IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 26, 2020, accepted October 18, 2020, date of publication October 21, 2020, date of current version November 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3032851

Combinatorial Test Suites Generation Strategy Utilizing the Whale Optimization Algorithm

ALI ABDULLAH HASSAN[1], SALWANI ABDULLAH[1], KAMAL Z. ZAMLI[2], (Member, IEEE), AND ROZILAWATI RAZALI[3]

[1]Center for Artificial Intelligence Technology, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia
[2]Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang, Pekan 26600, Malaysia
[3]Center for Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia

Corresponding author: Ali Abdullah Hassan (ali87hassan@gmail.com)

This work was supported in part by the Ministry of Education, Malaysia, under Grant FRGS /1/2019/ICT02/UKM/01/1, and in part by the Universiti Kebangsaan Malaysia under Grant DIP-2016-024.

ABSTRACT The potentially many software system input combinations make exhaustive testing practically impossible. To address this issue, combinatorial t-way testing (where t indicates the interaction strength, i.e. the number of interacting parameters (input)) was adopted to minimize the number of cases for testing. Complimentary to existing testing techniques (e.g. boundary value, equivalence partitioning, cause and effect graphing), combinatorial testing helps to detect faults caused by the faulty interaction between input parameters. In the last 15 years, applications of meta-heuristics as the backbone of t-way test suite generation have shown promising results (e.g. Particle Swarm Optimization, Cuckoo Search, Flower Pollination Algorithm, and Hyper-Heuristics (HHH), to name a few). Supporting the No Free Lunch theorem, as well as potentially offering new insights into the whole process of t-way generation, this article proposes a new strategy with constraint support based on the Whale Optimization Algorithm (WOA). Our work is the first attempt to adopt the WOA as part of a search-based software engineering (SBSE) initiative for t-way test suite generation with constraint support. The experimental results of the test-suite generation indicate that WOA produces competitive outcomes compared to some selected single-based and population-based meta-heuristic algorithms.

INDEX TERMS Search-based software engineering (SBSE), T-way testing, combinatorial testing, software testing, meta-heuristic.

I. INTRODUCTION

Ensuring conformance to specification, software testing is often considered a determinant of quality. In many situations, testers often race against time to release software on-time and on schedule. Practically, however, it is impossible to consider all exhaustive test cases because of the numerous time and resource constraints involved.

Combinatorial testing provides a convenient mechanism to minimize the number of test cases by considering a subset of interactions between parameters, called t-way testing. The fundamental idea of t-way testing is that ''a fault is usually caused by interactions of two or more system inputs (say, t number of parameters)'' [1], [2]. Many t-way testing applications have demonstrated encouraging results (e.g. at t = 6, almost 90 percent of faults can be triggered and detected). Nevertheless, it should be noted that combinatorial testing does not replace existing minimization strategies (such as boundary value, equivalence partitioning, cause-effect-graphing and the like) but rather complements them.

To-date, in line with the emergence of a new field called Search-based Software Engineering (SBSE), which deals with solving optimization problems within the Software Engineering lifecycle, many related works have adopted meta-heuristics to address the combinatorial t-way test suite generation. Such applications include PSO [3], Cuckoo Search (CS) [4], the Flower Pollination Algorithm (FPA) [5], Ant Colony System (ACS) [6], and High Level Hyper-Heuristics (HHH) [7].

The No Free Lunch theorem suggests that no single meta-heuristic is superior to the other in all optimization cases. In line with this idea, the adoption of a new meta-heuristic is most welcome. This article proposes a new strategy with constraint support for t-way test suite generation based on

The associate editor coordinating the review of this manuscript and approving it for publication was Seyedali Mirjalili.

the Whale Optimization Algorithm (WOA). The Whale Optimization Algorithm (WOA) is a recently developed algorithm based on the hunting behavior of the humpback whale [8]. WOA has a strong global search capacity due to its distinctive optimization mechanism [9]. In addition, WOA is less parameter-dependent and has a straightforward implementation [9]. It has therefore been commonly proposed in various domains to solve many issues, such as feature selection [10], clustering [11], flow shop scheduling [12], electronic engineering [13], energy [14], and electrical power [15], to name a few. Moreover, the WOA has also shown competitive outcomes in all domains. Owing to its robust performance against many existing meta-heuristics, the adoption of WOA for the currently proposed combinatorial t-way test suite generation appears justifiable.

Complementing existing works on t-way testing meta-heuristics, our contributions are two-fold. Firstly, we present the first work of its kind that adopts WOA for t-way test suite generation. More precisely, our work investigates the hypothesis that the adoption of WOA is useful for SBSE applications involving constrained and unconstrained software test suite generation. Secondly, we extensively evaluate the performance of WOA through a set of benchmark test suites.

We organized our paper as follows: Section II presents the background on the t-way strategy using definitions and scenario examples, while the related works are presented in Section III. In Section IV and Section V, we introduce the Whale Optimization Algorithm (WOA) and its implementation in t-way testing, respectively. Preliminary findings and discussion are presented in Section VI and Section VII concludes this research.

II. OVERVIEW OF T-WAY TESTING

To demonstrate t-way testing, let us consider the following hypothetical smart city planning example, as shown in Figure 1.

Smart city planning consists of five basic components/parameters, i.e. a transport system, e-service, smart traffic management, health cards, and water level monitoring. The transport system parameter takes three possible values (i.e., Transport System = Public Transport, e-hailing, Individual Vehicle), whereas the rest of the parameters take two possible values (i.e., e-Service = Wired, Wireless, Smart Traffic Management = Sensors, CCTV, Health Cards = Government Hospital, Private Hospital, and Water Level Monitoring = Tripping-bucket Rain Gauge, Hydrophone).

Figure 2 shows that the covering array as MCA (N;3, $3^1 2^4$) in the smart city planning example, assuming the interaction strength is $t = 3$. The exhaustive test for smart city planning requires $3 * 2 * 2 * 2 * 2 = 48$ test cases to cover all the smart city planning configurations. Meanwhile, when the meta-heuristic strategy (i.e., WOA) is used in 3-way testing, only 17 test cases are generated to cover all the configurations of the above-mentioned example.

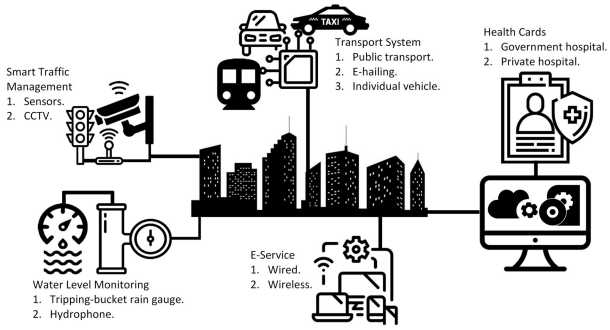


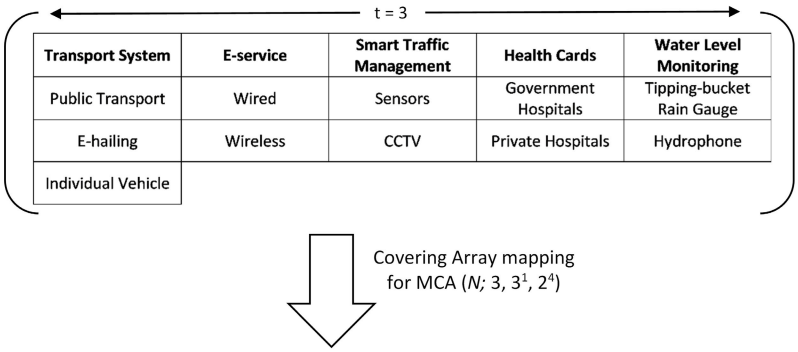
FIGURE 1. Smart city planning.

Mathematically, the test suite is a process of constructing an array, $N * k$ where N is the number of test cases and k is the number of parameter values. Every test case consists of a combination of k parameter values [1]. It is mandatory to include all combinations of the t-way parameter values in the test suite. The interaction strength is the number of interacting parameters, denoted as t. Some important definitions of the terminologies used are listed below:

- *T-way testing* is a combinatorial software testing method that examines the t-way interaction of every possible discrete combination of input parameters. This testing can be done much faster than an exhaustive search of all combinations of all parameters.
- An *interaction* represents a combination of two or more different parameters with a specific value.
- The *Covering Array* (CA) represents the test suite, which is an array of size $N * v$, where v is the value (option of system configuration/input user), p is the parameter (system configuration/ user input), t is the interaction strength, and N is the number of test cases generated and is denoted as CA $(N; t, v^p)$. Minimizing the size of the test suite, as well as retaining fault detection capabilities, are critical to escape time and resource constraints and to maintain the effective detection of faults [16].
- *Mixed Covering Array* (MCA) is denoted as MCA $(N; t, v_1^{p1}, v_2^{p2}, v_i^{pi})$ similar to CA except the number of parameter values varies.
- *t-tuple* is an array (table) containing a selection of t parameters.

Current t-way strategies generate test cases (i.e. the test suite) to cover every single combination generated as a result of parameter-interaction. However, some test suite combinations should be omitted when producing the final test suite as a result of unacceptable outputs or undesirable test suite combinations. Such combination types are identified as forbidden combinations or constraint combinations. Forbidden combinations or constraints are types of test cases that may result in defective performance or output. Thus, they should not be generated in the final test suite.

CA with constraints can be denoted as CCA $(N; t, v^p, F)$, where F is the forbidden combination. F can also be



Mixed Covering Array of MCA (N; 3, 3^1, 2^4)

No of Test cases	Transport System	E-service	Smart Traffic Management	Health Cards	Water Level Monitoring
1	E-hailing	Wired	Sensors	Government Hospitals	Tipping-bucket Rain Gauge
2	Individual Vehicle	Wired	CCTV	Government Hospitals	Hydrophone
3	Individual Vehicle	Wireless	CCTV	Private Hospitals	Tipping-bucket Rain Gauge
4	E-hailing	Wireless	Sensors	Private Hospitals	Hydrophone
5	Public Transport	Wired	Sensors	Private Hospitals	Tipping-bucket Rain Gauge
6	Public Transport	Wireless	Sensors	Government Hospitals	Hydrophone
7	E-hailing	Wired	CCTV	Private Hospitals	Hydrophone
8	Public Transport	Wireless	CCTV	Government Hospitals	Tipping-bucket Rain Gauge
9	Individual Vehicle	Wireless	Sensors	Government Hospitals	Tipping-bucket Rain Gauge
10	Individual Vehicle	Wired	Sensors	Private Hospitals	Hydrophone
11	Public Transport	Wireless	CCTV	Private Hospitals	Hydrophone
12	E-hailing	Wireless	CCTV	Government Hospitals	Tipping-bucket Rain Gauge
13	Public Transport	Wired	CCTV	Government Hospitals	Hydrophone
14	Individual Vehicle	Wired	CCTV	Government Hospitals	Tipping-bucket Rain Gauge
15	E-hailing	Wireless	Sensors	Government Hospitals	Hydrophone
16	Individual Vehicle	Wireless	Sensors	Government Hospitals	Hydrophone
17	E-hailing	Wired	Sensors	Private Hospitals	Tipping-bucket Rain Gauge

FIGURE 2. 3-way mixed covering array construction for smart city planning.

denoted as $F = \{(C_{p_{no},v_{no}}, C_{p_{no},v_{no}})_1, (C_{p_{no},v_{no}}, C_{p_{no},v_{no}})_2, \ldots, (C_{p_{no},v_{no}}, C_{p_{no},v_{no}})_n\}$, where C is the constraint, p_{no} represents the parameter number in the t-tuple table, and v_{no} represents the value number of the parameter in the t-tuple table. Section V further elaborates on these constraints.

III. RELATED WORK

Combinatorial interaction testing strategies use Greedy test suite construction algorithms [17]. Every iteration of the design process aims to cover the maximum number of combinations. The test suite may be constructed by using either one parameter at a time (OPAT) or one test at a time (OTAT) [18]. The OPAT approaches start the test suit composition for the first two parameters or smallest t-combination. Next, it expands the test suite horizontally by inserting one parameter per iteration until the t-way requirements have been completed. IPOG [19] and IPOG-D [20] are examples of such an approach.

Unlike OPAT, OTAT approaches start by producing one test case per iteration, including all the parameters, to cover the maximum number of combinations. The iteration lasts until all the t-combinations are covered. Because of its good performance, many studies have applied OTAT methods, such as Jenny [21] and TConfig [22]. A number of OTAT-based approaches have recently implemented meta-heuristic algorithms to produce a t-way test suite.

Meta-heuristic optimization algorithms give adequate solutions within a sensible time for solving hard and complex issues in science and engineering; thus, justifying the increased interest among researchers and scientists in this area. Meta-heuristic optimization algorithms solve optimization problems by imitating evolution behavior, swarm behavior, or the law of physics [23].

Evolutionary algorithms emulate the processes of evolution in nature. These algorithms use biological evolution-inspired mechanisms, such as reproduction, mutation, recombination, and selection. Examples of such algorithms are the Genetic Algorithm (GA) [24], Differential Evolution (DE) [25], Biogeography-Based Optimizer (BBO) [26], and Asexual Reproduction Optimization (ARO) algorithm [27], to name a few. Meanwhile, swarm-based algorithms emulate the social behavior of swarms, birds, insects, and animal groups. Such animal behaviors include searching for food, locating other individuals, and flocking, with example algorithms being Particle Swarm Optimization (PSO) [28], the Artificial Bee Colony (ABC) algorithm [29],

TABLE 1. Summary of strategies applied on t-way testing.

Year	AI-based Techniques / Algorithms
2015	· TCA [41] · Cuckoo Search (CS) [4] · Flower Strategy (FS) [5] · DPSO [42] · SITG [43]
2016	· Ant Colony System (ACS) [6] · Bat-inspired testing strategy (BTS) [44]
2017	· PABC [45] · ABCS [46] · pTLBO [47] · ATLBO [18]
2018	· FATG [48] · PKS [49]
2019	· Improved Jaya Algorithm (IJA) [50]
2020	· Sine Cosine Algorithm (SCA) [17] · Multiple Black Hole (MBH) algorithm [39]

the Bat Algorithm (BA) [30], the Cuckoo Search (CS) algorithm [31], the Grey Wolf Optimizer (GWO) algorithm [32], the Ant Lion Optimizer (ALO) algorithm [33], and the Moth-Flame Optimization (MFO) algorithm [23] to name a few. Lastly, physics-based algorithms emulate physical or chemical processes, such as gravity, ion motion, electrical charges, river systems, etc. Algorithms in this category include Simulated Annealing (SA) [34], the Gravitational Search Algorithm (GSA) [35], the Artificial Chemical Reaction Optimization Algorithm (ACROA) [36], Heat Transfer Search (HTS) [37], and Henry Gas Solubility Optimization (HGSO) [38], to name a few.

To date, there is a fairly comprehensive literature on combinatorial testing, spanning various approaches. Nevertheless, each of these approaches share a common aspect: when combined with heuristics, these approaches can harness the power of random combinatorial searching to evaluate t-strength covering arrays. Two main aspects must be focused on to formulate the optimization problem: the definition of the objective function; and the selection of the technique, whether utilizing a pure-based approach or a hybrid-based approach [39]. In terms of the objective function, the number of tuples covered for the candidate test case (i.e. weight) is used as the fitness value. Meanwhile, the number of uncovered tuples is taken as the cost of the candidate test case in another case, which needs minimization.

As for the type of meta-heuristic algorithm applied in t-way combinatorial testing, Table1 summarizes some of the algorithms introduced in the last five years, as further explained in the following sections. Some of these algorithms have been listed in [40] and their variants have also been updated and provided here as well.

PSO was firstly applied to t-way testing in 2010 to generate a test suite. PSO imitates the behavior of flocks of birds searching for food. The optimal solution (position) is calculated using individual position and velocity. In each flock, an individual moves towards the best individual position and the best global position (optimal solution) [3]. PSO has opened many developmental ideas on variant algorithms due to its rapid convergence rate behavior and less demanding computational requirements. These variants include DPSO [42], SITG [43], PSTG [51], etc.

TCA [41] integrates Greedy Tabu search and heuristic random walk. Initialization of test cases is generated using Greedy Tabu search. TCA performs the heuristic search method to extend the search to discover any uncovered interactions. Another algorithm emerged in 2015 called the Cuckoo Search (CS) [4], which was implemented in t-way combinatorial testing with a small number of control parameters. A variant of CS was also implemented in t-way testing that upgrades the search space with Levi flights [31]. Then, Flower Strategy (FS) [5] was introduced in 2015 derived from the efficiency of the Flower Pollination Algorithm (FPA). Some defining features of FPA are its simplicity, flexibility, and low complexity.

Ant Colony System (ACS) [6] is an AI-based strategy and is a type of Ant Colony Optimization (ACO). ACS has effectively resolved numerous combinatorial optimization issues. Its strategy provides all kinds of interactions, particularly IOR. Additionally, another strategy based on the Bat Algorithm (BA) was introduced, called the Bat-inspired Testing Strategy (BTS), where the BA works as the main search engine to obtain the optimal test suite size [44].

Meanwhile, in 2017, several strategies were implemented in t-way testing, such as the Artificial Bee Colony Algorithm (ABC) and the Teaching-Learning-Based Optimization algorithm (TLBO). ABC was designed to imitate a honey bee colony's feeding behavior. Several variants of ABC have also been implemented in t-way testing. For instance, the Pairwise Artificial Bee Colony algorithm (PABC) [45] was implemented in 2-way testing and the Artificial Bee Colony Strategy (ABCS) [46] was applied for a higher interaction strength of up to ten (i.e. $t \leq 10$). Meanwhile, TLBO mimics the classroom environment, which has two stages, i.e. a teacher (global search) and a learner (local search). TLBO was applied in pairwise testing (i.e. 2-way testing) to generate a test suite [47]. Meanwhile, another variant called Adaptive TLBO (ATLBO) was also implemented in t-way testing in another study [18]. ATLBO uses the Mamdani Fuzzy inference system to enhance the selection process between the global search and the local search [18].

Two novel algorithms were introduced in 2018 for application in t-way testing, namely the firefly algorithm (FA) and the kidney algorithm (KA). FA was inspired by the distinguishing feature of the firefly, namely the flash patterns that attract consorts and scare away predators. A strategy called FATG based on FA was introduced to minimize the test suite and reduce execution time [48]. KA emulates the role of the kidneys in the human body. KA involves two main procedures: filtration (local search) and reabsorption (global search). The Pairwise Kidney Strategy (PKS) was developed based on KA to generate a smaller test suite [49].

Improved Jaya Algorithm (IJA) [50] is a population-based algorithm developed to address constrained and unconstrained problems. The key idea behind the algorithm is that every candidate solution will seek the best solution while simultaneously evading the worse solution. IJA is implemented in t-way testing by only updating the best test case and the worse test case. Then, the current test case is updated based on the best and worst test cases. To improve diversity and a quality solution, lévy flight was introduced, as well as a mutation operation, to improve the convergence speed of the proposed method in generating a test suite [50].

Multiple Black Hole (MBH) algorithm [39] emerged in 2020 for application in combinatorial testing. The Black Hole algorithm is a modern meta-heuristic method focused on observable evidence of the black hole phenomenon and the behavior of stars when interacting with the black hole. The Black hole algorithm is considered a population-based algorithm. The stars are the solutions (test cases) and the best star (test case) is selected as the black hole, which all solutions move towards based on their current location and a random number. MBH is based on the multi-swarm principle, which can be defined as multiple black holes. Additionally, MBH introduced the black hole energy to promote the removal of certain black hole swarms and to produce fresh ones [39]. Another algorithm introduced in 2020 is SCA [17], which is a population-based algorithm that produces numerous initial random test cases and allows the cases to fluctuate outwards or towards the best possible test case using a sine and cosine mathematical model. SCA was enhanced by introducing a combination of linear and exponential magnitude updates for search displacement [17].

IV. WHALE OPTIMIZATION ALGORITHM

The SBSE field has seen the extension of several metaheuristic algorithms, such as Greedy Search, Simulated Annealing, Genetic Algorithms, Tabu Search, and even the Whale Optimization Algorithm (WOA). However, WOA was applied in regression testing via hybridization with the Artificial Neural Network (ANN) [52]. Harikarthik *et al.* [52] introduced an innovative effort to investigate the effectiveness of WOA in regression testing by hybridizing it with ANN to optimize its weights. As for the t-way test suite generation problem, no study has yet used (WOA) to address software engineering issues, i.e., the optimization problems mentioned earlier. Therefore, it appears that the SBSE research community has not fully explored the potential of WOA.

In 2016, Mirjalili and Lewis [8] introduced WOA, which is a modern nature-inspired AI-based algorithm. WOA imitates the hunting behavior of humpback whales. Humpback whales are intelligent and have a sophisticated way of performing collective work. These creatures use a special tracking technique known as the bubble-net feeding technique, as shown in Figure 3. The whales perform this technique by making peculiar bubbles along a circle or a '9'-shaped path. Then, they hunt near the surface and trap the victim in a net of bubbles.

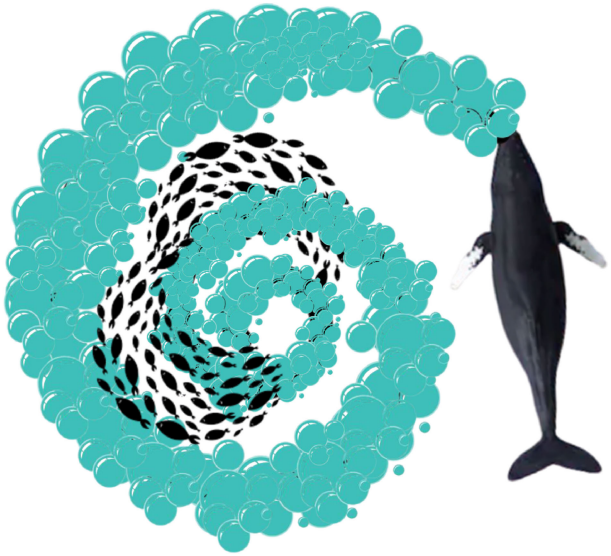


FIGURE 3. Bubble-net hunting behavior of humpback whales.

There are two stages of WOA: exploitation and exploration. The prey-encircling method and spiral bubble-net attacking technique are used in the exploitation stage, where both techniques update the position of the current search agent using the location of the best search agent. However, the spiral bubble-net attacking technique includes a randomness factor (i.e. explorational side), as seen in Equation (6). Meanwhile, in the exploration stage, a random search is conducted, where the position of the current search agent is updated based on a generated random search agent, as illustrated in Algorithm 1. The mathematical model for WOA is specified below:

A. EXPLOITATION PHASE

The two mechanisms used in this phase are as follows:

1) Encircling Prey:
Humpback whales can identify the victim's position and then surround the victim. In WOA, the target victim is presumed to be the current best candidate solution. Next, the best search agent is located, while all other search agents attempt to move towards it. In other words, the agent updates the movement (location) of the whale around the victim per the following mathematical model:

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

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

where t represents the current iteration, X^* represents the best solution obtained so far, and X is the current solution. Next, A and C are coefficients computed using Equations (3) and (4) respectively:

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

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

where a is reduced linearly from 2 to 0 during iterations as shown in Equation (5) and r is a random number in [0,1].

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

2) Bubble-net attacking technique:
This method involves two mechanisms: i) a shrinking encircling mechanism carried out by the reduction of the value of a in Equation (3), so the new location of a search agent is located between the genuine location of the agent and the location of the existing best agent; and ii) a spiral updating position mechanism used to calculate the distance between the current solution (whale) and the best solution (victim) using the spiral equation of Eq. (6):

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

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 number in $[-1, 1]$.
Humpback whales use both mechanisms simultaneously. To model this behavior, a 50% chance is introduced to select one of the mechanisms to update the location of the whales during the search. The mathematical model is outlined by Equation (7):

$$X(t+1) = \begin{cases} \text{Equation (2)}, & \text{if } p < 0.5 \\ \text{Equation (6)}, & \text{if } p \geq 0.5 \end{cases} \tag{7}$$

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

B. EXPLORATION PHASE

WOA is considered a global search. Therefore, the whales search randomly based on each other's location. Thus, the location of a search agent is randomly updated instead of depending on the best search agent found so far. This technique is used when the random values of A are greater than 1, to ensure the search agent moves away from a reference whale (best solution). This mechanism emphasizes global search and induces WOA to perform exploration. The mathematical model for this step is outlined by Equations (8) and (9):

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

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

V. IMPLEMENTATION OF WOA

The WOA-based approach is used to automatically generate a test-suite and to decrease the number of test cases. Figure 4 presents an overview of the WOA implementation in t-way testing, which consists of two phases:

A. T-TUPLE TABLE GENERATION

The outcome of this phase is the t-tuple table, which, as mentioned earlier, a sequence (or ordered list) of t elements.

Algorithm 1 Pseudo-Code of the WOA Algorithm

```
Initialize the whales population X_i (i = 1, 2, ..., n)
Calculate the fitness of each search agent
X* = the best search agent
while i < maximum number of iteration do
    for each search agent do
        Update a, A, C, l, and p
        if p < 0.5 then
            if |A| < 1 then
                Update the position of the current search agent using Eq (2)
            else if |A| > 1 then
                Select a random search agent ()
                Update the position of the current search agent using Eq (9)
            end if
        else if p ≥ 0.5 then
            Update the position of the current search using Eq (6)
        end if
    end for
    Check if any search agent goes beyond the search space and amend it
    Calculate the fitness of each search agent
    Update X* if there is a better solution
    t = t + 1
end while
return X*
```

To generate the t-tuple table, four steps are taken, as illustrated in Figure 4, and explained as follows: The first step is to obtain the system configuration or user input for the software to be tested. The second step is to decide on the interaction strength (t) of the t-way testing. The next step is to generate the parameter combination. For example, if we have 4 parameters (say a, b, c, and d) and the interaction strength, $t = 2$, then the 2-way combinations are (ab, ac, ad, bc, bd, and cd). The last step is to generate the t-tuple table that depends on parameter combination (generated in the previous step) and the values of the parameters. If we took the previous example of 4 parameters (a, b, c, d), each parameter has 2 values (0,1), so the t- tuple table will be represented by Table 2, where the x would be replaced randomly with one of the parameter values (0,1) during the search.

While in the presence of constraints, the forbidden combinations are obtained together with the system configuration or user input for the software to be tested.

B. TEST SUITE GENERATION

The t-tuple table generated in the previous phase is now an input for this stage, while the WOA attempts to cover its cells (interaction elements) with the minimum test cases. As illustrated in Figure 4, WOA will run until the t-tuple table becomes empty after applying the four steps shown

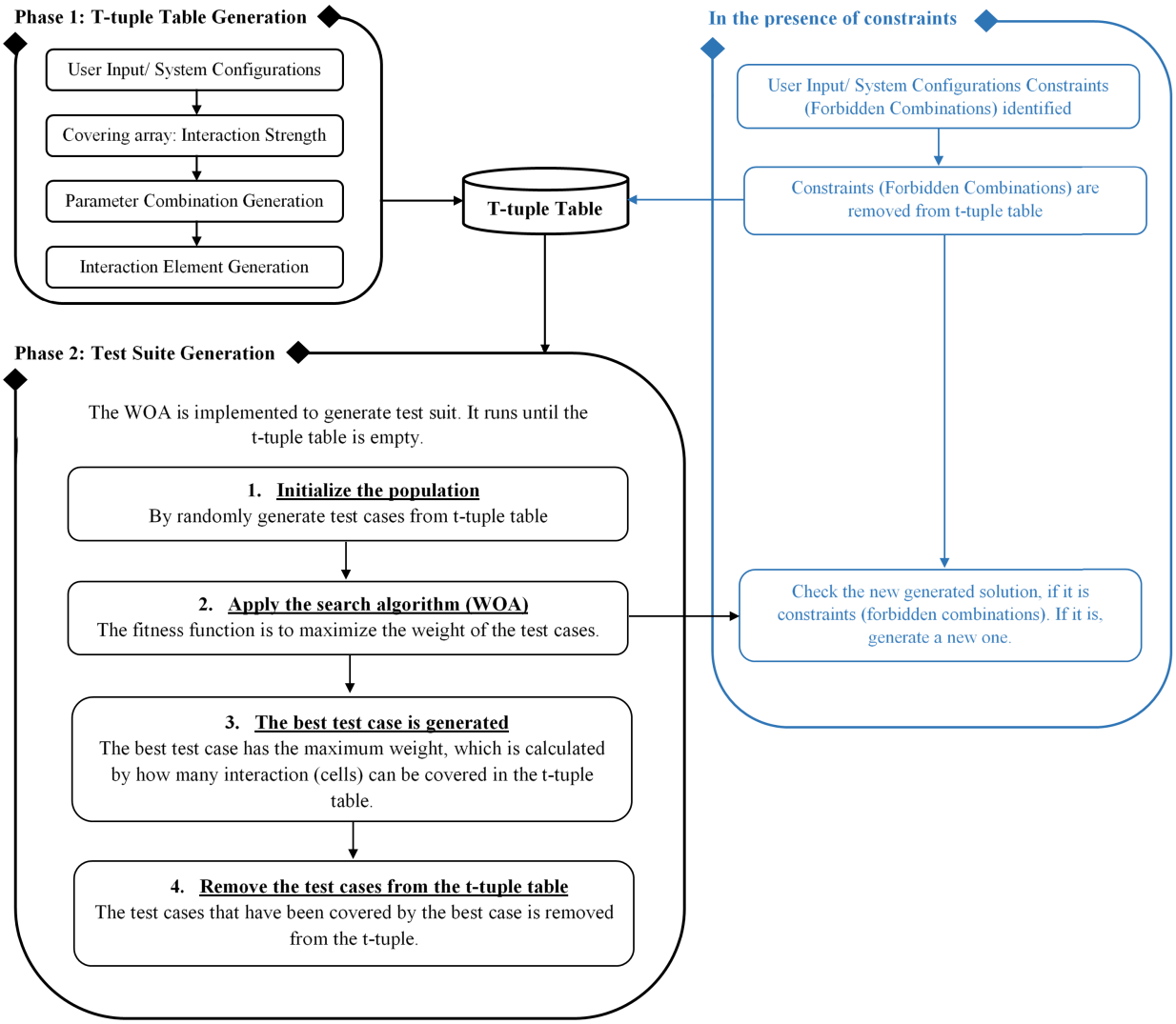


FIGURE 4. The overview of t-way testing implementation.

TABLE 2. T-tuple table.

ab	ac	ad	bc	bd	cd
00xx	0x0x	0xx0	x00x	x0x0	xx00
01xx	0x1x	0xx1	x01x	x0x1	xx01
10xx	1x0x	1xx0	x10x	x1x0	xx10
11xx	1x1x	1xx1	x11x	x1x1	xx11

in Figure 4. Figure 5 shows the elimination process in the t-tuple. As shown in Figure 5, WOA will search for the best test case based on weight. The weight is the number of six. This means that it covers six interactions in the t-tuple table, which are 1xx0, x1 × 0, xx10, 11xx, 1 × 1x, and x11x. Then, the covered interactions are removed from the t-tuple table and the best test case is added to the test suite array. This process continues until the t-tuple becomes empty, in other words, when all the cells (i.e. interactions) in the t-tuple table are covered.

Meanwhile, in the presence of constraints, each time the WOA updates its solution (i.e. generate new solution), the new solution will be checked whether or not it is one of the forbidden combinations. This step is to ensure that the solutions will not converge to one of the forbidden combinations.

Consider the example in Figure 6 of CCA $(N; 2, 2^4, F)$, where $F = \{(C_{p_1,v_2}, C_{p_2,v_1}), (C_{p_3,v_1}, C_{p_4,v_1})\}$. This means that the constraint covering array (CCA) consists of 4 parameters, with each having 2 values and an interaction strength of 2. Meanwhile, the forbidden combinations, F, has two constraints, and each constraint has a pair of tuples. The first constraint is $(C_{p_1,v_2}, C_{p_2,v_1})$, where the first tuple is C_{p_1,v_2} indicating parameter one and value two and the second tuple is C_{p_2,v_1} i.e., parameter two and value one. Thus, the first forbidden combination is (10xx), as per Figure 6. Similarly, the second forbidden combination will be (xx00), where x is a 'don't-care' value.

As for WOA, the generation process begins with a set of random solutions (initial population). Then, the solutions are

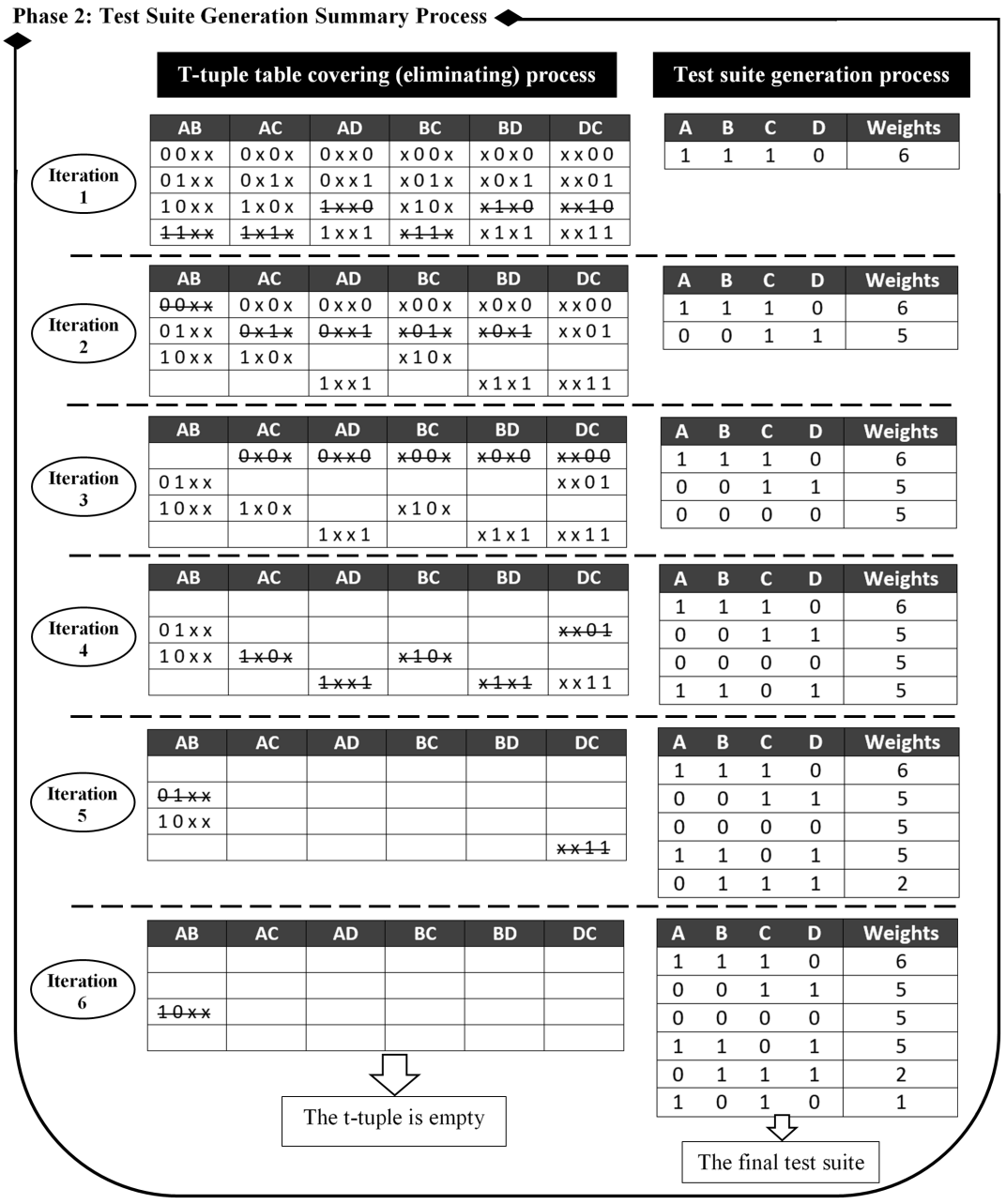


FIGURE 5. The process of t-tuple elimination and test suite generation.

evaluated using a fitness function to find the best solution. Then, the algorithm repeatedly executes the following steps until the stopping criterion is met. First, the coefficients are updated. Second, based on the random values of A and p, the algorithm updates the position of a solution using either Equation (2) or Equation (9) or Equation (6). Lastly, the WOA returns the best solution obtained.

VI. EXPERIMENT AND DISCUSSION

Our experiments aim to demonstrate the efficiency of WOA versus other existing, well-known, population-based meta- heuristic algorithms and pure computational strategies (i.e. the efficiency is described by the size of the generated test suite).

To express the computational cost performance of our strategy, a time complexity analysis of our strategy was done by considering the structure of our implementation as prescribed under Section V. The structure is displayed in Figure 7. Assuming that all other operations are carried out in a time constant, the time complexity of our strategy is $O(ExBxG) \approx O(n^3)$. The Big O notation can sometimes be used to describe execution time. However, a few studies have already computed the code execution time. There are some valid threats to comparing meta-heuristic algorithms

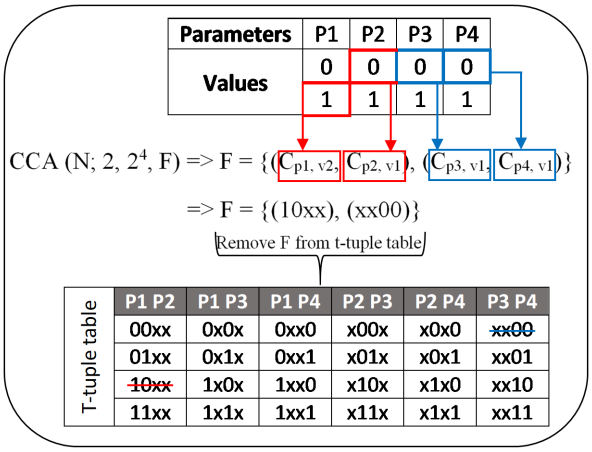


FIGURE 6. An illustration of the forbidden combinations denoted by F.

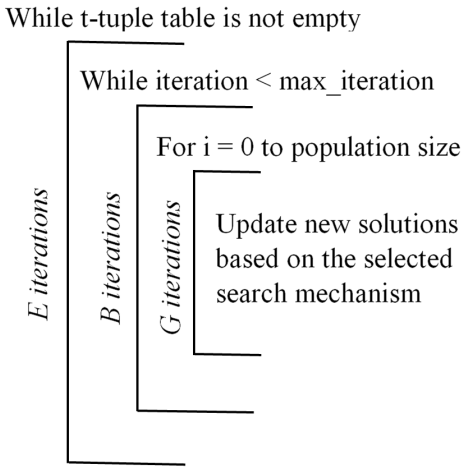


FIGURE 7. General structure of the WOA strategy.

performance specifically when execution time is compared [7]. Owing to factors such as differences in the implementation language (e.g. Java versus C versus MATLAB), the data structure, the system configuration, as well as running environment, a comparison of execution time is deemed unfair. The same observation has also been cited by other researchers [18], [53].

We split our experiments into three parts. First, we systematically tuned the parameters. Second, we evaluated and compared the WOA strategy with existing population-based meta-heuristic algorithms. Lastly, we benchmarked our strategy with existing constraint-supporting strategies. Another measurement was also applied based on Wilcoxon's signed-rank test for all reported results.

A. PARAMETER TUNING

One of the advantages of WOA is that it has a fewer number of parameters, unlike other meta-heuristic algorithms, such as PSO, HS, and GA, to name a few. However, population size and the maximum number of iterations are still required for tuning. This is because a big iteration value could be unproductive if the previous iterations did not produce a better solution. Conversely, too few iterations could perhaps prevent the best candidate solution from being reached. Comparably, a large population size raises the cost of computation; while a small one hinders a good solution from being obtained. Hence, it is necessary to carefully coordinate the selection of the maximum number of iterations and population size. 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 [54]–[56].

To tune the WOA parameters, the WOA strategy for CA(N; 2, 5^7) was executed repeatedly 20 times with a different population size and the maximum iteration number values tested, by setting the population size and varying the maximum iteration number (i.e. 10, 25, 50, 75, 100, 125, 150, 175, and 200). Then, reverse experiments were performed, where the population size was varied (i.e. 10, 30, 50, 70, 100, 120, 140, 160, 180, and 200) and the maximum iteration number was fixed. The best test suite size and the average test suite size are shown in Table 3 and Table 4, respectively, where the darkened cells indicate the most optimal size. The execution time is reported in seconds. The best execution time and the average execution time are shown in Table 3 and Table 4, respectively.

Per the results shown in Tables 3 and 4, it can be concluded that a large population size could yield better results and, on the contrary, a too-small population size could contribute to worse results. A large population size (i.e. 200) did not, however, necessarily produce better results so we had to consider that the execution time could also increase. Likewise, a high iteration value (i.e., 200) may not always provide the most optimal size in each case. The best results were obtained when the population size was set between 70 and 200. Otherwise, the iteration value would increase and the result would improve. The best result was obtained when the iteration value was varied from 75 to 175. Beyond that, when considering the best average results obtained, the population size was varied between 120 and 200 while the maximum number of iterations was varied between 100 and 175.

In Table 4, the best average results are marked in bold. We highlight two of the best average results: the first was achieved when the population size was 180 with a maximum number of iterations of 100, while the second was obtained when the population size was 180 and the maximum number of iterations was 150. In this case, we had to consider the execution time when choosing the optimal population size and maximum number of iterations. This is because the execution time increases when both the population size and the maximum number of iterations increase. Therefore, we selected 100 as the maximum number of iterations and 180 as the maximum population size.

B. BENCHMARKING WOA STRATEGY WITH EXISTING STRATEGIES

To assess the performance of WOA, we benchmarked it against other existing strategies in terms of CA size.

TABLE 3. Best test suite size and execution time acquired with varied population size and the maximum number of iterations for CA(N; 2, 5^7).

Population size	Iterations																	
	10		25		50		75		100		125		150		175		200	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
10	43	0.48	42	1.21	40	2.6	42	3.97	40	5.45	41	6.48	40	7.83	41	9.23	41	9.96
30	40	0.9	38	3.75	39	6.8	38	10.33	39	14.12	38	19.37	39	26.75	38	29.67	38	28.35
50	39	2.28	39	6.08	38	11.58	38	18.4	37	23.56	38	23.61	38	28.63	38	42.35	38	46.85
70	38	3.11	38	8.5	38	16.95	38	26.46	37	33.45	38	37.95	37	42.27	37	56.39	37	64.8
100	38	4.73	37	11.27	38	24.14	38	34.99	37	46.81	38	52.18	37	59.5	37	85.33	37	93.75
120	38	5.7	37	13.61	38	27.99	38	42.44	38	55.59	37	82.1	37	69.58	37	100.13	38	111.65
140	38	6.35	37	15.66	37	31.7	37	47.29	37	63.94	37	65.05	37	86.53	37	111.5	37	131.08
160	38	7.33	38	15.06	37	29.84	37	51.93	37	80.19	37	89.26	**36**	112.85	37	130.08	37	149.62
180	37	8.11	37	20.69	37	41.45	37	61.56	37	81.66	37	103.97	37	121.27	37	142.21	**36**	162.97
200	37	8.5	37	23.4	37	47.3	37	76.72	37	95.65	37	114.77	37	135.54	37	164.21	37	184.65

TABLE 4. Average test suite size and execution time acquired with varied population size and the maximum number of iterations for CA(N; 2, 5^7).

Population size	Iterations																	
	10		25		50		75		100		125		150		175		200	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
10	44.4	0.54	43.15	1.27	42.25	2.97	43.3	4.25	42.55	5.53	42.4	6.81	42.4	8.12	42.05	9.67	42.15	11.05
30	41.95	0.97	40.1	3.88	4.4	6.99	40.15	10.76	40.1	14.88	40	21.75	40.15	28.77	40	34.9	39.8	29.98
50	40.45	2.37	39.8	6.38	39.6	12.03	39.55	19.43	39.1	24.63	39.2	26.4	39.5	31.75	39.3	47.65	39.15	52.06
70	39.85	3.31	39.6	9.59	39.4	17.62	39.3	27.2	39.25	35.15	39.2	41.1	39.35	47.78	39.3	62.01	38.9	68.14
100	39.45	4.92	39.25	11.95	39.15	26.06	38.85	37.34	38.65	49.84	38.95	57.7	38.9	68.92	38.6	87.5	38.55	99.18
120	39.35	5.89	38.75	15.08	38.85	30.34	39.05	48.79	39	56.92	38.65	90.31	38.75	83.25	38.4	103.82	38.85	114.02
140	39.1	6.67	39.15	17.03	38.5	32.9	38.95	52.07	38.9	68.51	38.75	87.81	38.9	96.64	38.3	124.74	38.5	136.13
160	39.05	7.84	39.15	17.55	38.55	34.77	38.55	57.8	38.6	84.15	38.5	103.67	38.35	119.05	38.45	136.45	38.6	162.05
180	39.4	9.15	38.8	21.39	38.85	42.56	38.4	66.92	**38.2**	84.02	38.65	108.8	**38.2**	130.63	38.35	149.23	38.4	183.22
200	38.75	9.01	38.85	24.41	38.5	52.79	38.3	88.01	38.6	99.76	38.5	123.16	38.95	140.9	38.35	171.44	38.5	192.78

TABLE 5. Parameter values used for the existing meta-heuristic algorithms.

Algorithm	Parameters	Values
PSO [42]	Population size	80
	Acceleration coefficients	1.375
	Inertia weight	0.3
	Iterations	100
CS [4]	Population size	100
	Probability Pa	0.25
	Iterations	100
GS [57]	Population size	250
	Mutation rate	0.4
	Crossover rate	0.4
ABCVS [58]	Population size	50
	Number of cycles	80
WOA	Population size	180
	iterations	100

The experiments were divided into two of the following well-known datasets:

1) Comparing the WOA strategy with currently available strategies using CA(t, v^7), where the number of parameters remains constant while their values are varied. In addition, the interaction strength t is varied from 2 to 6.
2) Comparing the WOA strategy with existing strategies using CA($t, 3^P$), where the number of parameters is varied and their values are kept constant. In addition, the interaction strength t is varied from 2 to 6.

The experimental environment is a laptop operating on Windows 10, with a 64-bit, 2.71 GHz, an Intel Core i5 CPU, and 8 GB of RAM. The proposed strategy was coded and implemented in Java. Table 5 shows the parameter settings for each meta-heuristic algorithm used for the comparison.

In Table 6, the configurations of CA($t, 3^P$) were adopted, where t was varied as $2 \leq t \leq 6$, p was varied as $3 \leq p \leq 12$, and v was kept constant at $v = 3$; the results are reported in terms of the best test suite size, as well as average test suite size, after repeating the experiment 30 times (for statistical significance) [17]. The results reveal that WOA outperformed all the pure computational strategies and most of the AI-based strategies, including GBGA, PSO, CS, and ABCVS. Moreover, WOA produced competitive results to that of the GS and APSO strategies, bearing in mind that we used the standard WOA without any modifications.

It can be noted from Table 6 that the WOA strategy produced better results when the search space got larger, compared to other AI-based strategies, because WOA has the ability to explore more, but it lacks exploitation when it comes to a small search space.

Meanwhile, Table 7 displays the configurations of CA(t, v^7) where t is varied as $2 \leq t \leq 6$, v is varied as $2 \leq v \leq 7$ and p is kept constant at $p = 7$. The results show that the WOA strategy outperformed all the pure computational strategies and most of the AI-based strategies including PSO, CS, and APSO. In addition, WOA yielded competitive results to that of the GS strategy, although the standard WOA strategy was used.

Similarly, Table 7 also shows that the WOA strategy delivered better results with a larger search space compared to other AI-based strategies, because WOA can explore more

TABLE 6. Test suite size performance for CA(t, 3^P) where P was varied from 3 to 12 and t was varied from 2 to 6.

CA(t, 3^P)		Pure Computation Strategies					AI-based Strategies							
		Jenny [21]	TConfig [22]	PICT [59]	IPOG-D [20]	IPOG [19]	GBGA [60]	PSO [42]	GS [57]	CS [4]	APSO [61]	ABCVS [58]	WOA	
t	p	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Best	Size Best	Size Mean
2	3	**9**	10	10	15	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	9.8
	4	13	10	13	15	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	10.3
	5	14	14	13	15	15	12	12	**11**	**11**	**11**	**11**	**11**	12.86
	6	15	15	14	15	15	14	13	13	13	**12**	13	14	14.56
	7	16	15	16	15	15	15	15	**14**	**14**	15	15	**14**	15.26
	8	17	17	16	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	15.93
	9	18	17	17	**15**	**15**	16	17	**15**	**15**	16	16	16	16.7
	10	19	17	18	21	**15**	16	17	16	17	17	17	16	17.73
	11	17	20	18	21	17	17	17	**16**	18	NS	17	17	17.93
	12	19	20	19	21	21	18	18	**16**	18	NS	18	17	18.76
3	4	34	32	34	**27**	32	29	30	**27**	28	**27**	**27**	**27**	31.53
	5	40	40	43	45	41	39	39	**38**	**38**	41	**38**	**38**	40.33
	6	51	48	48	45	46	45	45	43	43	45	44	**42**	45.9
	7	51	55	51	50	55	49	50	49	**48**	**48**	49	**48**	51.6
	8	58	58	59	**50**	56	54	54	54	53	**50**	54	53	55.96
	9	62	64	63	71	63	58	58	58	58	59	58	**57**	59.7
4	5	109	97	100	162	97	**87**	96	90	94	94	98	90	100.03
	6	140	141	142	162	141	133	133	**129**	132	**129**	135	**129**	134.65
	7	169	166	168	226	167	156	155	**153**	154	154	157	154	159.46
5	6	348	305	310	386	305	**273**	312	301	304	NS	**273**	304	312.22
	7	458	477	452	678	466	433	441	**432**	434	NS	433	**432**	441.26
6	7	1089	921	1015	1201	**921**	982	977	963	973	NS	982	959	976.73
	8	1466	1515	1455	1763	1493	NS	1402	1399	1401	NS	NS	**1394**	1402.24

TABLE 7. Test suite size performance for CA(t, v^7) where v was varied from 2 to 7 and t is was varied from 2 to 6.

CA(t, v^7)		Pure Computation Strategies					AI-based Strategies					
		Jenny [21]	TConfig [22]	PICT [59]	IPOG-D [20]	IPOG [19]	PSO [42]	GS [57]	CS [4]	APSO [61]	WOA	
t	v	Best Size	Best size	Best size	Best size	Best size	Best size	Best size	Best size	Best size	Size Best	Size Mean
2	2	8	7	7	8	7	**6**	**6**	**6**	**6**	**6**	7.5
	3	16	15	16	15	15	15	**14**	15	15	**14**	15.5
	4	28	28	27	32	29	26	**24**	25	25	25	25.76
	5	37	40	40	45	45	37	36	37	**35**	36	38.56
	6	55	57	56	72	55	NS	**52**	NS	NS	**52**	54.33
	7	74	76	74	91	**49**	NS	68	NS	NS	70	72.53
3	2	14	16	15	14	16	13	**12**	**12**	15	**12**	14.2
	3	51	55	51	50	55	50	49	49	**48**	49	51.46
	4	124	**112**	124	114	**112**	116	116	117	118	116	119.36
	5	236	239	241	252	237	225	**221**	223	239	223	230.07
	6	400	423	413	470	420	NS	**374**	NS	NS	382	386.25
4	2	31	36	32	40	35	29	**27**	**27**	30	**27**	30.86
	3	169	166	168	226	167	155	153	155	153	**152**	158.46
	4	517	568	529	704	614	487	486	487	**472**	484	488.58
5	2	57	56	57	80	60	53	**51**	53	NS	52	55.73
	3	458	477	452	678	466	441	**432**	439	NS	**432**	441.58
	4	1938	**1792**	1933	2816	**1792**	1826	1821	1845	NS	1815	1823.1
6	2	87	**64**	72	96	**64**	**64**	65	66	NS	**64**	75.3
	3	1087	**921**	1015	1201	**921**	977	963	973	NS	945	969.61
	4	6127	NS	5847	5120	**4096**	5599	5608	5610	NS	5567	5567

TABLE 8. Wilcoxon test for the results reported on Tables 6 and 7.

	Ranks			Test Statistics	
	WOA>	WOA<	WOA =	Z	Asymp. Sig. (2-tailed)
GBGA	2	13	7	-1.937	0.530
PSO	1	32	7	-4.929	0.000
GS	11	12	20	-0.653	0.514
CS	2	22	16	-3.974	0.000
APSO	5	12	11	-1.599	0.110
ABCVS	2	12	8	-2.18	0.029
IPOG	8	30	5	-2.708	0.007
IPOG-D	4	37	2	-4.79	0.000
PICT	0	42	1	-5.651	0.000
TConfig	4	37	1	-3.998	0.000
Jenny	0	41	2	-5.587	0.000

in a larger search space and because exploration is one of its advantages.

To ensure the superiority of the WOA strategy over the other existing strategies, a statistical analysis was conducted, particularly the Wilcoxon signed-rank test, which is a non-parametric test for matched or coupled data concentrating on differential ratings. However, this test also considers the extent of the observed differences in response to evaluating the signs of the differences. The Wilcoxon signed-rank test was used because it can inform the researcher if a significant difference exists between two results.

The Wilcoxon signed-rank test produced two factors. The first is the Asymp. Sig. (2-tailed) and Z, which are statistical tests indicating the difference between two groups. An Asymp. Sig. (2-tailed) value smaller than 0.05 implies a significant difference between the two groups. Although the value of Z is not relevant and beyond the applicability of this study, this value was nonetheless provided in this report. The second factor is the ranking, which ranks the values that are greater than, equal to, or less than the comparable values.

In all the tables presenting the statistical results, in the ranks part, "WOA <" indicates the number of cases the WOA strategy generated with a smaller CA size compared to the other strategies (i.e., pure computational and AI-based strategies). In other words, this label indicates the number of times the WOA strategy generated better results. Similarly, "WOA =" indicates the number of times the results were the same, while "WOA >" represents the number of times the WOA strategy produced the worst results.

Table 8 presents the result of the Wilcoxon test reported in Tables 6 and 7. Table 8 shows that the WOA strategy generated better outcomes than the pure computational strategies; thus confirming the superiority of WOA over the other strategies. As for the AI-based strategies, WOA also produced significantly different outcomes compared to PSO, CS, and ABCVS. Meanwhile, WOA statistically produced competitive results to that of GBGA, GS, and APSO; but it must also be considered that these strategies have been modified and enhanced while ours was not.

TABLE 9. Parameter settings of the implemented algorithms.

Algorithm	Parameters	Values
FPA	Population size	25
	Iterations	75
	Switch probability	0.8
	Beta	1.5
SCA	Population size	25
	Iterations	75
	Magnitude	10
CS	Population size	25
	Iterations	75
	Probability ep	0.2
Jaya	Population size	25
	Iterations	75
LAHC	Population size	25
	Iterations	75
	Perturbation probability	0.25
	Direction probability	0.5
WOA	Population size	25
	Iterations	75

C. BENCHMARKING WOA STRATEGY IN THE PRESENCE OF CONSTRAINTS AGAINST FIVE DIFFERENT ALGORITHMS

In this section, we present our experiments for benchmarking WOA against 5 recent algorithms. These algorithms are the Sine-Cosine algorithm (SCA) [62], the Jaya algorithm [63], the Flower Pollination algorithm (FPA) [64], the Cuckoo Search algorithm (SC) [65], and the Late Acceptance Hill Climbing algorithm (LAHC) [66]. All the algorithms, including WOA, support constrained t-way testing. The settings of each algorithm are summarized in Table 9. We ran each algorithm 30 times and recorded the best results from these 30 runs.

The performance of the algorithms was mainly evaluated in terms of test suite size. In the evaluation, we compared the best test suite size and the average test suite size acquired by the algorithms, as per Table 10 and Table 11, respectively. Then, the Wilcoxon signed-rank test was applied to the results reported by the six algorithms.

We divided our experiments into three dataset groups. We also designed their constraints (i.e. forbidden combinations). The details of the datasets are as follows:

1) Comparing the WOA strategy with five different algorithms using CCA$(2, 3^P, F)$, where the number of parameters was varied and their values ($v = 3$) and interaction strength ($t = 2$) were kept constant. In addition, the number of constraints (i.e., forbidden combinations) were varied between 3 and 5 pairs of constraints, as shown in Tables 10 and 11.
2) Comparing the WOA strategy against five different algorithms using CCA$(2, v^7, F)$, where the number of parameters ($p = 7$) and the interaction strength

TABLE 10. Comparison of the implemented algorithms using best results of the three datasets.

	System Configurations	Forbidden Combinations (F)	Implemented AI-based Algorithms					
			LAHC	SCA	Jaya	FPA	CS	WOA
Dataset one	CCA$(2, 3^3, F)$	$F = \{(C_{p_2,v_3}, C_{p_3,v_1}), (C_{p_2,v_2}, C_{p_3,v_1}), (C_{p_1,v_1}, C_{p_3,v_2})\}$	**11**	**11**	**11**	**11**	**11**	**11**
	CCA$(2, 3^4, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_1}), (C_{p_2,v_3}, C_{p_4,v_2}), (C_{p_1,v_3}, C_{p_3,v_2}), (C_{p_1,v_2}, C_{p_2,v_1}, C_{p_4,v_3})\}$	**10**	**10**	**10**	**10**	**10**	**10**
	CCA$(2, 3^5, F)$	$F = \{(C_{p_2,v_1}, C_{p_3,v_1}), (C_{p_1,v_3}, C_{p_3,v_1}, C_{p_5,v_3}), (C_{p_2,v_2}, C_{p_4,v_2}, C_{p_5,v_1}), (C_{p_2,v_3}, C_{p_3,v_3})\}$	**11**	**11**	**11**	**11**	**11**	**11**
	CCA$(2, 3^6, F)$	$F = \{(C_{p_3,v_2}, C_{p_6,v_2}), (C_{p_5,v_3}, C_{p_6,v_1}), (C_{p_4,v_1}, C_{p_6,v_3}), (C_{p_2,v_1}, C_{p_4,v_3}, C_{p_5,v_1}, C_{p_6,v_2})\}$	**12**	**12**	13	**12**	13	**12**
	CCA$(2, 3^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_6,v_3}, C_{p_7,v_3}), (C_{p_1,v_1}, C_{p_7,v_2}), (C_{p_2,v_1}, C_{p_7,v_1}), (C_{p_1,v_2}, C_{p_6,v_2})\}$	14	14	14	14	**13**	14
	CCA$(2, 3^8, F)$	$F = \{(C_{p_3,v_1}, C_{p_4,v_2}, C_{p_5,v_3}), (C_{p_6,v_1}, C_{p_7,v_1}, C_{p_8,v_1}), (C_{p_5,v_2}, C_{p_7,v_3}, C_{p_8,v_2}), (C_{p_1,v_3}, C_{p_7,v_3}, C_{p_8,v_2})\}$	15	**14**	15	**14**	**14**	**14**
	CCA$(2, 3^9, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_2}, C_{p_6,v_2}, C_{p_7,v_2}, C_{p_9,v_1}), (C_{p_4,v_2}, C_{p_6,v_1}, C_{p_8,v_2}), (C_{p_1,v_1}, C_{p_2,v_1}, C_{p_7,v_3}, C_{p_8,v_3}, C_{p_9,v_3}), (C_{p_2,v_2}, C_{p_4,v_1}, C_{p_8,v_2})\}$	15	15	15	15	**14**	**14**
	CCA$(2, 3^{10}, F)$	$F = \{(C_{p_8,v_1}, C_{p_9,v_1}, C_{p_{10},v_1}), (C_{p_2,v_2}, C_{p_6,v_1}, C_{p_8,v_3}), (C_{p_4,v_3}, C_{p_6,v_3}, C_{p_8,v_3}, C_{p_{10},v_3}), (C_{p_1,v_2}, C_{p_5,v_1}, C_{p_6,v_2}, C_{p_6,v_3}, C_{p_7,v_3}, C_{p_9,v_2}, C_{p_{10},v_1})\}$	17	**16**	**16**	**16**	**16**	**16**
Dataset two	CCA$(2, 2^7, F)$	$F = \{(C_{p_1,v_1}, C_{p_2,v_1}), (C_{p_4,v_1}, C_{p_5,v_2})\}$	**7**	**7**	**7**	**7**	**7**	**7**
	CCA$(2, 3^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_6,v_3}, C_{p_7,v_3}), (C_{p_2,v_1}, C_{p_3,v_3}, C_{p_6,v_2})\}$	14	14	**13**	14	14	14
	CCA$(2, 4^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_4}), (C_{p_5,v_4}, C_{p_4,v_2}, C_{p_7,v_3}), (C_{p_7,v_1}, C_{p_2,v_1}, C_{p_1,v_2}), (C_{p_1,v_1}, C_{p_2,v_1})\}$	25	24	**23**	24	24	**23**
	CCA$(2, 5^7, F)$	$F = \{(C_{p_1,v_5}, C_{p_2,v_5}), (C_{p_3,v_4}, C_{p_4,v_3}), (C_{p_5,v_3}, C_{p_6,v_5}), (C_{p_6,v_1}, C_{p_7,v_1}), (C_{p_5,v_1}, C_{p_3,v_3}, C_{p1,v2}, C_{p7,v4})\}$	39	35	**33**	36	36	34
	CCA$(2, 6^7, F)$	$F = \{(C_{p_1,v_6}, C_{p_3,v_5}, C_{p_4,v_4}), (C_{p_6,v_6}, C_{p_7,v_6}), (C_{p_2,v_4}, C_{p_5,v_3}), (C_{p_1,v_2}, C_{p_4,v_1}, C_{p_6,v_2}, C_{p_5,v_2})\}$	54	**49**	**49**	**49**	51	**49**
	CCA$(2, 7^7, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_2}, C_{p_3,v_1}, C_{p_4,v_1}), (C_{p_7,v_7}, C_{p_6,v_7}, C_{p_5,v_6}, C_{p_4,v_6}), (C_{p_4,v_3}, C_{p_5,v_4}, C_{p_6,v_5}), (C_{p_5,v_1}, C_{p_7,v_5})\}$	75	**65**	66	67	71	66
	CCA$(2, 8^7, F)$	$F = \{(C_{p_1,v_4}, C_{p_2,v_4}, C_{p_5,v_6}, C_{p_6,v_6}, C_{p_7,v_8}), (C_{p_2,v_3}, C_{p_3,v_3}, C_{p_4,v_2}, C_{p_5,v_2}, C_{p_6,v_1}, C_{p_7,v_1}), (C_{p_3,v_5}, C_{p_4,v_5}), (C_{p_6,v_7}, C_{p_5,v_7})\}$	98	**85**	86	86	93	**85**
Dataset three	CCA$(2, 2^{10}, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_5,v_2}, C_{p_7,v_1}, C_{p_8,v_2})\}$	**8**	**8**	**8**	**8**	**8**	**8**
	CCA$(3, 2^{10}, F)$	$F = \{(C_{p_1,v_1}, C_{p_2,v_2}, C_{p_4,v_1}), (C_{p_5,v_2}, C_{p_7,v_1}, C_{p_8,v_2})\}$	17	**16**	**16**	17	17	17
	CCA$(4, 2^{10}, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_2}, C_{p_3,v_2}, C_{p_4,v_1}), (C_{p_6,v_1}, C_{p_7,v_1}, C_{p_9,v_2}, C_{p_{10},v_2})\}$	**32**	34	33	34	34	**32**
	CCA$(5, 2^{10}, F)$	$F = \{(C_{p_8,v_1}, C_{p_8,v_2}, C_{p_{10},v_1}), (C_{p_4,v_2}, C_{p_6,v_2}, C_{p_7,v_2}), (C_{p_2,v_1}, C_{p_3,v_1})\}$	81	77	**74**	77	79	**74**
	CCA$(6, 2^{10}, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_1}, C_{p_3,v_1}, C_{p_8,v_1}, C_{p_9,v_1}, C_{p_{10},v_1})\}$	157	158	**154**	157	**154**	**154**

$(t = 2)$ were kept constant and their values were varied. In addition, the number of constraints (i.e. forbidden combinations) was varied between 3 and 5 pairs of constraints, as shown in Tables 10 and 11.

3) Comparing the WOA strategy against five different algorithms using CCA$(t, 2^{10}, F)$, where the number of parameters and their values were kept constant ($p = 2$ and $v = 10$). While the interaction strength t was varied from 2 to 6. In addition, the number of constraints (i.e, forbidden combinations) were varied between 1 and 3 pairs of constraints, as shown in Tables 10 and 11.

To evaluate the performance of the WOA strategy, we compared WOA against six other t-way strategies that were also implemented. The performance evaluation criteria included size (i.e., optimal test suite size) [50]. The experimental results are presented in Tables 10 and 11; Table 10 shows the minimum (i.e., best) test suite size while Table 11 shows the average suite size for each competing strategy. The best results obtained by each strategy are marked in bold.

The results of Tables 10 and 11 show that WOA performed better than LAHC, FPA, and CS for both best test suite and average test suite, as WOA favors exploration, which allows it to explore more especially when the search space gets larger, while the CS performance achieved slower convergence that led to failure to perform well. FPA, meanwhile, lacks exploration. Comparing WOA with SCA, WOA had better average test suite results than SCA and produced competitive results in terms of best test suite because SCA has a good local search ability but lacks the global search ability. Lastly, WOA and Jaya produced competitive results when compared to each other. In terms of average results, Jaya achieved better results more frequently than WOA because of its ability to balance between global search and local search, as it is a parameter-free algorithm.

Statistically, the Wilcoxon signed-rank test was applied to the results reported in Table 10. Table 12 presents the outcomes of the Wilcoxon signed-rank test. Statistically, WOA produced better test suite sizes than LAHC, FPA and CS, with the exception of the Jaya and SCA algorithms. On a positive note, WOA managed to produce better results more frequently than Jaya and SCA while most cases showed equal results.

Additionally, the Wilcoxon test reported the results in Table 11 and the outcomes of the Wilcoxon test are shown in Table 13. WOA produced results that are significantly different from that of LAHC, SCA and FPA, with the exception of the Jaya and CS algorithms. This is because of WOA's exploration advantage while LAHC, SCA and FPA lack this ability.

TABLE 11. Comparison of the implemented algorithms using average results of the three datasets.

	System Configurations	Forbidden Combinations (F)	Implemented AI-based Algorithms					
			LAHC	SCA	Jaya	FPA	CS	WOA
Dataset one	CCA$(2, 3^3, F)$	$F = \{(C_{p_2,v_3}, C_{p_3,v_1}), (C_{p_2,v_2}, C_{p_3,v_1}), (C_{p_1,v_1}, C_{p_3,v_2})\}$	11.13	**11**	**11**	**11**	**11**	**11**
	CCA$(2, 3^4, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_1}), (C_{p_2,v_3}, C_{p_4,v_2}), (C_{p_1,v_3}, C_{p_3,v_2}), (C_{p_1,v_2}, C_{p_2,v_1}, C_{p_4,v_3})\}$	11.23	**10.86**	10.96	10.93	10.97	10.93
	CCA$(2, 3^5, F)$	$F = \{(C_{p_2,v_1}, C_{p_3,v_1}), (C_{p_1,v_3}, C_{p_3,v_1}, C_{p_5,v_3}), (C_{p_2,v_2}, C_{p_4,v_2}, C_{p_5,v_1}), (C_{p_2,v_3}, C_{p_3,v_3})\}$	12.43	12.06	**12**	12.26	12.03	**12**
	CCA$(2, 3^6, F)$	$F = \{(C_{p_3,v_2}, C_{p_6,v_2}), (C_{p_5,v_3}, C_{p_6,v_1}), (C_{p_4,v_1}, C_{p_6,v_3}), (C_{p_2,v_1}, C_{p_4,v_3}, C_{p_5,v_1}, C_{p_6,v_2})\}$	13.76	13.7	13.63	13.73	**13.53**	13.63
	CCA$(2, 3^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_6,v_3}, C_{p_7,v_3}), (C_{p_1,v_1}, C_{p_7,v_2}), (C_{p_2,v_1}, C_{p_7,v_1}), (C_{p_1,v_2}, C_{p_6,v_2})\}$	15.66	15.16	**15.03**	15.2	15.1	15.13
	CCA$(2, 3^8, F)$	$F = \{(C_{p_3,v_1}, C_{p_4,v_2}, C_{p_5,v_3}), (C_{p_6,v_1}, C_{p_7,v_1}, C_{p_8,v_1}), (C_{p_5,v_2}, C_{p_7,v_3}, C_{p_8,v_2}), (C_{p_1,v_3}, C_{p_7,v_3}, C_{p_8,v_2})\}$	15.83	15.2	15.33	15.33	**15.13**	15.2
	CCA$(2, 3^9, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_2}, C_{p_6,v_2}, C_{p_7,v_2}, C_{p_9,v_1}), (C_{p_4,v_2}, C_{p_6,v_1}, C_{p_8,v_2}), (C_{p_1,v_1}, C_{p_2,v_1}, C_{p_7,v_3}, C_{p_8,v_3}, C_{p_9,v_3}), (C_{p_2,v_2}, C_{p_4,v_1}, C_{p_8,v_2})\}$	16.66	15.93	15.66	15.86	15.76	**15.63**
	CCA$(2, 3^{10}, F)$	$F = \{(C_{p_8,v_1}, C_{p_9,v_1}, C_{p_{10},v_1}), (C_{p_2,v_2}, C_{p_6,v_1}, C_{p_8,v_3}), (C_{p_4,v_3}, C_{p_6,v_3}, C_{p_8,v_3}, C_{p_{10},v_3}), (C_{p_1,v_2}, C_{p_5,v_1}, C_{p_6,v_2}, C_{p_6,v_3}, C_{p_7,v_3}, C_{p_9,v_2}, C_{p_{10},v_1})\}$	17.4	16.7	**16.36**	16.8	16.76	16.46
Dataset two	CCA$(2, 2^7, F)$	$F = \{(C_{p_1,v_1}, C_{p_2,v_1}), (C_{p_4,v_1}, C_{p_5,v_2})\}$	7.33	7.3	7.26	7.33	7.2	**7.16**
	CCA$(2, 3^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_6,v_3}, C_{p_7,v_3}), (C_{p_2,v_1}, C_{p_3,v_3}, C_{p_6,v_2})\}$	15.3	**14.76**	14.93	15.03	**14.76**	14.9
	CCA$(2, 4^7, F)$	$F = \{(C_{p_3,v_2}, C_{p_4,v_4}), (C_{p_5,v_4}, C_{p_4,v_2}, C_{p_7,v_3}), (C_{p_7,v_1}, C_{p_2,v_1}, C_{p_1,v_2}), (C_{p_1,v_1}, C_{p_2,v_1})\}$	26.2	**24.75**	24.93	25.1	25	24.93
	CCA$(2, 5^7, F)$	$F = \{(C_{p_1,v_5}, C_{p_2,v_5}), (C_{p_3,v_4}, C_{p_4,v_3}), (C_{p_5,v_3}, C_{p_6,v_5}), (C_{p_6,v_1}, C_{p_7,v_1}), (C_{p_5,v_1}, C_{p_3,v_3}, C_{p1,v2}, C_{p7,v4})\}$	40.56	36.5	**36.24**	37.16	37.43	36.83
	CCA$(2, 6^7, F)$	$F = \{(C_{p_1,v_6}, C_{p_3,v_5}, C_{p_4,v_4}), (C_{p_6,v_6}, C_{p_7,v_6}), (C_{p_2,v_4}, C_{p_5,v_3}), (C_{p_1,v_2}, C_{p_4,v_1}, C_{p_6,v_2}, C_{p_5,v_2})\}$	57.66	50.53	**50.23**	51.13	53.53	50.63
	CCA$(2, 7^7, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_2}, C_{p_3,v_1}, C_{p_4,v_1}), (C_{p_7,v_7}, C_{p_6,v_7}, C_{p_5,v_6}, C_{p_4,v_6}), (C_{p_4,v_3}, C_{p_5,v_4}, C_{p_6,v_5}), (C_{p_5,v_1}, C_{p_7,v_5})\}$	77.56	67.8	67.56	68.7	72.43	**67.43**
	CCA$(2, 8^7, F)$	$F = \{(C_{p_1,v_4}, C_{p_2,v_4}, C_{p_5,v_6}, C_{p_6,v_6}, C_{p_7,v_8}), (C_{p_2,v_3}, C_{p_3,v_3}, C_{p_4,v_2}, C_{p_5,v_2}, C_{p_6,v_1}, C_{p_7,v_1}), (C_{p_3,v_5}, C_{p_4,v_5}), (C_{p_6,v_7}, C_{p_5,v_7})\}$	100.26	87.56	**87.1**	88.4	94.5	87.26
Dataset three	CCA$(2, 2^{10}, F)$	$F = \{(C_{p_2,v_2}, C_{p_3,v_2}), (C_{p_5,v_1}, C_{p_6,v_1}), (C_{p_5,v_2}, C_{p_7,v_1}, C_{p_8,v_2})\}$	8.33	8.29	8.23	8.17	8.13	**8.13**
	CCA$(3, 2^{10}, F)$	$F = \{(C_{p_1,v_1}, C_{p_2,v_2}, C_{p_4,v_1}), (C_{p_5,v_2}, C_{p_7,v_1}, C_{p_8,v_2})\}$	18.26	18.36	**17.66**	17.99	17.9	17.96
	CCA$(4, 2^{10}, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_2}, C_{p_3,v_2}, C_{p_4,v_1}), (C_{p_6,v_1}, C_{p_7,v_1}, C_{p_9,v_2}, C_{p_{10},v_2})\}$	40	39.46	**39.13**	39.72	39.5	39.3
	CCA$(5, 2^{10}, F)$	$F = \{(C_{p_8,v_1}, C_{p_8,v_2}, C_{p_{10},v_1}), (C_{p_4,v_2}, C_{p_6,v_2}, C_{p_7,v_2}), (C_{p_2,v_1}, C_{p_3,v_1})\}$	84.53	84.5	80.3	83	80.15	**79.78**
	CCA$(6, 2^{10}, F)$	$F = \{(C_{p_1,v_2}, C_{p_2,v_1}, C_{p_3,v_1}, C_{p_8,v_1}, C_{p_9,v_1}, C_{p_{10},v_1})\}$	161.56	161.60	**157.30**	161.29	158.46	158.79

TABLE 12. Wilcoxon test for the results reported on Table 10.

	Ranks			Test Statistics	
	WOA>	WOA<	WOA =	Z	Asymp. Sig. (2-tailed)
LAHC	0	10	10	-2.812	0.005
SCA	2	6	12	-1.723	0.085
Jaya	3	5	12	-0.707	0.480
FPA	0	8	12	-2.558	0.011
CS	1	8	11	-2.448	0.014

TABLE 13. Wilcoxon test for the results reported on Table 11.

	Ranks			Test Statistics	
	WOA>	WOA<	WOA =	Z	Asymp. Sig. (2-tailed)
LAHC	0	20	0	-3.921	0.000
SCA	5	13	2	-2.026	0.043
Jaya	8	8	4	-1.037	0.300
FPA	0	18	2	-3.724	0.000
CS	6	12	2	-1.808	0.071

Side by side, our approach is comparable to that of the Binary Decision Diagram (BDD) [67] and SATSolver [68]. BDD exploits a decision diagram to ensure restrictions are turned into constraints. Although useful, the BDD approach is known to suffer from a state explosion problem, which can potentially limit the size of the constrained configuration. SATSolver addresses the aforementioned limitation of BDD but to the expense of large overheads due to the extensive use of the Conjunctive Normal Form (CNF) to represent the constraints. On a positive note, both approaches implicitly guarantee backtrack-freeness (i.e. dead-end constraint satisfiability during the configuration process). Our approach excels in terms of simplicity as compared to both BDD and SATSolver, although the backtrack-freeness must be explicitly checked every time a new solution is generated.

VII. CONCLUSION

A convincing review of the most current approaches for t-way testing was presented. Plus, the recently developed Whale Optimization Algorithm (WOA) was presented for implementation in current state-of-the-art constrained and unconstrained t-way testing and its implementation explained step by step.

In terms of overall performance, WOA showed competitive results to that of well-known AI-based metaheuristics from the literature, bearing in mind that we used the original WOA while the other methods had been modified. Additionally, we designed our own constraints on well-known CAs and implemented six recently-developed AI-based algorithms, including WOA, to comprehensively compare and evaluate the performance of each. The results showed that WOA outperformed most of the AI- based strategies and all of the pure computational strategies. Moreover, WOA showed consistent (i.e. no odd outlier result) overall performance.

In a future work, given our promising results, we expect to expand our approach as a multi-objective optimization method for combinatorial testing. As NSGA-II, four parameters will be taken into consideration to assess the optimality of the test suit, which are the test suit size, the test case priority, the test case frequency, and the test case constraints [69]. The aim is to create a test suit with decreased size and increased priority.

Additionally, we will enhance WOA by either hybridization or by combining it with other meta-heuristics because two main drawbacks of WOA were noted from the experiments: the first drawback is that its adaptive parameter depends on random distribution while the second drawback is that WOA suffers from premature convergence like any other meta-heuristic (evolutionary and swarm) algorithm.

ACKNOWLEDGMENT

ALI ABDULLAH HASSAN would like to thank Hadhramout Foundation, Yemen, for his support in tuition fees.

REFERENCES

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

[2] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 6, pp. 418–421, Jun. 2004.

[3] X. Chen, Q. Gu, J. Qi, and D. Chen, "Applying particle swarm optimization to pairwise testing," in *Proc. IEEE 34th Annu. Comput. Softw. Appl. Conf.*, Jul. 2010, pp. 107–116.

[4] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Inf. Softw. Technol.*, vol. 66, pp. 13–29, Oct. 2015.

[5] A. B. Nasser, Y. A. Sariera, A. A. Alsewari, K. Z. Zamli, "Assessing optimization based strategies for t-way test suite generation: The case for flower-based strategy," in *Proc. IEEE Int. Conf. Control Syst., Comput. Eng. (ICCSCE)*, Nov. 2015, pp. 150–155.

[6] N. Ramli, R. R. Othman, and M. S. A. R. Ali, "Optimizing combinatorial input-output based relations testing using ant colony algorithm," in *Proc. 3rd Int. Conf. Electron. Design (ICED)*, Aug. 2016, pp. 586–590.

[7] K. Z. Zamli, B. Y. Alkazemi, and G. Kendall, "A tabu search hyper-heuristic strategy for t-way test suite generation," *Appl. Soft Comput.*, vol. 44, pp. 57–74, Jul. 2016.

[8] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[9] Y. Sun, X. Wang, Y. Chen, and Z. Liu, "A modified whale optimization algorithm for large-scale global optimization problems," *Expert Syst. Appl.*, vol. 114, pp. 563–577, Dec. 2018.

[10] Y. Miao, M. Zhao, V. Makis, and J. Lin, "Optimal swarm decomposition with whale optimization algorithm for weak feature extraction from multicomponent modulation signal," *Mech. Syst. Signal Process.*, vol. 122, pp. 673–691, May 2019.

[11] V. Tiwari and S. C. Jain, "Histopathological image classification using efficient bag-of-features and whale optimization algorithm," in *Proc. Int. Conf. Sustain. Comput. Sci., Technol. Manage. (SUSCOM)*. Jaipur, India: Amity Univ. Rajasthan, Mar. 2019. [Online]. Available: https://ssrn.com/abstract=3356718

[12] T. Jiang, C. Zhang, and Q.-M. Sun, "Green job shop scheduling problem with discrete whale optimization algorithm," *IEEE Access*, vol. 7, pp. 43153–43166, 2019.

[13] N. M. Laskar, K. Guha, I. Chatterjee, S. Chanda, K. L. Baishnab, and P. K. Paul, "HWPSO: A new hybrid whale-particle swarm optimization algorithm and its application in electronic design optimization problems," *Int. J. Speech Technol.*, vol. 49, no. 1, pp. 265–291, Jan. 2019.

[14] G. Xiong, J. Zhang, D. Shi, and Y. He, "Parameter extraction of solar photovoltaic models using an improved whale optimization algorithm," *Energy Convers. Manage.*, vol. 174, pp. 388–405, Oct. 2018.

[15] F. Mohamed, M. Abdel-Nasser, K. Mahmoud, and S. Kamel, "Economic dispatch using stochastic whale optimization algorithm," in *Proc. Int. Conf. Innov. Trends Comput. Eng. (ITCE)*, Feb. 2018, pp. 19–24.

[16] S. Singh and R. Shree, "An analysis of test suite minimization techniques," *Int. J. Eng. Sci. Res. Technol.*, vol. 5, pp. 252–260, Feb. 2016.

[17] K. Z. Zamli, F. Din, A. B. Nasser, and A. Alsewari, "Combinatorial test suite generation strategy using enhanced sine cosine algorithm," in *Proc. ECCE*. Singapore: Springer, 2020, pp. 127–137.

[18] K. Z. Zamli, F. Din, S. Baharom, and B. S. Ahmed, "Fuzzy adaptive teaching learning-based optimization strategy for the problem of generating mixed strength t-way test suites," *Eng. Appl. Artif. Intell.*, vol. 59, pp. 35–50, Mar. 2017.

[19] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for T-Way software testing," in *Proc. 14th Annu. IEEE Int. Conf. Workshops Eng. Computer-Based Syst. (ECBS)*, Mar. 2007, pp. 549–556.

[20] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing," *Softw. Test., Verification Rel.*, 18, no. 3, pp. 125–148, 2008.

[21] B. Jenkins. (2016). *Jenny Test Tool*. [Online]. Available: http://www.burtleburtle.net./bob/math/jenny.html

[22] A. W. Williams, "Determination of test configurations for pair-wise interaction coverage," in *Testing of Communicating Systems*. Hoboken, NJ, USA: Wiley, 2000, pp. 59–74.

[23] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowl.-Based Syst.*, vol. 89, pp. 228–249, Nov. 2015.

[24] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Reading, MA, USA: Addison-Wesley, 1989.

[25] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[26] D. Simon, "Biogeography-based optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, Dec. 2008.

[27] A. Farasat, M. B. Menhaj, T. Mansouri, and M. R. S. Moghadam, "ARO: A new model-free optimization algorithm inspired from asexual reproduction," *Appl. Soft Comput.*, vol. 10, no. 4, pp. 1284–1292, Sep. 2010.

[28] J. Kennedy, R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.

[29] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, Oct. 2007.

[30] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO)*, Springer, 2010, pp. 65–74.

[31] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, Dec. 2009, pp. 210–214.

[32] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[33] S. Mirjalili, "The ant lion optimizer," *Adv. Eng. Softw.*, vol. 83, pp. 80–98, May 2015.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[35] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, Jun. 2009.

[36] B. Alatas, "ACROA: Artificial chemical reaction optimization algorithm for global optimization," *Expert Syst. Appl.*, vol. 38, no. 10, pp. 13170–13180, Sep. 2011.

[37] V. K. Patel and V. J. Savsani, "Heat transfer search (HTS): A novel optimization algorithm," *Inf. Sci.*, vol. 324, pp. 217–246, Dec. 2015.

[38] F. A. Hashim, E. H. Houssein, M. S. Mabrouk, W. Al-Atabany, and S. Mirjalili, "Henry gas solubility optimization: A novel physics-based algorithm," *Future Gener. Comput. Syst.*, vol. 101, pp. 646–667, Dec. 2019.

[39] H. N. Nsaif Al-Sammarraie and D. N. A. Jawawi, "Multiple black hole inspired meta-heuristic searching optimization for combinatorial testing," *IEEE Access*, vol. 8, pp. 33406–33418, 2020.

[40] N. Ramli, R. R. Othman, Z. I. A. Khalib, and M. Jusoh, "A review on recent T-way combinatorial testing strategy," in *Proc. MATEC Web Conf.*, vol. 140, 2017, Art. no. 01016.

[41] J. Lin, C. Luo, S. Cai, K. Su, D. Hao, and L. Zhang, "TCA: An efficient two-mode meta-heuristic algorithm for combinatorial test generation (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 494–505.
[42] H. Wu, C. Nie, F.-C. Kuo, H. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 575–591, Aug. 2015.
[43] K. Rabbi, Q. Mamun, and M. R. Islam, "An efficient particle swarm intelligence based strategy to generate optimum test data in t-way testing," in *Proc. IEEE 10th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2015, pp. 123–128.
[44] Y. Alsariera, A. Nasser, and K. Zamli, "Benchmarking of Bat-inspired interaction testing strategy," *Int. J. Comput. Sci. Inf. Eng. (IJCSIE)*, vol. 7, pp. 71–79, May 2016.
[45] A. K. Alazzawi, A. A. B. Homaid, A. A. Alomoush, and A. A. Alsewari, "Artificial bee colony algorithm for pairwise test generation," *J. Telecommun., Electron. Comput. Eng.*,vol. 9, nos. 1–2, pp. 103–108, 2017.
[46] A. K. Alazzawi, H. Md Rais, and S. Basri, "Artificial bee colony algorithm for t-Way test suite generation," in *Proc. 4th Int. Conf. Comput. Inf. Sci. (ICCOINS)*, Aug. 2018, pp. 1–6.
[47] F. Din and K. Z. Zamli, "Fuzzy adaptive teaching learning-based optimization strategy for pairwise testing," in *Proc. 7th IEEE Int. Conf. Syst. Eng. Technol. (ICSET)*, Oct. 2017, pp. 17–22.
[48] A. A. Alsewari, L. M. Xuan, and K. Z. Zamli, "Firefly combinatorial testing strategy," in *Proc. Sci. Inf. Conf.* Cham, Switzerland: Springer, 2018, pp. 936–944.
[49] 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, Oct. 2018.
[50] 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 *Proc. Int. Conf. Reliable Inf. Commun. Technol.* Cham, Switzerland: Springer, 2019, pp. 352–361.
[51] B. S. Ahmed and K. Z. Zamli, "PSTG: A T-Way strategy adopting particle swarm optimization," in *Proc. 4th Asia Int. Conf. Math./Anal. Model. Comput. Simul.*, 2010, pp. 1–5.
[52] S. K. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified ann and whale optimization algorithm," *Cluster Comput.*, vol. 22, no. S5, pp. 11425–11434, Sep. 2019.
[53] B. S. Ahmed, L. M. Gambardella, W. Afzal, and K. Z. Zamli, "Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading," *Inf. Softw. Technol.*, vol. 86, pp. 20–36, Jun. 2017.
[54] J. Stardom, "Metaheuristics and the search for covering and packing arrays [microform]," Ph.D. dissertation, Dept. Math., Simon Fraser Univ., Burnaby, BC, Canada, 2001.
[55] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a t-way interaction test suite using the particle swarm optimization approach," *Int. J. Innov. Comput., Inf. Control*, vol. 8, no. 1, pp. 431–452, 2012.
[56] 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, Jun. 2012.
[57] 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, pp. 165–185, Feb. 2018.
[58] A. K. Alazzawi, H. Md, and S. Basri, "ABCVS: An artificial bee colony for generating variable T-Way test sets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 4, pp. 259–274, 2019.
[59] J. Czerwonka, D. Butt, and C. Gens, "Pairwise testing in real word: practical extensions to test case generators," in *Proc. 24th Pacific Northwest Softw. Qual. Conf.*, 2006, pp. 419–430.
[60] J. Torres-Jimenez and J. C. Perez-Torres, "A greedy algorithm to construct covering arrays using a graph representation," *Inf. Sci.*, vol. 477, pp. 234–245, Mar. 2019.
[61] 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 Syst. Appl.*, vol. 42, no. 22, pp. 8753–8765, Dec. 2015.
[62] S. Mirjalili, "SCA: A sine cosine algorithm for solving optimization problems," *Knowl.-Based Syst.*, vol. 96, pp. 120–133, Mar. 2016.
[63] R. Venkata Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *Int. J. Ind. Eng. Comput.*, vol. 7, no. 1, pp. 19–34, 2016.
[64] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Proc. Int. Conf. Unconventional Comput. Natural Comput.* Berlin, Germany: Springer, 2012, pp. 240–249.
[65] A. H. Gandomi, X.-S. Yang, and A. H. Alavi, "Cuckoo search algorithm: A Metaheuristic approach to solve structural optimization problems," *Eng. with Comput.*, vol. 29, no. 1, pp. 17–35, Jan. 2013.
[66] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *Eur. J. Oper. Res.*, vol. 258, no. 1, pp. 70–78, Apr. 2017.
[67] I. Segall, R. Tzoref-Brill, and E. Farchi, "Using binary decision diagrams for combinatorial test design," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2011, pp. 254–264.
[68] A. Yamada, A. Biere, C. Artho, T. Kitamura, and E.-H. Choi, "Greedy combinatorial test case generation using unsatisfiable cores," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2016, pp. 614–624.
[69] A. Sabbaghi and M. R. Keyvanpour, "A novel approach for combinatorial test case generation using multi objective optimization," in *Proc. 7th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2017, pp. 411–418.

ALI ABDULLAH HASSAN was born in Hadhramout, Yemen, in 1987. He received the B.Sc. degree in computer science from the International Islamic University Malaysia, in 2012, and the M.Sc. degree in computer science from Universiti Teknologi Malaysia, in 2014. Since 2018, he has been a Researcher with the Data Mining and Optimization Research Group, Center for Artificial Intelligence Technology, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia (UKM), Malaysia. His research interests include artificial intelligence, particularly machine learning, optimization algorithms, and Software testing applications.

SALWANI ABDULLAH received the B.Sc. degree in computer science from Universiti Teknologi Malaysia, the master's degree specializing in computer science from Universiti Kebangsaan Malaysia (UKM), and the Ph.D. degree in computer science from The University of Nottingham, U.K. She is currently a Professor of computational optimization with the Faculty of Information Science and Technology, UKM. Her research interests include artificial intelligence and operation research, particularly computational optimization algorithms (heuristic and meta-heuristic, evolutionary algorithms, local search) that involve different real world applications for single and multi-objective continuous and combinatorial optimization problems, such as timetabling, scheduling, routing, nurse rostering, dynamic optimization, data mining problems (feature selection, clustering, classification, time-series prediction), intrusion detection, and search-based software testing.

KAMAL Z. ZAMLI (Member, IEEE) received the B.Sc. degree in electrical engineering from the Worcester Polytechnic Institute, USA, in 1992, the M.Sc. degree in real-time software engineering from Universiti Teknologi Malaysia, in 2000, and the Ph.D. degree in software engineering from Newcastle University, Newcastle upon Tyne, U.K., in 2003. He is currently a Professor attached to the Faculty of Computing, University Malaysia Pahang. His main research interests include search-based software engineering, combinatorial software testing, and computational intelligence.

ROZILAWATI RAZALI is currently an Associate Professor in information system with the Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia. Her research interests include software engineering, human–computer interaction, and information systems (business informatics).