EMITTER International Journal of Engineering Technology
Vol. 13, No. 2, December 2025, pp. 229~252
DOI : 10.24003/emitter.v13i2.979

Enhanced Wingsuit Flying Search (EWFS) Algorithm for Combinatorial T-way Test Suite Generation

Nurol Husna Che Rose[1], Rozmie Razif Othman[1], Hasneeza Liza Zakaria[1], Anjila J Suali[1], Husna Jamal Abdul Nasir[1], Jalal Mohammed Hachim Altmemi[2]

[1] Advanced Computing, Centre of Excellence (CoE), Universiti Malaysia Perlis, Malaysia
[2] Information Technology Management Department, Southern Technical University, Basrah, Iraq
Corresponding Author: husnarose@unimap.edu.my

Received July 10, 2025; Revised August 17, 2025; Accepted October 5, 2025

Abstract

The Wingsuit Flying Search (WFS) algorithm is a newly developed global meta-heuristic algorithm. It is efficient and easy to implement, requiring no parameter tuning apart from the population size and the maximum number of iterations. Recently, WFS has been developed based on applying t-way strategies, where t represents the interaction strength. Despite the encouraging results, WFS's search strategy leans more toward local optima due to the narrowing of the boundary search space and the increased value of the search sharpness. Hybridising two or more algorithms enhances search performance by effectively balancing the strengths and mitigating the weaknesses of each method. Thus, this paper proposes a new hybrid Lévy Flight with Wingsuit Flying Search (WFS) algorithm called Enhanced Wingsuit Flying Search Algorithm (EWFS). EWFS uses a control mechanism to identify the best dynamic solution during runtime. The Lévy Flight motion helps the solution escape from local optima and improves the searching process when it gets stuck. Comparison between EWFS and WFS uses the benchmarking configuration of CA(N; 2, 5^7), while the comparison with other metaheuristic algorithms is based on the following covering array configurations: CA(N; t, 3^p), CA(N; t, v^7), CA(N; 2, 2^p), and CA(N; t, 2^{10}). The experimental result shows that EWFS is statistically better regarding test suite size reduction than the recent t-way strategies. It also offers improved results of 65% over the original WFS and resolves the issues of excessive exploitation and getting stuck in local minima or maxima.

Keywords: Combinatorial testing, T-way testing, Wingsuit Flying Search (WFS) Algorithm, Optimization

1. INTRODUCTION

Testing plays a crucial role in the software development lifecycle. As an essential aspect of quality assurance, testing ensures that the software operates as specified. It helps identify and prevent potential issues or defects [1], [2]. Exhaustive testing is highly effective as it thoroughly examines every aspect of the testing process. Nevertheless, this approach is often impractical due to time constraints and data storage limitations. As a result, researchers have proposed various sampling-based techniques to reduce the size of test suites. These techniques include equivalence partitioning and boundary value analysis. However, this approach has limitations when applied to systems involving interaction parameters [3].

As a solution, researchers have introduced combinatorial testing to address faults arising from interactions. This approach is known as t-way testing, where t represents the interaction strength. It efficiently detects defects resulting from parameter interactions. T-way testing systematically reduces the test size, decreasing the effort required for test execution.

Metaheuristic strategies have become an option in t-way strategies to achieve the minimum test case intelligently. The core algorithm often influences the effectiveness of a metaheuristic-based t-way strategy. One example is the Wingsuit Flying Search Optimization (WFS) algorithm [4]. WFS is a newly developed optimisation algorithm with potential for use in t-way strategies. However, there are some limitations to this algorithm. First, the search sharpness of the WFS algorithm will gradually increase for each iteration. This will indirectly cause the size of the search boundary space to decrease and the number of new neighbors in the population to increase, leading to exploitation. Thus, the search is likely to get stuck at the local optimum. Secondly, if the situation persists, WFS lacks an effective mechanism to escape local optima.

The situation will impact the research results and lead to inaccuracies in finding the minimal test suite size. To overcome these challenges, this paper proposes an enhancement for WFS by incorporating the Lévy Flight algorithm with WFS. The Lévy Flight Optimization Algorithm provides balanced control of the WFS parameters between exploration and exploitation search. Therefore, the key contributions of this research can be summarised as follows:

- A new strategy is proposed to generate t-way test suites using the Wingsuit Flying Search (WFS) algorithm, referred to as the EWFS algorithm. According to the rules of the control mechanism in the EWFS algorithm, Lévy Flight is used to enhance search space exploration.
- The performance of EWFS is evaluated against other existing t-way strategies. The benchmark analysis focuses on a combinatorial test suite with constraint support. Furthermore, the Wilcoxon Signed Rank test compares EWFS with the original WFS and another algorithm.

1.1 BACKGROUND OF T-WAY TESTING

To illustrate t-way testing, consider the following hypothetical scenario of a smart home security system, as shown in Figure 1. Exhaustive testing of the Smart Home System requires evaluating all possible combinations of its configurations, as each component (motion sensor, window sensor, door sensor, smoke detector, and lighting control) contains two values (i.e., motion sensor = {motion, motionless}, window sensor = {open, closed}, door sensor = {authorised open, unauthorised open}, smoke detector = {smoke, no smoke}, lightning control = {dim, off}. This leads to 2 being raised to the power of 5, or 32 test cases, to cover all possible scenarios.

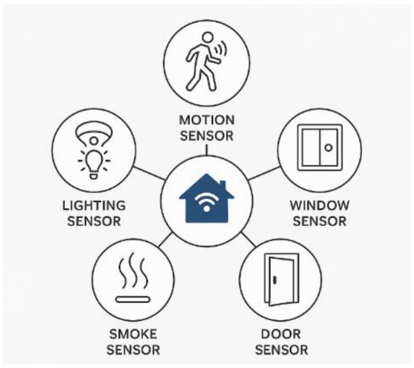
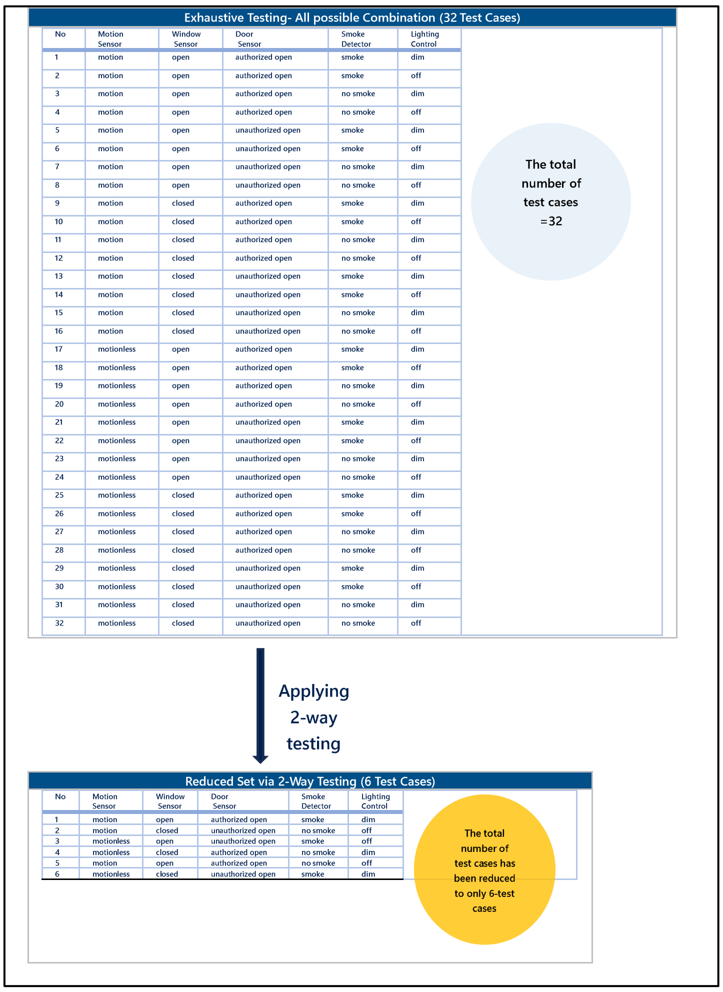


Figure 1. The sensors integrated in a Smart Home System


Exhaustive Testing- All possible Combination (32 Test Cases)

No	Motion Sensor	Window Sensor	Door Sensor	Smoke Detector	Lighting Control
1	motion	open	authorized open	smoke	dim
2	motion	open	authorized open	smoke	off
3	motion	open	authorized open	no smoke	dim
4	motion	open	authorized open	no smoke	off
5	motion	open	unauthorized open	smoke	dim
6	motion	open	unauthorized open	smoke	off
7	motion	open	unauthorized open	no smoke	dim
8	motion	open	unauthorized open	no smoke	off
9	motion	closed	authorized open	smoke	dim
10	motion	closed	authorized open	smoke	off
11	motion	closed	authorized open	no smoke	dim
12	motion	closed	authorized open	no smoke	off
13	motion	closed	unauthorized open	smoke	dim
14	motion	closed	unauthorized open	smoke	off
15	motion	closed	unauthorized open	no smoke	dim
16	motion	closed	unauthorized open	no smoke	off
17	motionless	open	authorized open	smoke	dim
18	motionless	open	authorized open	smoke	off
19	motionless	open	authorized open	no smoke	dim
20	motionless	open	authorized open	no smoke	off
21	motionless	open	unauthorized open	smoke	dim
22	motionless	open	unauthorized open	smoke	off
23	motionless	open	unauthorized open	no smoke	dim
24	motionless	open	unauthorized open	no smoke	off
25	motionless	closed	authorized open	smoke	dim
26	motionless	closed	authorized open	smoke	off
27	motionless	closed	authorized open	no smoke	dim
28	motionless	closed	authorized open	no smoke	off
29	motionless	closed	unauthorized open	smoke	dim
30	motionless	closed	unauthorized open	smoke	off
31	motionless	closed	unauthorized open	no smoke	dim
32	motionless	closed	unauthorized open	no smoke	off

The total number of test cases =32

Applying 2-way testing

Reduced Set via 2-Way Testing (6 Test Cases)

No	Motion Sensor	Window Sensor	Door Sensor	Smoke Detector	Lighting Control
1	motion	open	authorized open	smoke	dim
2	motion	closed	unauthorized open	no smoke	off
3	motionless	open	unauthorized open	smoke	off
4	motionless	closed	authorized open	no smoke	dim
5	motion	open	authorized open	no smoke	off
6	motionless	closed	unauthorized open	smoke	dim

The total number of test cases has been reduced to only 6-test cases

Figure 2. Exhaustive testing compared to t-way testing

However, two-way testing (t = 2) can significantly reduce test cases. Two-way testing focuses on examining interactions between pairs of components rather than all possible combinations, which is often sufficient for identifying configuration faults. As a result, only six test cases are necessary to comprehensively cover the system's configuration space, as illustrated in Figure 2. This approach reduces the testing effort and ensures efficient fault detection while maintaining the system's reliability.

2. RELATED WORKS

To thoroughly examine all possible solutions, t-way optimisation algorithms generate different combinations of parameters and their corresponding values. This process finds the best solution by balancing exploration and exploitation. The search process can be categorised into two main types: computational search and metaheuristic search. Researchers use mathematical formulas and equations for computational search and assign appropriate values accordingly. The algorithms methodically evaluate possible solutions based on predefined rules or heuristics. This approach typically uses greedy techniques [5], where the optimal choice is considered. Test cases covering the most uncovered interactions will be prioritised in this context.

This approach utilises test suite construction methods such as One Parameter at a Time (OPAT) and One Test at a Time (OTAT). For OPAT, a parameter is selected and combined with other system input parameters horizontally [6]. This extension creates new test data for the test suite and covers any uncovered tuples. The extension process iterates until all tuples are covered. IPOG [7] and IPOG-D [8] are examples of OPAT strategies. In contrast, OTAT generates one test case at a time by combining parameters. Generated test cases are compared in each iteration, and the one covering the most interaction tuples is selected and added vertically to the final test suite. This process repeats iteratively until all tuples are covered at least once. Various strategies, including those utilising metaheuristic algorithms, have adopted OTAT methods. These strategies include Jenny [9], Pairwise Independent Combinatorial Testing (PICT)[10], [11], and Test Configuration (TConfig)[12].

Metaheuristic search is an advanced algorithmic approach to solving complex optimisation problems. It usually draws inspiration from various sources, such as natural phenomena or behaviors [13]. It starts by generating a random solution within the search space, individually or as part of a population. Then, a search technique is applied iteratively to find a near-optimal solution. The fitness of each population point is calculated. The test case with the highest value is then selected and added to the final test suite.

Metaheuristic algorithms can be classified based on their search behavior into three categories. These include global-based algorithms (exploration), local-based algorithms (exploitation), and hybrid algorithms that combine both exploration and exploitation [14]. Algorithms that prioritise discovering new areas in the solution space are known as global algorithms.

These algorithms perform searches over a wide area and focus on avoiding being trapped in local optima. On the other hand, algorithms that refine solutions within a small area are categorised as local algorithms. These algorithms improve or fine-tune solutions within a known good region of the solution space [15]. They focus on precise optimisation toward the best solution and are often associated with local search strategies. Hybrid algorithms combine global and local search mechanisms to maintain a balance between exploration and exploitation [16], [17]

Particle Swarm Optimisation (PSO) is a global optimisation algorithm inspired by the social behavior of birds flocking or fish schooling. In this approach, individuals collaborate and communicate to achieve a common goal. Initially, the particle is randomly initialised. The particles then move based on their current position's personal best (pBest) and the population's global best (gBest). This movement is assisted by a velocity vector that indicates the magnitude and direction of the particles' speed. The particles explore the search space by evaluating their personal best solutions. They also compare them with the global best solution found by any particle in the swarm. After several iterations, all particles will eventually gather at the last best location, the global minimum of the population. PSO has since evolved into new variants, including Discrete Particle Swarm Optimization (DPSO)[18] and Adaptive Particle Swarm Optimization (APSO) [19].

Genetic Algorithm (GA) is a global evolutionary algorithm for optimisation and search problems. It is based on natural selection, where the fittest individuals are chosen to reproduce. These individuals then generate offspring for the next generation.GA explores a broad search space and uses mechanisms to escape from local optima. GA has been enhanced as a Genetic Strategy (GS) that applies a tuning process in generating test suites for t-way testing [20]. The parameters of GS, such as population size, crossover rate, and mutation rate, were carefully optimised. This fine-tuning aimed to enhance both its efficiency and performance.

Cuckoo Search (CS) is a global optimisation algorithm that reduces the search space by reducing the number of test cases that require evaluation [21]. It is used to optimise test case selection by utilising the Lévy Flight mechanism, which helps avoid it from getting stuck in local optima [22].

[23] introduced a hybrid approach that combines the local-based Sine Cosine Algorithm (SCA) with Q-learning, known as QLSCA. Hybridisation is an alternative proposed by researchers to strengthen the strategy and overcome the algorithm's weaknesses. In t-way testing, this hybridisation aims to minimise the number of test suites. The Q-learning methodology uses the SCA switching probability method to decide the optimal course of action during runtime rather than a system of rewards and penalties. To increase the range of solutions, the QLSCA incorporates the Lévy Flight motion and crossover to allow escape from local optima.

The Artificial Bee Colony (ABC) is a global-based optimisation algorithm created by simulating a honey bee colony's eating pattern. The ABC strategy was implemented in t-way testing. It is known as the Artificial Bee Colony for generating Variable t-way Sets (ABCVS) [24]. In this algorithm, honey represents the test cases, while high-quality honey refers to test cases that cover the most interaction elements.

Another strategy, known as The Flower Pollination Algorithm (FPA)[25], is modeled after the pollination behavior of flowering plants. Pollen grains are transferred from the male anther of one flower to the female stigma of another through pollinators such as insects, birds, wind, bats, and water. Implemented the Pairwise Flower Strategy (PairFS) and the Flower Pollination Algorithm (FPA) for t-way testing. It offers an efficient approach with fewer control parameters and integrates global and local search mechanisms [26].

Then, a local-based algorithm, Migrating Bird Optimization (MBO)[27], was inspired by the V-shaped flight formation of migrating birds, which enhances energy efficiency and communication. In t-way testing, MBO treats test cases as "birds" in the formation, optimising their arrangement to generate an effective test suite. The MBO method identifies the best test cases by applying the energy-saving behaviors of various long-distance flying birds through neighborhood search [28]. The technique employs a leader-follower mechanism to refine the positions of the "birds progressively" (test cases). The leader guides the movement while the followers adjust their positions to reduce overlap and enhance coverage.

The Graph-Based Greedy Algorithm (GBGA) [29] is a local search algorithm that constructs a Covering Array (CA) using a graph representation. GBGA creates test cases for t-way testing by visualising test suites as a graph. Edges show legitimate relationships, whereas nodes represent test parameters or combinations. The approach minimises computational effort while ensuring practical and systematic test suite development. It achieves this by prioritising the coverage of the least tested combinations at each stage.

The Pairwise Kidney Algorithm is a local search algorithm designed for t-way combinatorial testing created by [30]. The "kidney" metaphor in this method refers to a data structure or process. It simplifies the systematic pairing of input parameter values to ensure that all possible pairs are covered (pairwise testing). Compared to exhaustive testing, the pairwise kidney approach enhances resource utilisation and reduces computing costs. This makes it particularly effective for scenarios with large input spaces.

As a variation of the Jaya Algorithm (JA) for t-way testing, the local search Improved Jaya Algorithm (IJA)[31] was created. To improve the original JA's search capabilities, IJA adds Lévy Flight and mutation operators. An additional variation is used to enhance the search process's diversity: the Latin Hypercube Sampling Strategy (LHS-JA) [32]. The Latin Hypercube Sampling Strategy (LHS-JA) enhances the Jaya Algorithm in t-way testing by broadening the search's scope. By ensuring uniform test case sampling throughout the input space, LHS improves coverage and avoids clustering. LHS improves efficiency and guarantees comprehensive coverage of all parameter combinations by assisting in the creation of a more representative and balanced test suite when used with the Jaya Algorithm.

3. ORIGINALITY

The originality of this research lies in the novel hybridization of the WFS Algorithm with the Lévy Flight algorithm, referred to as the EWFS algorithm. The WFS algorithm tends to focus more on exploitation during the search process, which limits its ability to explore the search space effectively. To address this limitation, this study hybridizes WFS with the Lévy Flight algorithm, a global search technique emphasizing exploration. By combining these two methods, the proposed approach balances exploitation and exploration. This balance allows it to effectively discover the smallest test suite size and optimize test suite generation in software testing. This research introduces the first hybridization of WFS with Lévy Flight for test suite generation, and this unique approach serves as the original contribution of the study.

4. SYSTEM DESIGN

4.1 WFS OPTIMISATION ALGORITHM

The Wingsuit Flying Search (WFS) algorithm, introduced by [4], was a modern nature-inspired metaheuristic algorithm applied in this study. It was designed based on the dynamics of wingsuit flying, where a flyer navigated through the air with precision and control. WFS mimicked the flyer's journey to land on an optimal spot, avoiding rough terrains and exploring smooth regions. The flyer modified its motions, such as the placement of its legs and wings. The flight routes were gradually optimised due to the slow adaptation process, allowing the flyer to reach the global optimum. This unique inspiration formed the basis of WFS's optimisation process. Figure 3 illustrates a wingsuit flyer in flight at the take-off stage.

Figure 3. A wingsuit flyer in flight at the take-off stage (Source: [33])

As seen in Algorithm 1, the Wingsuit Flying Search (WFS) algorithm operates in three phases. First, it generates initial points, then determines the neighborhood size for each point, and finally generates neighborhood points. The Halton Sequence is used to find the N starting locations in the first phase, where N is the population size. Every point is positioned randomly within an n-dimensional box at the first iteration (m=1), with the same starting discretisation step distance and equal spacing.

In the second phase, the neighborhood size for each point is determined. Starting from the second iteration (m=2), the points are sorted in descending order, with the first point considered the best. The best point is allocated the maximum number of neighboring points, denoted as (P_max(m)). The neighborhood size for each point is calculated using Equations (1) and (2), as presented below.

Calculate the neighborhood size using Equation (1).

$$P^{(m)(i)} = \left\lceil P_{max}^{(m)}\left(1 - \frac{(i-1)}{N^{(m)}-1}\right)\right\rceil \tag{1}$$

Calculate the number of nodes using Equation (2).

$$N^{(m)}\frac{2N}{P_{max}^{(m)}} \tag{2}$$

The largest neighborhood size is determined using Equation (3). Calculate the largest neighborhood size using Equation (3).

$$P_{max}^{(m)}\left\lceil \alpha^{(m)}N\right\rceil \tag{3}$$

Each point is associated with a specific neighborhood size, and the neighborhood points are generated based on this size. For instance, if the calculated value of the neighborhood size is 100, this indicates that 100 new neighborhood points are generated for the corresponding point. As the flyer approaches the landing zone, the discretization stage gradually decreases, like the increasing clarity of the landing area. This process matches the algorithm's operation, where greater precision and more informed decisions are made as the flyer approaches the landing. Figure 4 illustrates this concept, depicting the wingsuit flyer with a clear view of the landing area, thus reflecting the reduction in the discretization stage. Equation (4) calculates the discretization step, Δx, which decreases as α(m) approaches 1 in the last iteration when m = M.

Figure 4. A wingsuit flyer with clear landing scenery (Source: [33])

Calculate the discretization step using Equation (4).

$$\Delta \boldsymbol{x}^{(m)} = (1 - \alpha^{(m)})\, \Delta \boldsymbol{x}^{(1)},\ m \geq 2 \tag{4}$$

The boundary of the neighborhood points is defined using the three-set procedure outlined in Equation (5). The trajectory for the boundary of these points is defined by the neighborhood vector (vk, i(m)), which directs the search space's exploration.

Calculate the boundary of neighborhood points using Equation (5).

$$\begin{aligned} S_{k,1}\left(x_i^{(m)}\right) &= \left\{x_{k,i}^{(m)} - \Delta x_k^{(m)},\ x_{k,i}^{(m)}\right\},\ if\ v_{k,i}^{(m)} < 0; \\ S_{k,2}\left(x_i^{(m)}\right) &= \left\{x_{k,i}^{(m)},\ x_{k,i}^{(m)} + \Delta x_k^{(m)}\right\},\ if\ v_{k,i}^{(m)} > 0; \\ S_{k,3}\left(x_i^{(m)}\right) &= S_{k,1}\left(x_i^{(m)}\right) \cup S_{k,2}\left(x_i^{(m)}\right),\ if\ v_{k,i}^{(m)} = 0; \end{aligned} \tag{5}$$

At the end of each iteration, all new points are added to N(m). This process continues until the maximum iteration *M* is reached. The WFS algorithm then selects the point with the highest fitness value. This point is taken as the final output, which was then implemented in t-way testing by [34]. According to the results, the WFS algorithm in the t-way performs competitively in various configurations and delivers promising outcomes, especially in producing minimal test suite sizes efficiently. Figure 5 shows the WFS pseudocode.

```
Algorithm 1: Pseudocode of WFS

1. Initialize the required parameters.
2. Generate the initial points in an n-dimensional box using the Halton Sequence.
3. while m-iteration equals two and m < M do
    3.1 Determine the neighborhood size for each point using Equation (1)
    3.2 Calculate the number of nodes using Equation (2)
    3.3 Calculate the largest neighborhood size using Equation (3).
    3.4 Calculate the discretization steps using Equation (4).
    3.5 Determine and calculate the boundary using Equation (5).
    3.6 Generate the neighborhood points using the current N and equations (1)
        and (2).
    3.7 Calculate the fitness for each point.
    3.8 Sort the points according to the fitness values in descending order.
4. end while
5. Return the best point$P^{(m)(i)}$.
```

Figure 5. WFS Pseudocode

4.2 ENHANCED WINGSUIT FLYING SEARCH (EWFS) OPTIMISATION ALGORITHM

The proposed Enhanced Wingsuit Flying Search (EWFS) optimisation algorithm enhances the original WFS algorithm by integrating it with the Lévy Flight optimisation algorithm to address its existing limitations. One of the primary issues with WFS is that the search sharpness increases with each iteration. This will decrease the size of the search boundary and the number of neighboring points, which leads more towards exploitation. As a result, the algorithm frequently becomes trapped in local optima. Furthermore, WFS lacks an effective mechanism to escape from these local optima, which impacts the accuracy of the results and limits its ability to identify the minimal test suite size.

To overcome these issues, the Lévy Flight optimisation algorithm is incorporated to balance the exploration and exploitation phases of WFS. This integration allows for better control of WFS parameters, ensuring a more robust search process and minimising the chances of getting stuck at local optimal solutions. The primary contribution of this enhancement is its ability to achieve more accurate and efficient optimisation results by leveraging the Lévy Flight algorithm to maintain diversity and improve the global search capability of WFS.

4.2.1 LÈVY'S FLIGHT OPTIMISATION ALGORITHM

Lévy Flight is a random search pattern often used in optimisation algorithms [35]. It is based on the concept of "Lévy distribution," a probability distribution that describes random walks with highly variable step lengths, often following a power-law outline. Sharp peaks, asymmetry, and trailing characterise its probability density distribution. In simple terms, a Lévy Flight involves long jumps followed by smaller, more frequent steps, imitating the movement patterns found in nature, such as the hunting behavior of animals.

Lévy Flights are often generated using stable distributions, which are parameterised by α (stability index) and β (skewness parameter)[36]. The parameter α determines the tail heaviness of the distribution. When $\alpha=2$, the distribution becomes Gaussian, corresponding to a regular random walk. In the context of Lévy Flights, α typically lies within the range $0 < \alpha \leq 20$, where smaller values of α result in heavier tails, leading to more frequent long jumps.

The parameter β, known as the skewness parameter, governs the asymmetry of the distribution. When $\beta=0$, the distribution is symmetric, indicating an equal likelihood of jumps in either direction. A positive value of β skews the distribution to the right, favoring jumps in the positive direction. In contrast, a negative value ($\beta<0$) skews it to the left, favoring jumps in the negative direction. This parameter is crucial in modeling real-world processes where jumps exhibit directional bias or asymmetry rather than occurring randomly.

In optimisation algorithms, alpha (α) and beta (β) shape Lévy Flights' behavior. α controls the balance between small and large steps, aiding local and global search, while β introduces directional bias in the search space. α is also used as a scaling factor to adjust the size of the steps during the search process. The scaling factor formula is as follows.

$$x_{t+1} = x_t + \alpha \, . Le'vy\,(\lambda) \tag{6}$$

4.2.2 EWFS STRATEGY FOR T-WAY TESTING

The WFS and Lévy's Flight algorithms are integrated to form the Enhanced Wingsuit Flying Search (EWFS) optimisation algorithm. The main goal of this integration is to balance exploring and refining solutions to find the best result.

Figure 6 illustrates the application of the t-way strategy in the EWFS framework, while Figure 7 shows the pseudocode of EWFS. The First Phase, the input configuration phase, involves initialising the system input credentials and WFS parameters. In this phase, the number of parameters (p), the strength value (t), and the corresponding values must be specified. Additionally, the iteration count (m), the maximum number of iterations (M), the velocity (v), and the population size (N) must be defined. In the Second Phase, a tuple list is generated from the combination of parameters based on the specified strength and values. As a result, a comprehensive list of tuples is produced during this phase.

The next phase involves the generation of the test suite. In this phase, the WFS population is initialised using the Halton Sequence. Subsequently, test cases are generated through the application of the WFS algorithm. The WFS algorithm involves three sub-phases: a generation of initial points, the determination of neighborhood size, and the generation of neighborhood points.

```
Algorithm 2: Pseudocode of EWFS
1. Initialize the required parameters.
2. Generate the initial points in an n-dimensional box using the Halton Sequence.
3. while m-iteration equals two and m < M do
    3.1 Determine the neighborhood size for each point using Equation (1)
    3.2 Calculate the number of nodes using Equation (2)
    3.3 Calculate the largest neighborhood size using Equation (3).
    3.4 Calculate the discretization steps using Equation (4).
    3.5 Determine and calculate the boundary using Equation (5).
    3.6 Generate the neighborhood points using the current N and Equations (1) and (2).
    3.7 Calculate the fitness for each point.
        for every best solution, do
            update
            while  do
                apply Equation (6)
            end while
        end
    3.8 Sort the points according to the fitness values in descending order.
4. end while
5. Return the best point $P^{(m)(i)}$
```

Figure 6. EWFS Pseudocode

In the phase of generation of initial points, the initial population is generated randomly using the Halton Sequence to ensure a well-distributed search space. In the first iteration (m=1), P(0) representing Pmax is identified as the point with the maximum number of neighbors and is designated as the current best solution.

To determine the neighborhood size phase, the neighborhood size for each point is determined based on two parameters: the alpha value, which represents the sharpness of the search, and the flier velocity. These parameters dictate the extent and density of the search space around each point.

Then, the next phase is the generation of neighborhood points; they occur, and the process of generating neighborhood points involves the calculation of a boundary parameter, Delta x (Δx). As the number of iterations increases, the alpha value (search sharpness) progressively increases, reducing Delta x (Δx). A smaller Delta x (Δx) confines the neighborhood to a narrower boundary, leading to a denser concentration of points. This phenomenon emphasises exploitation but increases the risk of the search getting stuck in a local optimum.

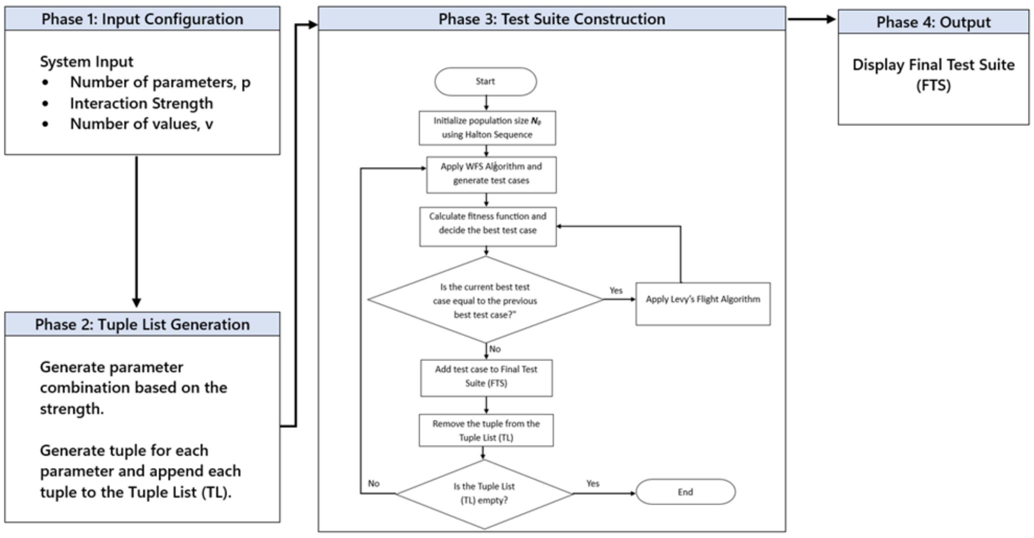

Phase 1: Input Configuration
System Input
Number of parameters, p
Interaction Strength
Number of values, v
Phase 2: Tuple List Generation
Generate parameter combination based on the strength.
Generate tuple for each parameter and append each tuple to the Tuple List (TL).
Phase 3: Test Suite Construction
Start
Initialize population size N_0 using Halton Sequence
Apply WFS Algorithm and generate test cases
Calculate fitness function and decide the best test case
Is the current best test case equal to the previous best test case?"
Yes
Apply Levy's Flight Algorithm
No
Add test case to Final Test Suite (FTS)
Remove the tuple from the Tuple List (TL)
No
Is the Tuple List (TL) empty?
Yes
End
Phase 4: Output
Display Final Test Suite (FTS)

Figure 7. T-way in the EWFS framework

The neighborhood points generated are assessed using the fitness function, and the optimal solution is subsequently incorporated into the Final Test Suite (FTS). At this stage, the control mechanism is essential for maintaining the effectiveness of the solution selection process.

A control mechanism is employed to monitor the current iteration by comparing the best solution, P(0), of the current iteration with the best solution of the previous iteration. The best solution is identified as the population point with the highest neighbor density. If the best solution remains unchanged across successive iterations, it indicates stagnation in a local optimum. To address this issue, the global exploration capabilities of the Lévy's Flight algorithm are utilised.

Lévy's Flight introduces new population points into the existing population using a random walk [34] formula. Following this, Pmax is recalculated. A difference between the new best solution and the previous one signifies that Lévy's Flight has effectively enhanced the WFS algorithm, improving its ability to escape local optima and identify the global optimum.

The control mechanism compares results across iterations, ensuring that the search remains dynamic and avoids local trapping. The best solutions identified in each iteration are recorded in a final test suite. The covered test cases from the Tuple List are removed, and this process proceeds until all the tuples are covered and the list becomes empty. In the end, the final test suite for the t-way strategy in EWFS is produced.

5. EXPERIMENT AND ANALYSIS

5.1 BENCHMARKING WITH WINGSUIT FLYING SEARCH (WFS) ALGORITHM

Two types of experiments have been conducted to evaluate the performance of the EWFS algorithm. The test suite size produced is evaluated using the original WFS algorithm and other existing algorithms. The first experiment is based on one case study covering the array CA (N; 2,5^7). It is selected due to its common use in tuning many AI-based approaches [37]. The population size used is 50, and the iteration was 100. The experiments were conducted on a desktop PC running Windows 10, equipped with a 2.40 GHz Intel® Core™ i5-1135G7 CPU and 4 GB of RAM. The WFS algorithm was implemented using the Java 13.0.1 programming language and executed within the Eclipse development environment.

Table 1. Test case comparison: WFS and EWFS Approaches

No	**Configuration**	**WFS**	**EWFS**
	CA(t,v^P)	Best	Best
1	CA (2,3^3)	**9**	**9**
2	CA (2,3^4)	**9**	**9**
3	CA (2,3^5)	**12**	**12**
4	CA (2,3^6)	14	**13**
5	CA (2,3^7)	15	**14**
6	CA (2,3^8)	16	**15**
7	CA (2,3^9)	17	**16**
8	CA (2,3^{10})	18	**17**
9	CA (2,3^{11})	**18**	**18**
10	CA (2,3^{12})	**18**	**18**
11	CA (2,3^{13})	**19**	**19**
12	CA (3,3^4)	28	**27**
13	CA (3,3^5)	39	**38**
14	CA (3,3^6)	47	**40**
15	CA (3,3^7)	52	**50**
16	CA (3,3^8)	57	**55**
17	CA (3,3^9)	**60**	**60**
18	CA (4,3^5)	**95**	**95**
19	CA (4,3^6)	136	**132**
20	CA (4,3^7)	161	**156**
21	CA (5,2^{10})	84	**81**
22	CA (6,2^{10})	161	**159**

The configuration is divided into four main groups. The first one was CA (2, 3^{3-13}) where t = 2 and v = 3while p varies from 3 to 13; the second one was CA (3, 3^{4-9}) where t = 3 and v = 3 while p varies from 4 to 9; the third one was CA (4, 3^{5-7}) where t = 4 and v = 3 while p ranges from 5 to 7; and the last one was CA (2^{5-6}, 2^{10}) where v = 2 and p = 10 while t varies from 5 to 6. The proposed method was run 100 times, and the best results were recorded.

The Wilcoxon signed-rank test was also run to show the significant difference based on the result in Table 1. The significant difference can be identified from the asymptotic significance (2-tailed) in Table 2, where a p-value of less than 0.05 provides the confidence needed to reject the null hypothesis, thereby indicating a statistically significant difference between the two strategies.

Additionally, Table 2 also shows the rank of EWFS compared to WFS. The label "EWFS>" represents the frequency of EWFS producing more test cases than WFS. Meanwhile, "EWFS<" indicates that the frequency of EWFS produces fewer test cases than WFS. Likewise, "EWFS =" signifies that the frequency of EWFS produces the same number of test cases as WFS.

This study focuses on best-case performance for each configuration, aligning with standard combinatorial test suite research practices. While some studies report additional statistics such as mean or standard deviation, these typically involve different experimental setups. Given the fixed configurations used in this work, highlighting best-case results provides a clearer basis for benchmarking algorithm performance.

The result revealed that the Enhanced Wingsuit Flying Search (EWFS) Algorithm outperforms the original one in most cases. EWFS consistently produces fewer test cases than WFS as the strength (t) and parameter values increase. For example, of 22 configuration settings, eight configurations show ties between EWFS and WFS. Meanwhile for the outcomes of CA (2,3^6), CA (2,3^7), CA (2,3^8), CA (2,3^9), CA (2,3^{10}), CA (3,3^4), CA (3,3^5), CA (3,3^6), CA (3,3^7), CA (3,3^8), CA (4,3^6), CA (4,3^7), CA (5,2^{10}) and CA (6,2^{10}) shows EWFS consistently outperform WFS.

Table 2. Wilcoxon Test Analysis: EWFS Compared to WFS

Algorithm	**Ranks**			**Test Statistics**	
	EWFS>	**EWFS<**	**EWFS =**	**Asymp. Sig (2-Tailed)**	**Null Hypothesis**
WFS	0	14	8	< 0.01	Reject

This indicates that EWFS is likely more efficient in scenarios with larger configurations. Also, through CA (6, 2^{10}), the difference between WFS and EWFS becomes more pronounced at particular higher values, as EWFS produces only 159 vs 161 by WFS. While this may seem minor, the cumulative effect of such reductions could have significant impacts in larger systems where efficiency and performance are crucial. Furthermore, the Wilcoxon signed-rank test results indicate that the null hypothesis is rejected, suggesting a significant difference between WFS and EWFS regarding test case generation.

5.2 BENCHMARKING WITH THE EXISTING STRATEGIES

In the second experiment, EWFS is benchmarked against existing strategies based on the size of the covering array (CA). The experiment is conducted using the following four well-known datasets[22], [31], [38], [39].

Table 3. Test suite size performance for CA (t,3^p)

CA(t, 3^p)		Pure computation strategies					AI-based strategies								
t	p	Jenny	Tconfig	PICT	IPOG-D	IPOG	GBGA	QLSCA	GS	DPSO	APSO	CS	ABVCS	WFS	EWFS
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best
2	3	9	10	10	15	9	9	9	9	NA	9	9	9	9	9
	4	13	10	13	15	9	9	9	9	9	9	9	9	9	9
	5	14	14	13	15	15	12	11	11	11	11	11	11	11	11
	6	15	15	14	15	15	14	14	13	14	12	13	13	13	13
	7	16	15	16	15	15	15	14	14	15	15	14	15	15	14
	8	17	17	16	15	15	15	15	15	15	15	15	15	15	15
	9	18	17	17	15	15	16	15	15	15	16	15	16	16	16
	10	19	17	18	21	15	16	15	16	16	17	17	17	17	17
	11	17	20	18	21	17	17	16	16	17	NA	18	17	17	17
	12	19	20	19	21	21	18	16	16	16	NA	18	18	18	17
3	4	34	32	34	27	32	29	27	27	NA	27	28	27	28	28
	5	40	40	43	45	41	39	39	38	41	41	38	38	38	38
	6	51	48	48	45	46	45	33	43	33	45	43	44	46	43
	7	51	55	51	50	55	49	49	49	48	48	48	49	51	50
	8	58	58	59	50	56	54	52	54	52	50	53	54	56	55
	9	62	64	63	71	63	58	56	58	56	59	58	58	61	60
4	5	109	97	100	162	97	87	81	90	NA	94	94	98	96	93
	6	140	141	142	162	141	133	129	129	131	129	132	135	133	133
	7	169	166	168	226	167	156	150	153	150	154	154	157	160	156
5	6	348	305	310	386	305	273	NA	301	NA	NA	304	273	298	300
	7	458	477	452	678	466	433	NA	432	NA	NA	434	433	440	433
6	7	1089	921	1015	1201	921	982	NA	963	NA	NA	973	982	963	952
	8	1466	1515	1455	1763	1493	NA	NA	1399	NA	NA	1401	NA	1412	1395

1. Comparison using CA(t, 3^p):

EWFS is compared with existing strategies using covering arrays of the form CA(t, 3^p), where the number of parameters (p) varies from 3 to 12. In contrast, the value per parameter remains constant at 3. The interaction strength (t) ranges from 2 to 6.

2. Comparison using CA(t, v^7):

This experiment compares EWFS with existing strategies using CA(t, v^7), where the number of parameters remains fixed at 7, and the number of values (v) per parameter varies from 2 to 7. The interaction strength (t) also varies from 2 to 6.

3. Comparison using CA(2, 2^p):
In this experiment, EWFS is benchmarked against existing strategies using CA(2, 2^p), where the interaction strength and value per parameter are fixed at 2, while the number of parameters (p) varies from 3 to 15, 50, and 100.
4. Comparison for higher interaction strength using CA(t, 2^{10}):
EWFS is compared with existing strategies using CA(t, 2^10), where both the number of parameters (10) and values per parameter (2) remain constant. The interaction strength (t) varies from 2 to 10 to assess performance at higher strengths.

Table 4. Test suite size performance for CA (t,v^7)

CA (t,v^7)		Pure computation strategies					AI-based strategies						
t	v	Jenny	Tconfig	PICT	IPOG-D	IPOG	QLSCA	GS	DPSO	APSO	CS	WFS	EWFS
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best
2	2	8	7	7	8	7	7	6	7	6	6	7	7
	3	16	15	16	15	15	15	14	14	15	15	15	15
	4	28	28	27	32	29	23	24	24	25	25	27	25
	5	37	40	40	45	45	34	36	34	35	37	40	37
	6	55	57	56	72	55	48	52	47	NA	NA	57	55
	7	74	76	74	91	49	64	68	64	NA	NA	78	74
3	2	14	16	15	14	16	15	12	15	15	12	13	12
	3	51	55	51	50	55	49	49	49	48	49	51	48
	4	124	112	124	114	112	112	116	112	118	117	123	119
	5	236	239	241	252	237	215	221	216	239	223	242	234
	6	400	423	413	470	420	364	374	365	NA	NA	418	405
4	2	31	36	32	40	35	31	27	34	30	27	25	24
	3	169	166	168	226	167	149	153	150	153	155	156	154
	4	517	568	529	704	614	477	486	472	472	487	513	499
5	2	57	56	57	80	60	NA	51	NA	NA	53	51	49
	3	458	477	452	678	466	NA	432	NA	NA	439	442	435
	4	1938	1792	1933	2816	1792	NA	1821	NA	NA	1845	1861	1829
6	2	87	64	72	96	64	NA	65	NA	NA	66	66	66
	3	1087	921	1015	1201	921	NA	963	NA	NA	973	955	951
	4	6127	NA	5847	5120	4096	NA	5608	NA	NA	5610	5610	5597

The result revealed that EWFS is the most efficient and scalable strategy, providing optimized configurations with fewer test cases as t increases.

The findings from the first experiment, as presented in Table 3, demonstrate that EWFS (Enhanced Wingsuit Flying Search Algorithm) consistently provides among the lowest test case counts for nearly all configurations, particularly as t and p values grow. For instance, in CA (4,5), EWFS requires 93 test cases, which is fewer than most other strategies, and similarly outperforms others at higher values, such as CA (6,8) with 1395 test cases.

Among pure computation methods, Tconfig and IPOG-D generally perform well but still fall short compared to the AI-based strategies at higher complexities. For example, in CA (6,7), TConfig requires 921 test cases, while EWFS achieves 952. Pure computation strategies like Tconfig and IPOG-D can still be practical choices for less complex scenarios, as they remain competitive up to certain interaction levels. However, they generally do not match the efficiency of AI-based methods and EWFS as t and p values increase.

Table 5. Test suite size performance for CA (2, 2p) where p varied from 3 to 15, 50, and 100

$CA(2,2^P)$	Pure computation strategies					AI-based strategies						
p	Jenny	Tconfig	PICT	IPOG	PSO	MBO	CS	FPA	ABC	PKS	WFS	EWFS
	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best
3	5	4	4	4	4	4	4	4	4	4	4	4
4	6	6	5	6	6	6	5	6	5	5	6	6
5	7	6	7	8	6	6	6	6	6	6	6	6
6	8	7	6	8	7	7	6	7	7	6	6	7
7	8	9	7	8	7	7	7	7	7	6	7	7
8	8	9	8	8	8	7	8	8	8	7	7	8
9	8	9	9	10	8	8	8	8	8	8	8	8
10	10	9	9	10	8	8	8	8	8	8	8	8
11	9	9	9	10	9	8	8	8	9	8	8	8
12	10	9	9	10	9	8	9	9	9	8	8	8
13	10	9	9	10	9	9	NA	NA	9	8	8	9
14	10	9	10	10	9	9	NA	NA	9	9	9	9
15	10	9	10	10	10	9	NA	NA	9	9	9	9
50	NA	NA	NA	NA	NA	NA	12	NA	NA	NA	12	12
100	NA	NA	NA	NA	15	NA	15	NA	NA	NA	NA	NA

Likewise, the configuration setting CA (t, v7) experiment shows interesting results, as shown in Table 4. It highlights that for higher values of t, most strategies exhibit similar performance, with pure computation methods like IPOG and TConfig being highly competitive. AI-based methods such as QLSCA, GS, and DPSO generally match the performance of the best pure computation methods. Nonetheless, EWFS also produces smaller test suite sizes than other strategies, especially for CA (4, 27), achieving 24, and CA (5, 27), achieving 49.

The findings from the third experiment are presented in Table 5. As p increases (from 9 to 15), some AI methods, especially FPA, show fluctuations in their performance, indicating potential inconsistencies when dealing with higher dimensions. EWFS demonstrates strong performance, often matching or outperforming pure computation and other AI strategies such as ABC, PKS, and WFS across most p values. It shows resilience in delivering lower test case counts, with results like 8 for p=9 and 12 for p=50.

Table 6. Test suite size performance for CA $(t,2^{10})$

$CA(t,2^{10})$	Pure computation strategies					AI-based strategies				
t	Jenny	Tconfig	PSO	CS	IMTS	FPA	IJA	LHS-JA	WFS	EWFS
	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best
2	10	9	8	8	8	8	8	8	8	8
3	18	20	17	16	16	17	NA	NA	16	16
4	39	45	37	36	38	39	NA	NA	38	36
5	87	95	82	79	80	82	79	79	84	78
6	169	183	158	157	159	160	159	159	160	157
7	311	NA	NA	NA	NA	NA	297	301	291	288
8	521	NA	NA	NA	NA	NA	502	515	500	495
9	788	NA	NA	NA	NA	NA	584	584	574	568
10	1024	NA	NA	NA	NA	NA	1024	1024	1024	1024

Table 7: Wilcoxon test for the results reported on Tables 1,2,3, and 4

Algorithm	Ranks			Test Statistics	
	EWFS>	EWFS<	EWFS =	Asymp. Sig (2-Tailed)	Null Hypothesis
Jenny	1	55	9	< 0.01	Reject
TConfig	5	45	10	< 0.01	Reject
PICT	2	48	6	< 0.01	Reject
IPOG-D	5	35	3	< 0.01	Reject
IPOG	9	37	10	0.003	Reject
QLSCA	21	5	7	< 0.01	Reject
GS	5	15	0	0.002	Reject
PKS	5	0	8	0.025	Reject
IJA	0	5	2	0.043	Reject
LHSJA	0	5	2	0.043	Reject
DPSO	19	6	5	0.005	Accept
APSO	12	7	9	0.569	Accept
CS	12	17	2	0.242	Accept
PSO	0	4	1	0.059	Accept
ABVCS	5	7	10	0.547	Accept
MBO	1	0	12	0.317	Accept
FPA	0	1	9	0.317	Accept
ABC	1	2	10	0.564	Accept
IMTS	0	3	2	0.083	Accept
GBGA	6	8	8	0.897	Accept

The ability of EWFS to consistently maintain low test case counts across various configurations highlights its reliability and scalability compared to other strategies. As shown in Table 6, other AI-based methods such as IMTS and CS perform reasonably well, particularly at moderate interaction strengths. However, they may require further enhancement to remain competitive in high-dimensional scenarios. In contrast, the consistent performance and complete availability of data for both EWFS and WFS across all configurations further support their reliability

Further investigation was conducted using the Wilcoxon signed-rank test to assess the significance of pairwise performance differences between EWFS and other strategies. The results are presented in Table 7. EWFS demonstrates statistically significant differences when compared with Jenny, TConfig, PICT, IPOG-D, IPOG, QLSCA, GS, PKS, IJA, and LHSJA, as indicated by p-values below 0.05.

Among the 10 algorithms that showed statistically significant differences, eight were outperformed by EWFS, as it produced the smallest test suite sizes in most configurations. The remaining two algorithms were QLSCA and PKS. QLSCA outperformed EWFS in certain configurations, while PKS did not show any configuration where EWFS produced better results. However, PKS had eight results identical to EWFS and five configurations where EWFS generated larger test suites. The results of the remaining 10 methods accept the null hypothesis, indicating no statistically significant difference in performance compared to EWFS. These methods include GBGA, DPSO, APSO, CS, PSO, ABVCS, MBO, FPA, and ABC, all showing p-values equal to or greater than 0.05. EWFS outperformed its counterparts in only two cases, specifically in comparing PSO and IMTS.

However, this lack of significance is likely due to the limited number of comparable test cases, as several configurations involved relatively small test suite sizes (ranging from 7 to 15), reducing the Wilcoxon test's statistical power. Additionally, not all algorithms were evaluated across all configuration settings. As shown in Table 6, several of these methods had unavailable (NA) data for higher interaction strengths ($t = 7$ to 10), resulting in fewer data points for comparison. This limitation directly impacts the ability to detect meaningful differences between EWFS and the other methods in that configuration.

6. CONCLUSION

This paper introduces the Enhanced Wingsuit Flying Search (EWFS) algorithm for test suite generation in t-way testing. The experimental results show that EWFS is highly effective and competitive, outperforming several other strategies regarding test suite size and efficiency.

The strength of EWFS lies in the integration of the Lévy flight into the original WFS algorithm. While WFS tends to focus on local exploitation and gets stuck in local optima, implementing Lévy flight helps to overcome this limitation. By introducing a random walk mechanism, Lévy flight enhances the algorithm's ability to explore the search space more effectively and escape from local optima. This contributes to the improved performance of EWFS, particularly in higher configuration systems, as shown in Tables 5 and 6. The effectiveness of this approach is further supported by the statistically significant results presented in Table 7. These results indicate that EWFS produces significant outcomes and performs on par with other metaheuristic methods.

In the context of software testing, the use of EWFS is particularly relevant due to its potential to reduce the number of test cases required for achieving full interaction coverage. This makes it highly suitable for systems involving a large number of interacting parameters. For instance, EWFS has practical potential in configuration testing of flight booking systems, network protocols, and embedded systems, where interaction faults are common and must be efficiently detected using minimal test cases. Future work will explore extending EWFS to support variable-strength t-way testing and input-output relationship testing, further enhancing its applicability in complex testing scenarios.

Acknowledgments
The author would like to acknowledge the support of the Fundamental Research Grant Scheme (FRGS) under grant number FRGS/1/2023/ICT01 /UNIMAP/02/1 from the Ministry of Higher Education Malaysia.

REFERENCES

[1] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, **Software Testing With Large Language Models: Survey, Landscape, and Vision**, *IEEE Transactions on Software Engineering*, vol. 50, no. 4, 2024.

[2] Z. Jiang *et al.*, **A Review of Software Reliability Testing Techniques**, *Journal of Computing and Information Technology*, vol. 28, no. 3, 2020.

[3] 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.

[4] N. Covic and B. Lacevic, **Wingsuit Flying Search-A Novel Global Optimization Algorithm,** *IEEE Access*, vol. 8, pp. 53883–53900, 2020.

[5] M. Karimi-Mamaghan, M. Mohammadi, B. Pasdeloup, and P. Meyer, **Learning to select operators in meta-heuristics: An integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem,** *Eur J Oper Res*, vol. 304, no. 3, 2023.

[6] 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, 2023.

[7] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, **IPOG: A general strategy for T-way software testing,** *Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems*, pp. 549–556, 2007.

[8] 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, 2008.

[9] B. Jenkins, **jenny: a pairwise testing tool,** https://burtleburtle.net/bob/math/jenny.html, 2005.

[10] J. Czerwonka, **Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios,** *Proceedings of 24th Pacific Northwest Software Quality Conference*, pp. 419–430, 2008.

[11] B. Swathi and H. Tiwari, **Integrated Pairwise Testing based Genetic Algorithm for Test Optimization,** *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 4, 2021.

[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, no. October 2023, p. 111070, 2024.

[13] K. Rajwar, K. Deep, and S. Das, **An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges,** *Artif Intell Rev*, vol. 56, no. 11, 2023.

[14] B. Morales-Castañeda, D. Zaldívar, E. Cuevas, F. Fausto, and A. Rodríguez, **A better balance in metaheuristic algorithms: Does it exist?,** *Swarm Evol Comput*, vol. 54, 2020.

[15] H. N. K. Al-Behadili, **Stochastic Local Search Algorithms for Feature Selection: A Review,** 2021.

[16] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, **Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019),** *IEEE Access*, vol. 9, 2021.

[17] J. Piri, P. Mohapatra, R. Dey, B. Acharya, V. C. Gerogiannis, and A. Kanavos, **Literature Review on Hybrid Evolutionary Approaches for Feature Selection,** 2023.

[18] K. Li, D. Li, and H. Q. Ma, **An improved discrete particle swarm optimization approach for a multi-objective optimization model of an urban logistics distribution network considering traffic congestion,** *Advances in Production Engineering And Management*, vol. 18, no. 2, 2023.

[19] Y. Song, Y. Liu, H. Chen, and W. Deng, **A Multi-Strategy Adaptive Particle Swarm Optimization Algorithm for Solving Optimization Problem,** *Electronics (Switzerland)*, vol. 12, no. 3, 2023.

[20] S. Esfandyari and V. Rafe, **A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy,** *Inf Softw Technol*, vol. 94, 2018.

[21] X. S. Yang and S. Deb, **Engineering optimisation by cuckoo search,** *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, no. 4, 2010.

[22] A. B. Nasser, A. Alsewari, and K. Z. Zamli, **Learning Cuckoo Search Strategy for t-way Test Generation. Springer Singapore,** 2018.

[23] 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, pp. 1–29, 2018.

[24] A. K. Alazzawi, H. M. Rais, and S. Basri, **Parameters tuning of hybrid artificial bee colony search-based strategy for t-way testing,** *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 5s, 2019.

[25] P. E. Mergos and X. S. Yang, **Flower pollination algorithm parameters tuning,** *Soft Comput*, vol. 25, no. 22, 2021.

[26] A. B. Nasser, A. R. A. Alsewari, N. M. Tairan, and K. Z. Zamli, **Pairwise test data generation based on flower pollination algorithm,** *Malaysian Journal of Computer Science*, vol. 30, no. 3, pp. 242–257, 2017.

[27] S. Kouka, S. N. Makhadmeh, M. A. Al-Betar, L. M. Dalbah, and M. Nachouki, **Recent Applications and Advances of Migrating Birds Optimization,** 2024.

[28] H. L. Zakaria, K. Z. Zamli, and F. Din, **Hybrid Migrating Birds Optimization Strategy for t-way Test Suite Generation,** *J Phys Conf Ser*, vol. 1830, no. 1, 2021.

[29] J. Torres-Jimenez and J. C. Perez-Torres, **A greedy algorithm to construct covering arrays using a graph representation,** *Inf Sci (N Y)*, vol. 477, 2019.

[30] A. A. B. Homaid, A. A. Alsewari, A. K. Alazzawi, and K. Z. Zamli, **A Kidney Algorithm for Pairwise Test Suite Generation,** *Adv Sci Lett*, vol. 24, no. 10, pp. 7284–7289, 2018.

[31] A. B. Nasser, F. Hujainah, A. A. Al-Sewari, and K. Z. Zamli, **An improved jaya algorithm-based strategy for t-way test suite generation,** in *Advances in Intelligent Systems and Computing*, 2020.

[32] A. B. Nasser, A. S. H. Abdul-Qawy, N. Abdullah, F. Hujainah, K. Z. Zamli, and W. A. H. M. Ghanem, **Latin Hypercube Sampling Jaya Algorithm based Strategy for T-way Test Suite Generation,** in *ACM International Conference Proceeding Series*, 2020.

[33] Ramblers., **How to Wingsuit: How Wingsuits Work,** https:// www.ramblers.com.au/blog/how-to-wingsuit-how-wingsuits-work/.

[34] 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.

[35] J. Li, Q. An, H. Lei, Q. Deng, and G. G. Wang, **Survey of Lévy Flight-Based Metaheuristics for Optimization,** *Mathematics*, vol. 10, no. 15, 2022.

[36] F. El Asri, C. Tajani, and H. Fakhouri, **Investigation of ant colony optimization with Lévy flight technique for a class of stochastic combinatorial optimization problem,** *Mathematical Modeling and Computing*, vol. 10, no. 4, 2023.

[37] A. R. A. Alsewari and K. Z. Zamli**, Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support,** *Inf Softw Technol*, vol. 54, no. 6, pp. 553–568, 2012.

[38] B. Ahmed, **Generating Pairwise Combinatorial Interaction Test Suites Using Single Objective Dragonfly Optimisation Algorithm,** *Journal of Zankoy Sulaimani - Part A*, vol. 19, no. 1, 2017.

[39] J. M. Altmemi, R. R. Othman, and R. Ahmad, **SCAVS: Implement Sine Cosine Algorithm for generating Variable t-way test suite,** *IOP Conf Ser Mater Sci Eng*, vol. 917, no. 1, 2020.