ABCVS – 1.30	1		

embedded adaptive learning mechanism, making it more versatile yet computationally demanding compared to SCSO.

Table 6 presents the results of testing conducted using the Wilcoxon and Friedman tests, involving data from Groups 2 and 3 with total of 10 types of testing. The table presents a statistical analysis comparing the performance of SCSO against four other strategies: improved PSO, TConfig, SITG, and IPOG. The comparison is based on the Wilcoxon Rank test and the Friedman Mean Rank test. The results indicate that SCSO demonstrates comparable performance to improved PSO, TConfig, ABCVS, and SITG, as the *p*-values for these comparisons (0.944, 0.086, 0.715, and 0.513, respectively) are greater than the significance threshold of 0.05, leading to the retention of the null hypothesis. While SCSO shows slight advantages in certain samples, the Friedman Mean Rank test confirms that the differences are not statistically significant. However, when compared to IPOG, SCSO exhibits a clear statistical advantage, with a p-value of 0.005 leading to the rejection of the null hypothesis. SCSO outperforms IPOG in all samples and achieves a much better Friedman Mean Rank, solidifying its superiority in this comparison.

The analyses highlight that SCSO demonstrates a balanced performance, holding its ground against some strategies while excelling over weaker ones. SCSO shows comparable performance to Improved PSO, TConfig, SITG, PSTG, ABCVS, and CS, as indicated by retained null hypotheses and similar mean ranks. However, it decisively outperforms IPOG and TVG, with statistically significant results and superior Friedman Mean Ranks. Conversely, SCSO struggles against stronger strategies like ACOF, TTSGA, HHH, and HSS, which achieve better rankings in direct comparisons. Overall, SCSO proves to be a robust and competitive strategy, particularly effective against less dominant methods while maintaining reliability in varied contexts.

6. Conclusion

This paper presents the first implementation of SCSO in t-way testing for test suite generation. The results indicate that SCSO performs competitively, outperforming 15.79% of competing strategies, matching 42.11%, but being outperformed in 42.11% of cases. While SCSO shows strengths in handling uniform interaction strengths, it struggles against more advanced techniques such as ACOF, TTSGA, and WOA, highlighting its limitations in maintaining effective exploration throughout the search process.

SCSO mimics sand cat hunting behavior, where movement intensifies as the search progresses toward an optimal solution. However, as it nears the global optimum, its search radius contracts, leading to reduced exploration and increased reliance on exploitation. This transition increases the risk of premature convergence, causing SCSO to become trapped in local optima, particularly in complex search spaces. The benchmark results support this, as SCSO underperforms in nearly half of the cases, suggesting that an improved balance between exploration and exploitation is needed.

To enhance SCSO's effectiveness, future work should focus on improving its exploration capability. Hybridizing it with global optimization techniques such as simulated annealing, genetic algorithm, or Lévy flight could help mitigate premature convergence and improve search diversity. Additionally, extending its application to variable-strength and input–output relationship testing could enhance its adaptability. Exploring integration with other metaheuristic strategies may further refine its efficiency, making SCSO a more competitive approach for t-way test suite generation.

AUTHORS

Muhammad Aiman bin Mohd Asyraf* – Faculty of Intelligent Computing, University Malaysia Perlis, 02600 Arau, Perlis, Malaysia, e-mail: aimanasyraf@studentmail.unimap.edu.my https://orcid.org/0009-0009-3268-0120.

Rozmie Razif Bin Othman – Centre of Excellence for Advanced Computing, (AdvComp), University Malaysia Perlis, Malaysia, e-mail: rozmie@unimap.edu.my https://orcid.org/0000-0001-7940-8487.

Mohd Zamri Bin Zahir Ahmad – Centre of Excellence for Advanced Computing, (AdvComp), University Malaysia Perlis, Malaysia, e-mail: zamrizahir@unimap.edu.my https://orcid.org/0000-0003-3839-2284.

Ahmad Ashraf Abdul Halim – Centre of Excellence for Advanced Computing, (AdvComp), University Malaysia Perlis, Malaysia, e-mail: ashrafhalim@unimap.edu.my https://orcid.org/0000-0002-6152-1964.

Kentaro Go – Department of Computer Science and Engineering, University of Yamanashi, Kofu, Japan, e-mail: go@yamanashi.ac.jp https://orcid.org/0000-0003-3451-7924.

Nuraminah binti Ramli – Centre of Excellence for Advanced Computing, (AdvComp), University Malaysia Perlis, Malaysia, e-mail: nuraminah@unimap.edu.my https://orcid.org/0000-0002-3527-2431.

R. Badlishah Ahmad – Centre of Excellence for Advanced Computing, (AdvComp), University Malaysia Perlis, Malaysia, e-mail: badli@unimap.edu.my https://orcid.org/0000-0002-4862-2728.

Latifah Munirah Kamarudin – Faculty of Intelligent Computing, University Malaysia Perlis, 02600 Arau, Perlis, Malaysia, e-mail: latifahmunirah@unimap.edu.my https://orcid.org/0000-0002-2547-3934 .

Murad Muhammad Hasan Salih Al-Walidi – Faculty of Intelligent Computing, University Malaysia Perlis, 02600 Arau, Perlis, Malaysia, e-mail: muradmuhammad@studentmail.unimap.edu.my https://orcid.org/0009-0004-9672-689X.

*Corresponding author

ACKNOWLEDGEMENTS

The author would like to acknowledge the support from the Fundamental Research Grant Scheme (FRGS) under a grant number of FRGS/1/2024/ICT01/UNIMAP/02/1 from the Ministry of Higher Education Malaysia.

References

[1] N. Anwar and S. Kar, "Review Paper on Various Software Testing Techniques & Strategies," *Global Journal of Computer Science and Technology*, pp. 43–49, May 2019, doi: 10.34257/gjcstcvol19is2pg43.

[2] M. Bajjouk, M. Ehsan Rana, C. Reka Ramachandiran, and S. Chelliah, "Software testing for reliability and quality improvement," *Journal of Applied Technology and Innovation*, vol. 5, no. 2, p. 40, 2021, doi: 10.65136/jati.v5i1.216.

[3] S. Najihi, S. Elhadi, R. A. Abdelouahid, and A. Marzak, "Software Testing from an Agile and

Traditional View," *ScienceDirect Procedia Computer Science*, vol. 203, Aug. 2022, doi: 10.1016/j.procs.2022.07.116.

[4] M. Baqar and R. Khanda, "The Future of Software Testing: AI-Powered Test Case Generation and Validation," 2024.

[5] M. Z. Z. Ahmad, R. R. Othman, M. S. A. R. Ali, N. Ramli, M. W. Nasrudin, and A. A. A. Halim, "A Tuned Version of Ant Colony Optimization Algorithm (TACO) for Uniform Strength T-way Test Suite Generator: An Execution's Time Comparison," *J. Phys. Conf. Ser.*, vol. 1962, no. 1, 2021, doi: 10.1088/1742-6596/1962/1/012037.

[6] R. R. Othman, K. Zuhairi Zamli, and L. E. Nugroho, "General variable strength t-way strategy supporting flexible interactions," *Maejo International Journal of Science and Technology*, vol. 6, no. 03, pp. 415–429, 2012, doi: 10.14456/mijst.2012.30.

[7] A. A. Muazu, A. S. Hashim, and A. Sarlan, "Application and Adjustment of 'don't care' Values in t-way Testing Techniques for Generating an Optimal Test Suite," *Journal of Advances in Information Technology*, vol. 13, no. 4, pp. 347–357, 2022, doi: 10.12720/jait.13.4.347-357.

[8] B. S. Ahmed, "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing," *Engineering Science and Technology, an International Journal*, vol. 19, no. 2, pp. 737–753, Jun. 2016, doi: 10.1016/j.jestch.2015.11.006.

[9] H. M. Fadhil, M. N. Abdullah, and M. I. Younis, "TWGH: A Tripartite Whale–Gray Wolf–Harmony Algorithm to Minimize Combinatorial Test Suite Problem," *Electronics (Switzerland)*, vol. 11, no. 18, Sep. 2022, doi: 10.3390/electronics11182885.

[10] N. Ramli, R. R. Othman, Z. I. Abdul Khalib, and M. Jusoh, "A Review on Recent T-way Combinatorial Testing Strategy," in *MATEC Web of Conferences*, EDP Sciences, Dec. 2017. doi: 10.1051/matecconf/201714001016.

[11] K. M. Htay, R. R. Othman, A. Amir, and J. M. H. Alkanaani, "Gravitational search algorithm based strategy for combinatorial t-way test suite generation," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, pp. 4860–4873, 2022, doi: 10.1016/j.jksuci.2021.06.020.

[12] E. Pira and M. Khodizadeh-Nahari, "Combinatorial t-way test suite generation using an improved asexual reproduction optimization algorithm," *Appl. Soft Comput.*, vol. 150, 2024, doi: 10.1016/j.asoc.2023.111070.

[13] M. Alshinwan et al., "Enhanced Prairie Dog Optimization with Differential Evolution for solving engineering design problems and network intrusion detection system," *Heliyon*, vol. 10, no. 17, Sep. 2024, doi: 10.1016/j.heliyon.2024.e36663.

[14] D. Wu, H. Rao, C. Wen, H. Jia, Q. Liu, and L. Abualigah, "Modified Sand Cat Swarm Optimization Algorithm for Solving Constrained Engineering Optimization Problems," *Mathematics*, vol. 10, no. 22, Nov. 2022, doi: 10.3390/math10224350.

[15] A. Seyyedabbasi and F. Kiani, "Sand Cat swarm optimization: a nature-inspired algorithm to solve global optimization problems," *Eng. Comput.*, vol. 39, no. 4, pp. 2627–2651, 2022, doi: 10.1007/s00366-022-01604-x.

[16] Ana Crudu and MoldStud Research Team, "Role of software testing in ensuring product quality," MoldStud. Accessed: Feb. 23, 2025. [Online]. Available: https://moldstud.com/articles/p-role-of-software-testing-in-ensuring-product-quality#:~:text=Software%20testing%20is%20important%20for,and%20effectiveness%20of%20software%20testing

[17] K. M. Htay, H. L. Zakaria, R. R. Othman, N. Ramli, and A. Amir, "A Pairwise T-Way Test Suite Generation Strategy Using Gravitational Search Algorithm," *ICAICST 2021 - 2021 International Conference on Artificial Intelligence and Computer Science Technology*, 2021.

[18] D. R. Kuhn, R. Bryce, F. Duan, L. S. Ghandehari, Y. Lei, and R. N. Kacker, "Combinatorial Testing: Theory and Practice," in *Advances in Computers no. 99*, 2015, ch. 1, p. 99. Accessed: Feb. 23, 2025. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=918448

[19] Muazu A. A., Hashim A. S., Audi U. I., Aliyu Y., and Yahaya M. S., "Combinatorial Interaction Testing for T-Way Test Case Generation: A Scoping Review of the Perspective Features," *Journal of China University of Mining and Technology*, no. 4, p. 121, 2024, doi: 10.1654/zkdx.2024.29.4-1.

[20] A. Aminu Muazu, A. Sobri Hashim, A. Sarlan, and M. Abdullahi, "SCIPOG: Seeding and constraint support in IPOG strategy for combinatorial t-way testing to generate optimum test cases," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 1, pp. 185–201, Jan. 2023, doi: 10.1016/j.jksuci.2022.11.010.

[21] N. F. Maidin, S. Hassan, S. Baharom, and A. B. Md. Sultan, "A Comparative Study on Testing Optimization Techniques with Combinatorial Interaction Testing for Optimizing Software Product Line Testing," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, no. 1, pp. 77–94, 2025, doi: https://doi.org/10.37934/araset.49.1.7794.

[22] J. M. Altmemi, R. R. Othman, and R. Ahmad, "Review of combinatorial testing strategy,"

International Journal of Advanced Trends in Computer Science and Engineering, vol. 8, no. 6, pp. 2877–2881, Nov. 2019, doi: 10.30534/ijatcse/2019/31862019.

[23] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E. G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," Jan. 16, 2022, *Elsevier B.V.* doi: 10.1016/j.ejor.2021.04.032.

[24] N. Ramli, R. R. Othman, R. Hendradi, and I. Iszaidy, "T-way Test Suite Generation Strategy based on Ant Colony Algorithm to Support T-way Variable Strength," *J. Phys. Conf. Ser.*, vol. 1755, no. 10, 2020.

[25] A. B. Nasser, A. A. Alsewari, N. M. Tairan, and K. Z. Zamli, "Pairwise Test Data Generation Based On Flower Pollination Algorithm," *Malaysian Journal of Computer Science*, pp. 242–257, 2017, Accessed: Feb. 05, 2025. [Online]. Available: https://ejournal.um.edu.my/index.php/MJCS/article/view/7069/4715

[26] P. A. Kowalski, S. Kucharczyk, and J. Mańdziuk, "Constrained Hybrid Metaheuristic Algorithm for Probabilistic Neural Networks Learning," *ArXiv*, Jan. 2025, [Online]. Available: http://arxiv.org/abs/2501.15661

[27] Williams A., "TConfig - Test Configuration Generator," Github. Accessed: Feb. 05, 2025. [Online]. Available: https://github.com/awwilliams/tconfig?tab=readme-ov-file

[28] Y. Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," *Proceedings - 3rd IEEE International High-Assurance Systems Engineering Symposium, HASE 1998*, 1998, doi: 10.1109/HASE.1998.731623.

[29] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A General Strategy for T-Way Software Testing," in *Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems*, 2007, pp. 549–556.

[30] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG-IPOG-D: Efficient test generation for multi-way combinatorial testing," *Software Testing Verification and Reliability*, vol. 18, no. 3, pp. 125–148, Sep. 2008, doi: 10.1002/stvr.381.

[31] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays," *J. Res. Natl. Inst. Stand. Technol.*, vol. 113, pp. 287–297, 2008, doi: 10.6028/jres.113.022.

[32] J. Mohammed and H. Altmemi, "An Analogous T-Way Test Generation Strategy for Software Systems MC-MIPOG," *Software Engineering*, vol. 7, no. 1, pp. 1–12, 2018, doi: 10.5923/j.se.20180701.01.

[33] B. S. Ahmed and K. Z. Zamli, "PSTG: A t-way strategy adopting particle Swarm Optimization," *AMS2010: Asia Modelling Symposium 2010 - 4th International Conference on Mathematical Modelling and Computer Simulation*, pp. 1–5, 2010, doi: 10.1109/AMS.2010.14.

[34] B. S. Ahmed and K. Z. Zamli, "A Variable Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2171–2185, 2011, doi: 10.1016/j.jss.2011.06.004.

[35] A. R. A. Alsewari and K. Z. Zamli, "Interaction Test Data Generation Using Harmony Search Algorithm," *2011 IEEE Symposium on Industrial Electronics and Applications, ISIEA 2011*, pp. 559–564, 2011, doi: 10.1109/ISIEA.2011.6108775.

[36] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of Minimized Combinatorial Test Suite for Configuration-aware Software Functional Testing Using The Cuckoo Search algorithm," *Inf. Softw. Technol.*, vol. 66, pp. 13–29, 2015, doi: 10.1016/j.infsof.2015.05.005.

[37] K. Rabbi, Q. Mamun, and R. Islam, "An Efficient Particle Swarm Intelligence Based Strategy to Generate Optimum Test Data in T-way Testing," *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 123–128, 2015, doi: 10.1109/ICIEA.2015.7334096.

[38] K. Z. Zamli, B. Y. Alkazemi, and G. Kendall, "A Tabu Search Hyper-Heuristic Strategy for T-way Test Suite Generation," *Applied Soft Computing Journal*, vol. 44, pp. 57–74, 2016, doi: 10.1016/j.asoc.2016.03.021.

[39] F. Din and K. Z. Zamli, "Fuzzy adaptive teaching learning-based optimization strategy for GUI functional test cases generation," in *ACM International Conference Proceeding Series*, 2018. doi: 10.1145/3185089.3185148.

[40] K. Z. Zamli, F. Din, B. S. Ahmed, and M. Bures, "A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem," *PLoS One*, vol. 13, no. 5, May 2018, doi: 10.1371/journal.pone.0195675.

[41] A. K. Alazzawi, H. M. Rais, and S. Basri, "ABCVS: An Artificial Bee Colony for Generating Variable T-Way Test Sets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 4, pp. 259–274, 2019, doi: 10.14569/ijacsa.2019.0100431.

[42] M. Z. Z. Ahmad, R. R. Othman, M. S. A. R. Ali, and N. Ramli, "A Self-Adapting Ant Colony Optimization Algorithm Using Fuzzy Logic (ACOF) for Combinatorial Test Suite Generation," in *IOP*

Conference Series: Materials Science and Engineering, 2020. doi: 10.1088/1757-899X/767/1/012017.

[43] A. A. Hassan, S. Abdullah, K. Z. Zamli, and R. Razali, "Combinatorial Test Suites Generation Strategy Utilizing the Whale Optimization Algorithm," *IEEE Access*, vol. 8, pp. 192288–192303, 2020, doi: 10.1109/access.2020.3032851.

[44] M. L. Prasad, J. K. R. Sastry, B. Mallikarjuna, V. Sitaramulu, C. Srinivasulu, and B. B. Naib, "Development of a Programmed Generation of t-way Test cases Using an Improved Particle Swarm Optimization Strategy," in *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering, ICACITE 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1394–1399. doi: 10.1109/ICACITE53722.2022.9823783.

[45] N. H. C. Rose, R. R. Othman, H. L. Zakaria, A. J. Suali, and Z. A. Ahmad, "Wingsuit Flying Search Optimization Algorithm Strategy for Combinatorial T-Way Test Suite Generation," *International Journal of Advances in Soft Computing and its Applications*, vol. 16, no. 3, pp. 272–293, 2024, doi: 10.15849/IJASCA.241130.15.

[46] J. Yang, Y. Zhang, Z. Wang, Y. Todo, B. Lu, and S. Gao, "A Cooperative Coevolution Wingsuit Flying Search Algorithm with Spherical Evolution," *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, Dec. 2021, doi: 10.1007/s44196-021-00030-z.

[47] M. Z. Z. Ahmad, "A Self-Adapting Ant Colony Optimization Algorithm using Fuzzy Logic (ACOF) for Combinatorial Test Suite Generation," Universiti Malaysia Perlis, 2023.