Computers, Materials & Continua
DOI: 10.32604/cmc.2022.026310
Article

Whale Optimization Algorithm Strategies for Higher Interaction Strength T-Way Testing

Ali Abdullah Hassan[1,*], Salwani Abdullah[1], Kamal Z. Zamli[2] and Rozilawati Razali[1]

[1]Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, 43600, Selangor, Malaysia
[2]Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang, Pekan, 26600, Pahang, Malaysia
*Corresponding Author: Ali Abdullah Hassan. Email: ali87hassan@gmail.com
Received: 22 December 2021; Accepted: 10 March 2022

Abstract: Much of our daily tasks have been computerized by machines and sensors communicating with each other in real-time. There is a reasonable risk that something could go wrong because there are a lot of sensors producing a lot of data. Combinatorial testing (CT) can be used in this case to reduce risks and ensure conformance to specifications. Numerous existing meta-heuristic-based solutions aim to assist the test suite generation for combinatorial testing, also known as t-way testing (where t indicates the interaction strength), viewed as an optimization problem. Much previous research, while helpful, only investigated a small number of interaction strengths up to $t = 6$. For lightweight applications, research has demonstrated good fault-finding ability. However, the number of interaction strengths considered must be higher in the case of interactions that generate large amounts of data. Due to resource restrictions and the combinatorial explosion challenge, little work has been done to produce high-order interaction strength. In this context, the Whale Optimization Algorithm (WOA) is proposed to generate high-order interaction strength. To ensure that WOA conquers premature convergence and avoids local optima for large search spaces (owing to high-order interaction), three variants of WOA have been developed, namely Structurally Modified Whale Optimization Algorithm (SWOA), Tolerance Whale Optimization Algorithm (TWOA), and Tolerance Structurally Modified Whale Optimization Algorithm (TSWOA). Our experiments show that the third strategy gives the best performance and is comparable to existing state-of-the-arts based strategies.

Keywords: Software testing; optimization problem; swarm intelligence algorithm; combinatorial optimization; IoT

1 Introduction

The Internet of Things (IoT) is a groundbreaking technology that allows users to connect with things (objects) via the Internet by granting things the ability to feel the environment and communicate with other objects and people. IoT has encouraged an ambitious and valuable new generation of services. Nowadays, cities are becoming smarter by implementing intelligent systems such as traffic control systems, water management systems, energy management systems, public transportation systems, to name a few. However, the design of IoT systems, software applications, and services make testing a tedious process [1]. This is because of its homogeneous and decentralized nature and tremendous data collection and processing [1].

Combinatorial testing (CT) is used in this scenario to reduce the number of test cases. CT uses a few numbers of test cases to examine the interactions among the software's configurations (parameters). "a flaw is usually caused by interactions among a small number (say *t*) of parameters," says the idea [2]. As a result, it's crucial to look at all possible combinations of each *t* parameter since software system errors emerge when two or more system inputs interact [3].

There are two types of t-way testing procedures available: pure computational-based and artificial intelligence (AI)-based. Jenny [4], Test Configurations (TConfig) [5], Pairwise Independent Combinatorial Tool (PICT) [6], Generalized in-parameter-order strategy (IPOG) [7], and IPOG for multi-way testing (IPOG-D) [8] are pure computational-based approaches aiming at constructing a test set from the start based on the generalization of the Orthogonal Array (OA) to construct the test suite [8]. On the other hand, AI-based techniques such as Genetic algorithm (GA) [9], Particle Swarm Optimization algorithm (PSO) [10], and Ant Colony algorithm (ACA) [11] start by generating test cases from a pre-existing test set (i.e., the interaction elements, also known as the t-tuple table) and continue until the test suite is built.

AI-based techniques produce superior results by containing a few test cases [12]. On the other hand, pure computational-based strategies offer a comprehensive list of all conceivable combinations to be covered. Consequently, the test suite size results may not be optimal [8]. Yet, these methods are often relatively expeditious at producing test cases [13].

The number of parameters that interact with each other is called interaction strength. Two-way testing suggests an interaction between two parameters, and three-way testing means interaction between three parameters, etc. The previous studies have shown that most of the flaws in a standard software system may be identified by small interaction strengths (i.e., less than or equal to three) [10]. However, with the emergence of IoT technology, little-known work has concentrated on generating high order interaction strength (i.e., $t > 6$) due to resource constraints and combinatorial explosion problems. Additionally, further studies have indicated the need for higher strengths, particularly in the case of sophisticated software systems and IoT systems [12]. This is due to a massive increase in the size and configuration of the software systems, which gives rise to a more significant number of parameters with various interactions. Therefore, the following two criteria are regarded to assess the strategy: (1) the size of the test suite and (2) the support of higher interaction strength.

The significant contribution of this study is the introduction of three variants of the Whale Optimization Algorithm (WOA). The first variant is to modify the structure of WOA and the second one is to use the acceptance probability of the Exponential Monte-Carlo algorithm (EMC) [14] to accept the worst solutions. The third variant is the combination of the first and the second variant.

The remainder of the article has been organized as follows: Section 2 illustrates the background of t-way testing using an example. Section 3 reviews the previous related work. Section 4 discusses the origin of WOA and its advantages and disadvantages. Section 5 introduces the proposed method, which involves the three variants of WOA. Section 6 evaluates the three variants, while Section 7 evaluates and compares the best variant among the three with the existing state-of-the-art strategies. Threats to validity are discussed in Section 8. Finally, Section 9 presents the conclusion.

2 Combinatorial Testing Background

Often, the testers have to work within a strict schedule. Therefore, the testing techniques must be considerably effective to attain maximum test coverage and a high fault discovery rate. T-way testing, also known as combinatorial testing (CT), delivers the afore-stated features [3]. CT is a method for examining all the possible discrete combinations of the software system parameters [3]. T-way testing has adopted particular Design of Experiments (DoE) methods, including the Covering Array (CA).

To produce balanced t-tuples, CAs generalize orthogonal arrays (OAs) by requiring that the t-tuples be covered at least once instead of a specified number of times [15]. Mathematically, a tuple is a limited and arranged list of items (i.e., the values of the parameter interactions). T-tuples are a collection of t elements (or arranged lists). CA (N; t, v^p) is a combinatorial object with p as the parameters (i.e., system configurations, input data, or both), N as the number of test cases generated, v as the parameter values, and t as the interaction strength.

Test suite minimization is the main objective of t-way testing. A test suite is a collection of test cases meant to examine a software system to demonstrate that the system has a particular set of behaviors. T-way testing is performed by choosing a system configuration combination, which becomes a test case. Subsequently, a test suite is created by collecting test cases for the software system.

Consider the example of the washing machine control panel application as shown in Fig. 1. The washing machine control panel application has four primary configurations/parameters, i.e., cycle function, temperature, soil, and spin. Let's say that all the parameters take four values (i.e., Cycle Function = {heavy duty, normal, fast wash, delicates}, Temperature = {hot, warm, cold, tap cold}, Soil = {heavy, normal, light, extra light}, Spin = {high, medium, low, no spin}).

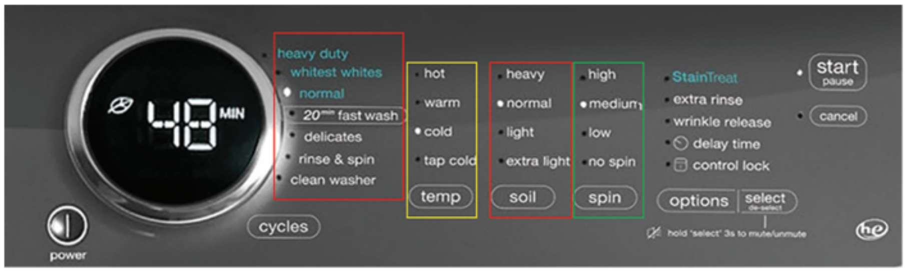

Figure 1: Washing machine control panel application

The covering array for the washing machine example shown in Fig. 1 above can be represented as CA (N; 2, 4^4), assuming the interaction strength is $t = 2$. The exhaustive test for the washing machine example required $4 \times 4 \times 4 \times 4 = 256$ test cases to cover the configurations. Meanwhile, when a meta-heuristic approach (e.g., TSWOA) was applied in 2-way testing, just 12 test cases were generated to cover all of the configurations in the washing machine example.

3 Related Work

T-way testing is classified as a discrete optimization problem because selecting the values in any particular test case is absolute. Stardom was the first to develop an AI-based strategy [9]. Stardom used tournament selection to implement GA, which started with a random generation of test cases that were partitioned and mated. The offspring were then created using the crossover procedure [9].

Shiba et al. built on Stardom's work to produce GA support 3-way testing and used the Ant Colony Algorithm (ACA) to construct CA with an interaction strength $t = 3$ [11]. Genetic Strategy (GS) is a version of GA that modifies the crossover and mutation operators to provide uniform and variable CAs and higher interaction strengths of up to twenty (i.e., $t = 20$) [13]. Furthermore, GS parameters, including population size, crossover, and mutation rates, were fine-tuned to enhance efficiency and performance [13].

PSTG [16] was the first AI-based technique to offer interaction strength *i* of up to six. Discrete Particle Swarm Optimization (DPSO) [17] is a PSO version with a schema based on divided search space and uses a separated particle swarm as its foundation. PSO performs better now that two additional procedures have been added: (1) particles are reinitialized, and (2) *gbest* (the best test case found) is also examined. DPSO additionally includes parameter adjustment instructions. Compared to traditional PSO and other known evolutionary algorithms, DPSO has produced better outcomes [17].

Harmony Search Strategy (HSS) is an AI-based strategy implemented to generate a test suite. HSS added a new test case to the CA (test suite) at each iteration until all t-interactions were covered and the CA was complete. HSS could support higher interaction strengths of up to fifteen [12]. Another AI-based technique used to generate CA with a six-interaction strength is Cuckoo Search (CS) [18]. The objective of introducing CS to t-way testing was to reduce the time spent searching for test cases. PSTG and CS produce similar results, with the latter being fast. Gravitational Search Test Generator (GSTG) was proposed to support higher interaction strength of up to ten [19]. GSTG is based on GSA, a population-based metaheuristic algorithm with stochastic search behavior [19].

Adaptive particle swarm optimization (APSO) is a hybridization of the Mamdani-type fuzzy inference system (FIS) and particle swarm optimization [20]. FIS is used to optimize the parameters of PSO. For most cases, APSO outperforms DPSO and CS [20]. Zamli et al. proposed FPSO [21], the same hybridization but with a different technique and functionality. PSO's global and local search processes were subjected to fuzzy adaptive selection as part of FPSO's approach. Both hybridizations produced CAs with interaction strengths up to four (i.e., $t \leq 4$), and FPSO outperformed DPSO in the average outcome in several situations. Another PSO hybridization is the hybrid artificial bee colony strategy (HABCSm), which uses PSO to enhance the standard ABC algorithm's performance [22]. HABCSm is a t-way testing strategy that uses the Hybrid Artificial Bee Colony (HABC) algorithm as its primary application for producing a final test set and the Hamming distance as the last selection criterion to aid in discovering new solutions [22].

The Q-learning sine cosine algorithm (QLSCA) combines the sine-cosine algorithm (SCA) and the Q-learning algorithm [23]. The Q-learning technique replaces the SCA switching probability, which uses a penalty and reward mechanism to find the optimum procedure during runtime. In most cases, QLSCA beat many previous techniques by generating the best average performance [23]. The Graph-Based Greedy Algorithm (GBGA) is a competitive greedy algorithm that constructs CAs using a graph representation. It can support up to six (i.e., $t = 6$) interaction strengths [24]. The Artificial Bee Colony Strategy (ABCVS) is a two-way generation strategy based on the Artificial Bee Colony (ABC) algorithm for a uniform and variable strength test suite [25]. ABCVS can support higher interaction strengths up to six (i.e., $t = 6$) [25]. Several scholars have recommended hybridization to boost strategies by improving quality, diversifying solutions, and addressing the frequent metaheuristic (AI-based) limitations. The purpose of hybridization in t-way testing is to reduce the test suite.

Summing up, all the previous works on t-way testing have usefully contributed to improving the current state-of-the-art on interaction t-way test case generation and potentially broadening its applications in line with the emergence of the IR4.0 agenda. Nonetheless, The No Free Lunch Theorem [26] states that no single metaheuristic can outperform others in the context of AI-based approaches to meta-heuristics, as some algorithms perform better than others on particular types of optimization problems. Our research examines how well WOA and its variants perform for general high strength interaction.

4 Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) is a contemporary metaheuristic algorithm inspired by nature [27]. WOA is a swarm-based method that mimics humpback whale hunting behavior. Humpback whales are intelligent creatures who have developed an advanced manner of cooperating with one another. They employ a unique tracking procedure known as bubble-net feeding, as shown in Fig. 2. This procedure creates peculiar bubbles along a circle or '9'-shaped path. The whales hunt near the surface and catch the victim in a net of bubbles.

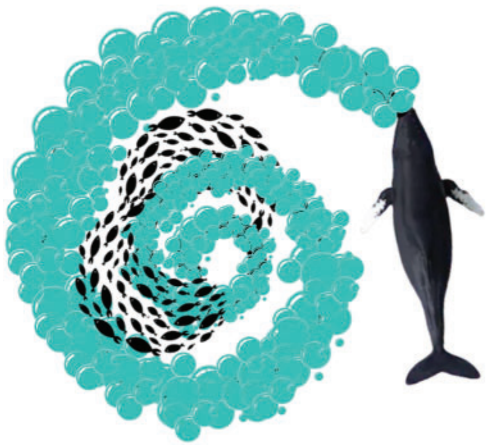

Figure 2: Bubble-net hunting behavior of humpback whales. Source [28]

The exploitation phase and the exploration phase are the two phases of WOA. The exploitation phase uses the encircling-a-prey approach and the spiral bubble-net attacking method, while the exploration phase entails randomly seeking a victim. WOA's mathematical model can be summarized as follows:

4.1 Exploitation Phase

The first approach is encircling-a-prey, in which humpback whales identify the victim's location and then encircle it. The target victim is assumed to be the current best candidate solution in WOA. After that, the best search agent is found, and the remaining search agents try to relocate their locations in its direction. In other words, it updates the whale's movement (location) around the victim, which can be described in a mathematical form as follows:

$$D = CX^{*} - X(t) \tag{1}$$

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

where t denotes the current iteration, X^{*} is the best solution found thus far, and X denotes the current solution. The coefficients A and C are calculated as follows:

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

$$C = 2.r \tag{4}$$

where a is linearly decreased from 2 to 0 along the iterations trajectory as displayed in Eq. 5, and r is a random number between 0 and 1.

$$a = 2 - t\frac{2}{MaxIter} \tag{5}$$

The second method is the Bubble-net attacking method, which comprises the shrinking encircling mechanism and the spiral updating position mechanism. The shrinking encircling mechanism is carried out by lowering a value in Eq. (3). Thus, the new position of a search agent is between the agent's actual position and the position of the existing best agent. The spiral updating position mechanism involves calculating the distance between the current solution (whale) and the best solution (victim) by applying the spiral Equation as follows:

$$X(t+1) = D'.e^{bl}.\cos 2\pi l + x^{*}(t) \tag{6}$$

where D' is the distance between the whale and the victim, b is a constant for defining the shape of the logarithmic spiral, and l is a random value between -1 and 1.

Humpback whales practice both techniques (encircling-a-prey and spiral-shaped) simultaneously. A 50% probability is introduced to model this behavior to select one of the techniques to update the whales' location during the search. The mathematical model is outlined as follows:

$$X(t+1) = \begin{cases} \textit{Encircling Equation } (2), \textit{if } p < 0.5 \\ \textit{Spiral–shaped path Equation } (6), \textit{if } p \geq 0.5 \end{cases} \tag{7}$$

where p is a random number in [0,1].

4.2 Exploration Phase

WOA considers a global search. The whales search at random based on their proximity to one another. As a result, rather than relying on the best search agent found thus far, the position of a search agent is adjusted at random. When A's random values are higher than one, this strategy drives the search agent to move away from a reference whale (best solution). As seen in Algorithm 1, this approach stresses global search and allows the WOA algorithm to explore. The mathematical model for the exploration phase is as follows:

$$D = C.X_{rand} - X \tag{8}$$

$$X(t+1) = X_{rand} - A.D \tag{9}$$

Algorithm 1: Pseudo-code of the WOA algorithm

```
1: Initialize the whales population $X_i$ (i = 1, 2, ..., n)
2: Calculate the fitness of each solution
3: $X^*$ = the best search agent.
4: while i < maximum number of iteration do
5:     for every solution do
6:         Update $a$, $A$, $C$, $l$, and $p$
7:         if ( $p < 0.5$) then
8:             if ( $|A| < 1$ ) then
9:                 Update the position of the current solution using Eq (2)
10:            else if ( $|A| > 1$) then
11:                Random solution is generated
12:                Update the position of the current solution using Eq (9)
13:        else if ( $p >= 0.5$ ) then
14:            Update the position of the current solution using Eq (6)
15:        end
16:        Check whether any solution exceeds the search space and adjust it
17:        Compute the fitness of every solution
18:        Update$X^*$ if there is a better solution
19:        $t = t + 1$
20:    end
21: return $X^*$
```

5 Proposed Method

The WOA algorithm and its variations have been extensively utilized to address various problems in various disciplines. WOA has been improved or modified in some domains to enhance solution quality and performance and address flaws such as premature convergence [29] and getting stuck in local optima [28,30].

Therefore, in this study, the following three variants of WOA for t-way test suite generation were proposed:

5.1 Structurally Modified Whale Optimization Algorithm (SWOA)

As stated earlier and illustrated in Fig. 3, WOA has two phases, exploration and exploitation, controlled by the controlling parameter A. The exploration phase is the random generation to enforce the global search. In contrast, the exploitation phase is represented by the shrinking and spiral-shaped path mechanisms. The randomization is used only in the shrinking mechanism, as visible in Fig. 3. This implies that WOA encourages greater exploitation, which leads to rapid convergence to a potentially non-optimal solution (i.e., premature convergence). Hence, SWOA was introduced to overcome the premature convergence issue by restructuring the WOA so that the decision of exploration and exploitation is undertaken first and controlled by A. Then, a probability of 50% chance is used to select one of the mechanisms, as illustrated in Fig. 4 and Algorithm 2.

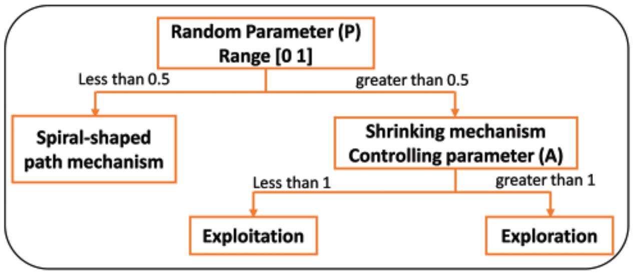

Figure 3: The layout structure of WOA

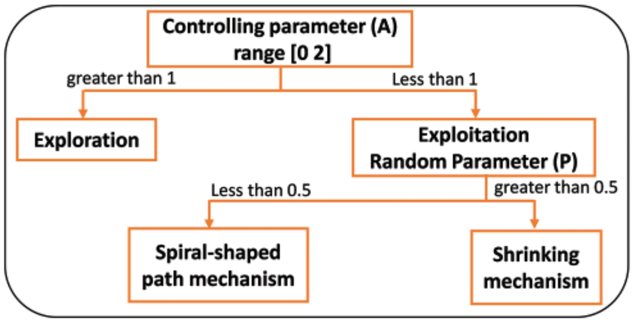

Figure 4: The layout structure of SWOA

Algorithm 2: Pseudo-code of the SWOA algorithm

```
 1: Initialize the whales population $X_i$ (i = 1, 2, ..., n)
 2: Calculate the fitness of each solution
 3: $X^*$ = the best search agent.
 4: while i < maximum number of iteration do
 5:     for every solution do
 6:         Update $a$, $A$, $C$, $l$, and $p$
 7:         if ( $|A| < 1$) then
 8:             if ( $p < 0.5$ ) then
 9:                 Update the position of the current solution using Eq (2)
10:             else if ( $p >= 0.5$) then
11:                 Update the position of the current solution using Eq (6)
12:         else if ( $|A| > 1$) then
13:                 Random solution is generated
14:                 Update the position of the current solution using Eq (9)
15:         end
16:     Check whether any solution exceeds the search space and adjust it
17:     Compute the fitness of every solution
18:     Update $X^*$ if there is a better solution
19:     $t = t + 1$
20: end
21: return $X^*$
```

5.2 *Tolerance Whale Optimization Algorithm (TWOA)*

It may be observed from Algorithm 1 that WOA always accepts the better solution [27]. As a consequence, WOA may get trapped in local optima. The second variant TWOA was used to resolve this issue, employing acceptance probability to avoid local optima. This acceptance probability was exercised in an Exponential Monte-Carlo algorithm (EMC-FC) [31], which allowed the worst solution to be accepted at a certain degree. The acceptance probability is calculated as follows:

$$e^{\frac{-\theta}{\lambda}} \tag{10}$$

where $\theta = \delta * t$, δ is the difference between the current and the best solution, t is the iteration counter (i.e., the current iteration) and λ is a successive non-improvement counter.

One may note that, as the number of iterations t increments, the probability of choosing a worse solution reduces. But, if for specific successive iterations, no improvement occurs, the likelihood of tolerating a worse solution would increment depending on the objective function of the current solution and the number of iterations. In other words, if δ is small or λ is high, the selection of a worse solution seems probable. Algorithm 3 demonstrates the pseudo-code of TWOA and illustrates how the acceptance probability is employed.

5.3 The Combination of SWOA and TWOA (TSWOA)

The last variant is the combination of the SWOA and TWOA, as Algorithm 4 shows.

6 Experimental Results

This section reports the assessment of the efficiency of the WOA variants and the selection of the best one among the three, which would subsequently be benchmarked with the existing strategies (the comparison process described in the next section). The efficiency was measured in terms of the size of the CA (i.e., test suite size). Another measurement has been provided, which is the convergence behavior of the proposed WOA variants.

This experiment conducted a preliminary test on the algorithm's parameters sensitivity to determine the population size and the maximum iteration number. The covering array CA (N; 2, 5^7) was chosen as a case study to tune the parameters. The justification for embracing this covering array is that many AI-based approaches are tuned using the same covering array [12]. The best result was obtained for a population size of 50, while the maximum number of iterations was 100. In this context, each variant of WOA was subjected to renowned CA configurations, as depicted in Tab. 1.

The CA configurations in Tab. 1 can be divided into four groups: the first one was CA(2, $3^{3\text{-}13}$) where $t = 2$ and $v = 3$ while p varies from 3 to 13; the second one was CA(3, $3^{4\text{-}9}$) where $t = 3$ and $v = 3$ while p varies from 4 to 9; the third one was CA(4, $3^{5\text{-}7}$) where $t = 4$ and $v = 3$ while p ranges from 5 to 7; and the last one was CA(5-6, 2^{10}) where $v = 2$ and $p = 10$ while t varies from 5 to 6.

The proposed methods were run 31 times, and the best and average results were considered. The results presented in Tab. 1 revealed that the enhanced variants of WOA outperformed the original one in most cases. In addition, the results indicated that TSWOA generated better output compared to the other variants. Both SWOA and TWOA produced better results in terms of best and average results than the original one. In comparison, the performance of SWOA was close to that of TWOA. SWOA performed slightly better than TWOA in certain cases.

Algorithm 3: Pseudo-code of the TWOA algorithm

```
1: Initialize the whales population $X_i$ (i = 1, 2, ..., n)
2: Calculate the fitness of each solution
3: $X^*$ = the best search agent.
4: while i <  maximum number of iteration do
5:     for every solution do
6:         Update $a$, $A$, $C$, $l$, and $p$
7:         if ( $p < 0.5$) then
8:             if ( $|A| < 1$ ) then
9:                 Update the position of the current solution using Eq (2)
10:            else if ( $|A| > 1$) then
11:                Random solution is generated
12:                Update the position of the current solution using Eq (9)
13:        else if ( $p >= 0.5$ ) then
14:            Update the position of the current solution using Eq (6)
15:    end
16:    Check whether any solution exceeds the search space and adjust it
17:    Compute the fitness of every solution
18:    $X_{current}$ = the best solution
19:    if ( $X_{current}$ better than $X^*$ ) then
20:        $X^* = X_{current}$
21:        $\lambda = 1$
22:        non-improvement = 0
23:    else
24:        $\delta = X_{current} - X^*$
25:        Generate a random number randNum in [0 1]
26:        if ( randNum $\leq e^{-\delta * t / \lambda}$ ) then
27:            $X^* = X_{current}$
28:            $\lambda = 1$
29:            non-improvement = 0
30:        else
31:            non-improvement ++
32:            if ( non-improvement == Max-non-improvement) then
33:                $\lambda$ ++
34:                non-improvement = 0
35:    $t = t + 1$
36: end
37: return $X^*$
```

Algorithm 4: Pseudo-code of the TSWOA algorithm

```
1: Initialize the whales population $X_i$ (i = 1, 2, ..., n)
2: Calculate the fitness of each solution
3: $X^*$ = the best search agent.
4: while i < maximum number of iteration do
5:     for every solution do
6:         Update $a$, $A$, $C$, $l$, and $p$
7:         if ( $|A| < 1$ ) then
8:             if ( $p < 0.5$ ) then
9:                 Update the position of the current solution using Eq (2)
10:            else if ( $p >= 0.5$ ) then
11:                Update the position of the current solution using Eq (6)
12:        else if ( $|A| > 1$ ) then
13:                Random solution is generated
14:                Update the position of the current solution using Eq (9)
15:    end
16:      Check whether any solution exceeds the search space and adjust it
17:      Compute the fitness of every solution
18:      $X_{current}$ = the best solution
19:      if ( $X_{current}$ better than $X^*$ ) then
20:          $X^* = X_{current}$
21:          $\lambda = 1$
22:          non-improvement = 0
23:      else
24:          $\delta = X_{current} - X^*$
25:          Generate a random number randNum in [0 1]
26:          if ( randNum $\le e^{-\delta * t / \lambda}$ ) then
27:              $X^* = X_{current}$
28:              $\lambda = 1$
29:              non-improvement = 0
30:          else
31:              non-improvement ++
32:              if ( non-improvement == Max-non-improvement) then
33:                  $\lambda$ ++
34:                  non-improvement = 0
35:      $t = t + 1$
36: end
37: return $X^*$
```

Table 1: Assessment of the proposed WOA variants

Configuration CA (t, v^p)	WOA		MWOA		TWOA		TMWOA	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
CA (2, 3^3)	**9**	9.74	**9**	9.67	**9**	9.29	**9**	9.41
CA (2, 3^4)	**9**	10.29	**9**	10.22	**9**	9.96	**9**	9.80
CA (2, 3^5)	**11**	12.71	**11**	12.64	**11**	12.67	**11**	12.12
CA (2, 3^6)	**13**	14.55	14	14.74	**13**	14.58	**13**	14.09
CA (2, 3^7)	15	15.16	**14**	15.12	15	15.22	**14**	14.77
CA (2, 3^8)	**15**	15.77	**15**	15.93	**15**	15.64	**15**	15.58

(Continued)

Table 1: Continued

Configuration CA (t, v^p)	WOA		MWOA		TWOA		TMWOA	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
CA (2, 3^9)	**15**	16.65	**15**	16.83	**15**	16.38	**15**	16.09
CA (2, 3^{10})	**16**	17.71	17	17.45	**16**	17.41	**16**	17.12
CA (2, 3^{11})	17	18.00	**16**	17.70	**16**	17.67	**16**	17.58
CA (2, 3^{12})	18	18.64	**17**	18.25	**17**	18.06	**17**	18.00
CA (2, 3^{13})	18	19.29	**18**	18.96	**18**	19.12	**18**	18.83
CA (3, 3^4)	28	31.03	**27**	30.35	28	31.45	**27**	30.29
CA (3, 3^5)	38	40.06	**37**	39.70	38	40.06	**37**	39.70
CA (3, 3^6)	41	45.32	41	45.09	40	44.74	**39**	44.45
CA (3, 3^7)	49	50.48	**48**	50.35	**48**	50.41	**48**	50.06
CA (3, 3^8)	**52**	54.74	53	54.48	53	54.32	**52**	54.38
CA (3, 3^9)	57	58.26	**56**	58.19	57	58.25	**56**	58.03
CA (4, 3^5)	93	99.58	**90**	98.54	92	98.83	**90**	97.93
CA (4, 3^6)	132	135.38	131	134.41	**128**	133.90	**128**	133.25
CA (4, 3^7)	**151**	157.13	153	155.96	154	157.00	**151**	156.48
CA (5, 2^{10})	**72**	82.13	75	82.16	73	81.58	**72**	80.12
CA (6, 2^{10})	156	158.61	**153**	158.67	156	159.09	**153**	157.90

Next, the convergence behavior of the proposed WOA variants was analyzed. Two CAs were selected for the analysis: CA (N; 2, 3^{13}) and CA (N; 4, 3^5). Figs. 5 and 6 depict the convergence behavior of the WOA variants. It may be noticed that TSWOA managed to converge faster than the other variants. Moreover, the enhanced variants SWOA and TWOA converge faster than the original WOA. Furthermore, Figs. 5 and 6 and the results in Tab. 1 indicated that TSWOA could escape the local optima effectively.

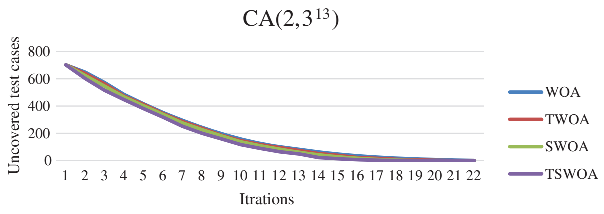

Figure 5: The convergence behavior of WOA variants for CA (N; 2; 3^{13})

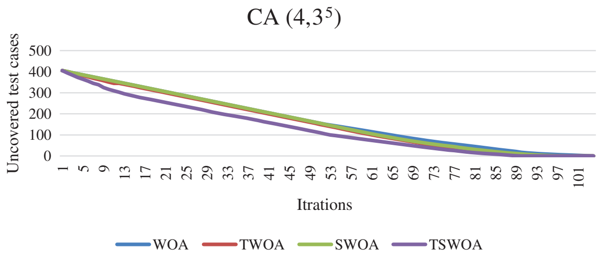

Figure 6: The convergence behavior of WOA variants for CA (N; 2; 3^5)

Statistical analysis was performed to ensure that TSWOA was superior to the other WOA variations, namely the Wilcoxon signed-rank test, a non-parametric test for matched or coupled data concentrated at differential ratings. However, it also considers the extent of the observed differences in response to evaluating the signs of the differences. The reason for utilizing the Wilcoxon signed-rank test is that it can inform if there is a significant difference between the two results.

Wilcoxon's signed-rank test produced two factors. The first one is Asymp. Sig. (2-tailed) and Z, which are statistical tests that indicate the difference between two groups. If the value of Asymp. Sig. (2-tailed) was smaller than 0.05, it implied a significant difference between the two groups. Although the value of Z is not relevant to this study, the value has nonetheless been provided in the report. The second factor is the ranking, which ranks which values are more significant than, equal to, or less than comparable values.

In all tables presenting the statistical results, in the part of the rank, "TSWOA <" indicates the number of cases TSWOA generated with a smaller size of CA compared to the other variants (i.e., WOA, SWOA, and TWOA). In other words, the number of times TSWOA generated better results. Similarly, "TSWOA =" indicated the number of times the results were the same, while "TSWOA >" represented the number of times TSWOA produced the worst results.

Tabs. 2 and 3 present the consequence of the Wilcoxon test on the results reported in Tab. 1. Tabs. 2 and 3 revealed that TSWOA generated significantly different outcomes than the other WOA variants, which confirms the superiority of TSWOA over the other WOA variants.

Table 2: Wilcoxon test of the best results reported in Tab. 1

	Ranks			Test statistics	
	TMWOA >	TMWOA <	TMWOA =	Z	Asymp. Sig. (2-tailed)
WOA	0	11	11	−3.020	0.003
MWOA	0	7	15	−2.392	0.017
TWOA	0	10	12	−2.913	0.004

Table 3: Wilcoxon test of the average results reported in Tab. 1

	Ranks			Test statistics	
	TMWOA >	TMWOA <	TMWOA =	Z	Asymp. Sig. (2-tailed)
WOA	0	22	0	−4.108	0.000
MWOA	1	20	1	−3.546	0.000
TWOA	2	20	0	−3.883	0.000

7 Benchmarking with Existing Strategies

In order to evaluate TSWOA's performance, it was compared to other existing techniques in terms of CA size. The experiments were divided into two parts. The first one has been divided again into the following two well-known datasets:

1. Comparing TSWOA with the currently available strategies using CA (t, v^7), the number of parameters remained constant, and their values varied. In addition, the interaction strength t ranged from 2 to 6.
2. Comparing TSWOA with the existing strategies using CA (t, 3^P), the number of parameters was varied, and their values remained constant. In addition, the interaction strength t varied from 2 to 6.

The second part of the experiments was conducted for higher interaction strength, and two datasets were subjected to experimentation. The first one was CA (t, 2^{10}), where the number of parameters and their values remained constant, and t varied from 2 to 10. The second one was the CAs reported in Tab. 9, where t ranged from 2 to 20. In addition, the Wilcoxon signed-rank test was performed on all the reported results.

In Tab. 4, the configurations of CA (t, 3^P) were adopted, where t varied as $2 \leq t \leq 6$, p varied as $3 \leq p \leq 12$, and v remained constant at $v = 3$; the results were reported in terms of the best and average among the 31 runs. The results revealed that TSWOA outperformed all the pure computational strategies and most AI-based strategies, including GBGA, GS, APSO, CS, and ABSVS. Moreover, TSWOA produced competitive results compared to QLSCA and DPSO strategies.

Table 4: Test suite size performance for CA (t, 3^P) where P varied from 3 to 12 and t varied from 2 to 6

CA (t, 3^P)		Pure computation strategies					AI-based strategies								
t	p	Jenny	TConfig	PICT	IPOG-D	IPOG	GBGA	QLSCA	GS	DPSO	APSO	CS	ABCVS	TMWOA	
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Mean
2	3	**9**	10	10	15	**9**	**9**	**9**	**9**	NS	**9**	**9**	**9**	**9**	9.41
	4	13	10	13	15	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	9.80
	5	14	14	13	15	15	12	**11**	**11**	**11**	**11**	**11**	**11**	**11**	12.12
	6	15	15	14	15	15	14	14	13	14	**12**	13	13	13	14.09
	7	16	15	16	15	15	15	**14**	**14**	15	15	**14**	15	**14**	14.77
	8	17	17	16	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	15.58
	9	18	17	17	**15**	**15**	16	**15**	**15**	**15**	16	**15**	16	**15**	16.09
	10	19	17	18	21	**15**	16	**15**	16	16	17	17	17	16	17.12
	11	17	20	18	21	17	17	**16**	**16**	17	NS	18	17	**16**	17.58
	12	19	20	19	21	21	18	**16**	**16**	**16**	NS	18	18	17	18.00
3	4	34	32	34	**27**	32	29	**27**	**27**	NS	**27**	28	**27**	**27**	30.29
	5	40	40	43	45	41	39	39	38	41	41	38	38	**37**	39.70
	6	51	48	48	45	46	45	**33**	43	**33**	45	43	44	39	44.45
	7	51	55	51	50	55	49	49	49	**48**	**48**	**48**	49	**48**	50.06
	8	58	58	59	**50**	56	54	52	54	52	**50**	53	54	52	54.38
	9	62	64	63	71	63	58	**56**	58	**56**	59	58	58	**56**	58.03
4	5	109	97	100	162	97	87	**81**	90	NS	94	94	98	90	97.93
	6	140	141	142	162	141	133	129	129	131	129	132	135	**128**	133.25
	7	169	166	168	226	167	156	**150**	153	**150**	154	154	157	151	156.48

(Continued)

Table 4: Continued

CA (t, 3^P)		Pure computation strategies					AI-based strategies								
t	p	Jenny	TConfig	PICT	IPOG-D	IPOG	GBGA	QLSCA	GS	DPSO	APSO	CS	ABCVS	TMWOA	
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Mean
5	6	348	305	310	386	305	**273**	NS	301	NS	NS	304	**273**	298	308.00
	7	458	477	452	678	466	433	NS	432	NS	NS	434	433	**430**	436.41
6	7	1089	921	1015	1201	**921**	982	NS	963	NS	NS	973	982	945	968.61
	8	1466	1515	1455	1763	1493	NS	NS	1399	NS	NS	1401	NS	**1389**	1398.58

Statistically, the Wilcoxon signed-rank test was also applied to the results reported in Tab. 4. Tab. 6 presents the outcomes of the Wilcoxon signed-rank test. It was confirmed that TSWOA produced significantly different results from the GBGA, GS, APSO, CS, and ABCVS strategies, except the QLSCA and DPSO strategies. However, one may argue that there were insufficient output samples for the QLSCA and DPSO strategies to provide a comprehensive picture. Both strategies' interaction strength was up to four (i.e., $t \leq 4$).

Tab. 5, which was for CA (t, v^7) configurations where t varied as $2 \leq t \leq 6$, v varied as $2 \leq v \leq 7$, and p remained constant at $p = 7$. The results showed that TSWOA outperformed the pure computational and AI-based strategies. Additionally, the Wilcoxon test was implemented on the results reported in Tab. 5, and the outcomes of the Wilcoxon test are illustrated in Tab. 7.

Table 5: Test suite size performance for CA (t, v^7) where v varied from 2 to 7 and t varied from 2 to 6

CA (t, v^7)		Pure computation strategies					AI-based strategies						
t	v	Jenny	TConfig	PICT	IPOG-D	IPOG	QLSCA	GS	DPSO	APSO	CS	TMWOA	
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Mean
2	2	8	7	7	8	7	7	**6**	7	**6**	**6**	7	7.25
	3	16	15	16	15	15	15	**14**	**14**	15	15	**14**	14.77
	4	28	28	27	32	29	**23**	24	24	25	25	24	25.25
	5	37	40	40	45	45	**34**	36	**34**	35	37	36	37.70
	6	55	57	56	72	55	48	52	**47**	NS	NS	51	53.16
	7	74	76	74	91	**49**	64	68	64	NS	NS	70	72.12
3	2	14	16	15	14	16	15	**12**	15	15	**12**	**12**	14.29
	3	51	55	51	50	55	49	49	49	**48**	49	**48**	50.06
	4	124	**112**	124	114	**112**	**112**	116	**112**	118	117	113	116.90
	5	236	239	241	252	237	**215**	221	216	239	223	221	224.80
	6	400	423	413	470	420	**364**	374	365	NS	NS	382	386.25
4	2	31	36	32	40	35	31	27	34	30	27	**25**	30.70
	3	169	166	168	226	167	149	153	**150**	153	155	151	156.48
	4	517	568	529	704	614	477	486	**472**	**472**	487	484	488.58

(Continued)

Table 5: Continued

CA (t, v^7)		Pure computation strategies					AI-based strategies						
t	v	Jenny	TConfig	PICT	IPOG-D	IPOG	QLSCA	GS	DPSO	APSO	CS	TMWOA	
		Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Mean
5	2	57	56	57	80	60	NS	**51**	NS	NS	53	53	55.41
	3	458	477	452	678	466	NS	432	NS	NS	439	**430**	436.41
	4	1938	**1792**	1933	2816	1792	NS	1821	NS	NS	1845	1813	1822.77
6	2	87	**64**	72	96	**64**	NS	65	NS	NS	66	**64**	72.00
	3	1087	**921**	1015	1201	921	NS	963	NS	NS	973	945	968.61
	4	6127	NS	5847	5120	**4096**	NS	5608	NS	NS	5610	5567	5591.12

Furthermore, the results of the Wilcoxon test presented in Tabs. 6 and 7 revealed that the results of TSWOA were not significantly different from those of QLSCA and DSPO. This occurred because there were not sufficient samples to compare and test as the interaction strength of both these strategies was only up to four (i.e., $t \leq 4$). Moreover, it has been demonstrated previously that the higher the interaction strength, the higher is the efficiency of the strategy (algorithm) in detecting faults [31]. Therefore, TSWOA will produce a test suite with an interaction strength of up to 20.

Table 6: Wilcoxon test for the results reported in Tab. 4

	Ranks			Test statistics	
	TMWOA >	TMWOA <	TMWOA =	Z	Asymp. Sig. (2-tailed)
ABCVS	1	15	6	−2.767	0.006
CS	0	15	8	−3.429	0.001
APSO	2	9	6	−2.157	0.031
DPSO	3	5	8	−0.574	0.566
GS	1	11	11	−2.886	0.004
QLSCA	5	4	10	−0.611	0.541
GBGA	2	16	4	−2.459	0.014
IPOG	2	17	4	−3.069	0.002
IPOG-D	1	19	3	−3.811	0.000
PICT	0	23	0	−4.208	0.000
TConfig	1	22	0	−3.563	0.000
Jenny	0	22	1	−4.116	0.000

Table 7: Wilcoxon test for the results reported in Tab. 5

	Ranks			Test statistics	
	TMWOA >	TMWOA <	TMWOA =	Z	Asymp. Sig. (2-tailed)
CS	1	14	2	−3.254	0.001
APSO	3	7	1	−1.746	0.081
DPSO	8	3	3	−1.514	0.130
GS	4	11	5	−1.720	0.085
QLSCA	9	4	1	−1.615	0.106
IPOG	4	13	2	−1.705	0.088
IPOG-D	1	19	0	−3.212	0.001
PICT	0	19	1	−3.826	0.000
TConfig	3	14	2	−2.274	0.023
Jenny	0	20	0	−3.924	0.000

Based on previously reviewed metaheuristic strategies, it was noticed that most of these strategies did not support higher interaction strength. Fig. 7 depicts the maximum support of the recently developed metaheuristic strategies for interaction strength.

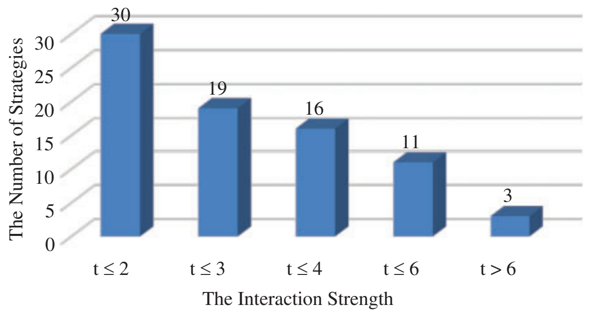

Figure 7: The Maximum support for interaction strength

Fig. 7 illustrates that most AI-based strategies produced a pairwise (i.e., t $\leq$ 2) test suite. Moreover, 30% of the strategies supported interaction strength of up to six. Only a few strategies went beyond that, such as the HSS [12], which supported up to 15, the GS [13], which supported up to 20, and the GSTG [19], which supported up to 10. As observed, and unlike most strategies, TSWOA supported high interaction strength.

Tabs. 8 and 9 display the ability of TSWOA to process higher interaction strength of up to 10 and 20, respectively. Tab. 8 presents the results for the configuration of CA (t, 2^{10}), where t varied from 2 to 10. When $t = 2$, two pure computational strategies IPOG and IPOG-D, along with GS, GSTG, and TSWOA, produced the best results. While, when $t = 3$, GS, GSTG, and TSWOA were observed to generate the best test suite size. When $t = 4$, only GS produced the best results. However, for the rest of the interaction strengths up to 10, TSWOA outperformed the other strategies and had better results. This implied that the proposed variant TSWOA generated better results when the search space was larger because the larger the interaction strength, the larger the search space.

Table 8: Test suite size performance for higher interaction strength CA (t, 2^{10}) where t varied from 2 to 10

CA (t, 2^{10})	Pure computation strategies					AI-based strategies					
t	Jenny	TConfig	PICT	IPOG-D	IPOG	GS		GSTG		TMWOA	
	Best	Best	Best	Best	Best	Best	Mean	Best	Mean	Best	Mean
2	10	9	9	**8**	**8**	**8**	8.60	**8**	NA	**8**	**8.48**
3	18	20	18	18	20	**16**	16.90	**16**	NA	**16**	**16.22**
4	39	45	43	51	45	**25**	**36.10**	27	NA	31	38.25
5	87	95	87	124	94	79	82.90	74	NA	**72**	**80.12**
6	169	183	173	231	181	157	159.90	156	NA	**153**	**157.90**
7	311	NS	309	NS	336	293	298.60	295	NA	**286**	**293.19**
8	521	NS	504	NS	503	495	504.40	502	NA	**490**	**501.16**
9	788	NS	728	NS	**512**	562	635.20	580	NA	**546**	**629.19**
10	**1024**	NS	**1024**	NS	**1024**	**1024**	**1024**	**1024**	NA	**1024**	**1024**

Table 9: Assessment of TSWOA against other strategies for higher interaction strength

CA	Pure computation strategies					AI-based strategies			
	Jenny	TConfig	PICT	IPOG-D	IPOG	GS	HSS	TMWOA	
	Best	Best	Best	Best	Best	Best	Best	Best	Mean
CA (2, 3^4)	13	10	13	15	9	9	9	**9**	9.8
CA (3, 3^5)	40	40	43	45	41	38	39	**37**	39.7
CA (4, 3^6)	140	141	142	162	141	129	132	**128**	133.2
CA (5, 3^7)	458	477	452	678	466	432	436	**430**	436.4
CA (6, 3^8)	1466	1515	1455	1493	1409	1339	1402	**1389**	1398.5
CA (7, 3^9)	4746	>day	4618	NS	NS	4437	4454	**4418**	4439.9
CA (8, 3^{10})	14999	>day	14599	NS	NS	13907	13990	**13872**	13914.3
CA (9, 3^{11})	47009	>day	45521	NS	NS	43809	44120	**43793**	43827.6
CA (10, 3^{12})	147004	>day	141990	NS	NS	136096	137006	**136029**	136123.6
CA (11, 3^{12})	305797	>day	278993	NS	NS	267631	>day	**267391**	267672.2
CA (12, 2^{14})	9422	>day	9112	NS	NS	8893	8905	**8889**	8895.1
CA (13, 2^{14})	13251	>day	12441	NS	NS	10251	**10250**	**10250**	10267.1
CA (14, 2^{15})	26579	>day	25036	NS	NS	**23377**	24121	23809	23822
CA (15, 2^{16})	53977	>day	51127	NS	NS	**46575**	46893	**46575**	46599.9
CA (16, 2^{17})	>day	>day	100266	NS	NS	**95680**	NS	**95680**	95702.1
CA (17, 2^{18})	>day	>day	>day	NS	NS	179546	NS	**179506**	179568.9

(Continued)

Table 9: Continued

CA	Pure computation strategies					AI-based strategies			
	Jenny	TConfig	PICT	IPOG-D	IPOG	GS	HSS	TMWOA	
	Best	Best	Best	Best	Best	Best	Best	Best	Mean
CA (18, 2^{19})	>day	>day	>day	NS	NS	330391	NS	**330367**	330433.9
CA (19, 2^{20})	>day	>day	>day	NS	NS	624940	NS	**624919**	624991.5
CA (20, 2^{20})	>day	>day	>day	NS	NS	1048576	NS	0000	1111

Tab. 9 presents the results for various configurations of CAs with interaction strength of up to 20. The results presented in Tab. 9 were obtained from a previously published report [13]. IPOG and IPOG-D supported interaction strength of up to 6 and did not produce better results. While PICT generated a test suite with a higher interaction strength up to 16, it took more than a day to produce the test suite for the interaction strengths greater than that. Similar observations were noted for TConfig and Jenny, where TConfig generated a test suite with interaction strengths of 6 and Jenny generated up to 16. GS and HSS developed test suites with interaction strengths up to 20 and 15, respectively. TSWOA outperformed both pure computational and AI-based strategies and generated a smaller test suite size with an interaction strength of up to 20.

8 Threats to validity

It is worth mentioning that we faced threats to the validity and attempted to minimize them. During our research, we encountered a couple of threats. The first threat is that the impartiality of benchmark experiments based on meta-heuristics may be a concern. Many comparisons with similar work rely solely on published results due to a lack of source codes. The second threat is the statistical analysis focused on the best-reported values rather than the mean or average values. The main issue is that some of the best results may be obtained by chance, affecting the conclusion.

9 Conclusion

This study proposed three variants of WOA, which supported a higher interaction strength of up to 20 (i.e., $2 \leq t \leq 20$), unlike most AI-based strategies. The proposed variants of WOA were developed through structure modification (SWOA), incorporation of the acceptance probability of the EMC-FC algorithm (TWOA), and a combination of both SWOA and TWOA (TSWOA). The results of the experiments indicated that the proposed variants of WOA effectively managed to overcome the issue of premature convergence and escaped the local optima. Moreover, TSWOA performed better than the other variants by statistically generating smaller test suite sizes. Furthermore, the TSWOA variant performed better than the pure computational-based strategies by statistically generating smaller test suite sizes. Further, it outperformed most of the existing AI-based strategies by conducting various experiments on different configurations. Additionally, the experimental results illustrated that TSWOA could support higher interaction strengths up to 20 while competing against other state-of-the-art strategies in terms of efficiency and performance. The higher the interaction strength, the more time it takes to produce test suites. Higher interaction strength would exponentially increase the search space (i.e., the t-tuple table). Within the context of the future work to be conducted by our research group, this report extends the possibility of TSWOA to support variable strength t-way testing with constraints.

Acknowledgement: The authors are grateful to the Ministry of Education, Malaysia, and the Universiti Kebangsaan Malaysia for their support. Ali Abdullah Hassan would like to express his gratitude to Hadhramout Foundation in Yemen for their tuition fee support.

Funding Statement: This work was supported by the Ministry of Education, Malaysia (FRGS /1/2019/ICT02/UKM/01/1) and the Universiti Kebangsaan Malaysia (DIP-2016-024).

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

References

[1] P. M. Pontes, B. Lima and J. P. Faria, "Izinto: A pattern-based IoT testing framework," in *Companion Proc. ISSTA/ECOOP, 2018 Workshops*, Amsterdam, Netherlands, pp. 125–131, 2018.

[2] A. H. Ronneseth and C. J. Colbourn, "Merging covering arrays and compressing multiple sequence alignments," *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2177–2190, 2009.

[3] D. R. Kuhn, D. R. Wallace and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[4] B. Jenkins, "Jenny test tool," *Online Tool*, 2016.

[5] A. W. Williams, Determination of test configurations for pair-wise interaction coverage. In: *Testing of Communicating Systems*. Vol. 48. Boston, MA: Springer, pp. 57–59, 2000.

[6] J. Czerwonka, "Pairwise testing in real-world: Practical extensions to test case generators," in *Proc. of the 24th Pacific Northwest Software Quality Conf.*, Portland, Oregon, USA, vol. 45, pp. 419–430, 2006.

[7] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "A general strategy for t-way software testing," in *14th Annual IEEE Int. Conf. and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, Tucson, AZ, USA, pp. 549–556, 2007.

[8] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "Efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.

[9] J. Stardom, "Metaheuristics and the search for covering and packing arrays [microform]," Ph.D. dissertation, Simon Fraser University, British Columbia, Canada, 2001.

[10] B. S. Ahmed, K. Z. Zamli and C. P. Lim, "Application of particle swarm optimization to uniform and variable strength covering array construction," *Applied Soft Computing*, vol. 12, no. 4, pp. 1330–1347, 2012.

[11] T. Shiba, T. Tsuchiya and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proc. of the 28th Annual Int. Computer Software and Applications Conf., COMPSAC*, Hong Kong, China, pp. 72–77, 2004.

[12] 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," *Information and Software Technology*, vol. 54, no. 6, pp. 553–568, 2012.

[13] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Information and Software Technology*, vol. 94, pp. 165–185, 2018.

[14] M. Ayob and G. Kendall, "A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine," in *Proc. of the Int. Conf. on Intelligent Technologies, InTech*, Chiang Mai, Thailand, vol. 3, pp. 132–141, 2003.

[15] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 59, no. 1, 2, pp. 125–172, 2004.

[16] B. S. Ahmed and K. Z. Zamli, "PSTG: a t-way strategy adopting particle swarm optimization," in *Fourth Asia Int. Conf. on Mathematical/Analytical Modelling and Computer Simulation*, Kota Kinabalu, Malaysia, pp. 1–5, 2010.

[17] H. Wu, C. Nie, F. C. Kuo, H. Leung and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 575–591, 2014.

[18] 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," *Information and Software Technology*, vol. 66, pp. 13–29, 2015.

[19] 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*, In Press, 2021.

[20] T. Mahmoud and B. S. Ahmed, "An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8753–8765, 2015.

[21] K. Z. Zamli, B. S. Ahmed, T. Mahmoud and W. Afzal, "Fuzzy adaptive tuning of a particle swarm optimization algorithm for variable-strength combinatorial test suite generation," 2018. [Online]. Available: https://arxiv.org/abs/1810.05824.

[22] A. K. Alazzawi, H. M. Rais, S. Basri, Y. A. Alsariera, L. F. Capretz *et al.,* "HABCSm: A hamming based t-way strategy based on hybrid artificial bee colony for variable strength test sets generation," 2021. [Online]. Available: https://arxiv.org/abs/2110.03728.

[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. e0195675, 2018.

[24] J. Torres-Jimenez and J. C. Perez-Torres, "A greedy algorithm to construct covering arrays using a graph representation," *Information Sciences*, vol. 477, no. 2, pp. 234–245, 2019.

[25] 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.

[26] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[27] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, no. 12, pp. 51–67, 2016.

[28] 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.

[29] G. Kaur and S. Arora, "Chaotic whale optimization algorithm," *Journal of Computational Design and Engineering*, vol. 5, no. 3, pp. 275–284, 2018.

[30] M. Abdel-Basset, G. Manogaran, D. El-Shahat and S. Mirjalili, "A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem," *Future Generation Computer Systems*, vol. 85, no. 1, pp. 129–145, 2018.

[31] S. Abdullah, N. R. Sabar, M. Z. A. Nazri and M. Ayob, "An exponential monte-carlo algorithm for feature selection problems," *Computers & Industrial Engineering*, vol. 67, no. 10, pp. 160–167, 2014.