IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 June 2022, accepted 28 June 2022, date of publication 6 July 2022, date of current version 12 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3188856

METHODS

Construction of Prioritized T-Way Test Suite Using Bi-Objective Dragonfly Algorithm

MASHUK AHMED[1], ABDULLAH B. NASSER[2], AND KAMAL Z. ZAMLI[1], (Member, IEEE)
[1]Faculty of Computing, Universiti Malaysia Pahang, Pekan, Pahang 26600, Malaysia
[2]School of Technology and Innovation, University of Vaasa, 65200 Vaasa, Finland

Corresponding author: Mashuk Ahmed (mashuk1203095@gmail.com)

The research work undertaken in this paper is supported and funded by Universiti Malaysia Pahang under the Fundamental Research Grant: FORMULATION OF BI-OBJECTIVE ELITIST DRAGONFLY ALGORITHM (BIDA) FOR CONSTRUCTING PRIORITIZED T-WAY TEST CASES (FRGS/1/2019/ICT02/UMP/02/13 (RDU1901209)) from the Ministry of Higher Education Malaysia.

ABSTRACT Software testing is important for ensuring the reliability of software systems. In software testing, effective test case generation is essential as an alternative to exhaustive testing. For improving the software testing technology, the t-way testing technique combined with metaheuristic algorithm has been great to analyze a large number of combinations for getting optimal solutions. However, most of the existing t-way strategies consider test case weights while generating test suites. Priority of test cases hasn't been fully considered in previous works, but in practice, it's frequently necessary to distinguish between high-priority and low-priority test cases. Therefore, the significance of test case prioritization is quite high. For this reason, this paper has proposed a t-way strategy that implements an adaptive Dragonfly Algorithm (DA) to construct prioritized t-way test suites. Both test case weight and test case priority have equal significance during test suite generation in this strategy. We have designed and implemented a Bi-objective Dragonfly Algorithm (BDA) for prioritized t-way test suite generation, and the two objectives are test case weight and test case priority. The test results demonstrate that BDA performs competitively against existing t-way strategies in terms of test suite size, and in addition, BDA generates prioritized test suites.

INDEX TERMS Bi-objective, dragonfly algorithm, multi-objective optimization, prioritized test suite, test case priority, t-way testing.

I. INTRODUCTION

Software testing and optimization are essential while evaluating the robustness of software systems. A software system needs to meet various requirements of the users. In software testing, the test case generation problem is an essential problem [1]. Exhaustive testing isn't so effective due to using every combination of input values. Therefore, exhaustively testing all inputs or outputs can cause combinatorial explosion problems. As alternative approaches, random testing is used to generate random test cases, however, it isn't also so effective as it can't ensure the required optimal test cases. Another option, parallel testing, has cost issues and often requires higher resources. There's also partition testing.

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

In above-mentioned methods, detecting faults because of interaction is difficult as there's no provision. However, t-way testing provides solutions to many of these mentioned problems. T-way testing has come into use at present due to its effective sampling strategy. 't' means the interaction strength in the term 't-way.' This testing method generates test suites and caters bugs due to input parameters' interactions [2]. In test suite construction through t-way testing, the testing strategy follows special methods (mathematical methods or computational methods) to keep the test suite size as small as possible and also covers all interactions which involve specific combinations of parameters' values for specified interaction strength. T-way testing has 2 categories: OTAT or one-test-at-a-time and OPAT or one parameter-at-a-time. Recent t-way testing focuses not only on test case weights but also on test case priorities. Prioritized t-way testing

helps to solve multi-objective optimization problems more effectively.

In t-way testing, metaheuristic algorithms are effective for sampling optimized test suite sets from a huge number of combinatorial values with a specific interaction strength. There are various metaheuristic algorithms such as Genetic Algorithm (GA), Ant Colony Algorithm (ACA), Flower Pollination Algorithm (FPA), Cuckoo Search (CS) Algorithm, Bee Algorithm, Particle Swarm Optimization (PSO), Dragonfly Algorithm (DA), Hill Climbing (HC), Harmony Search (HS), Sine Cosine Algorithm (SCA), Simulated Annealing (SA), Tabu Search (TS), Teaching Learning Based Algorithm (TLBO), etc. Metaheuristic algorithms have been proven to be flexible and simple. Also, these algorithms have mechanisms without deprivation [3]. Metaheuristic algorithms effectively identify optimal test cases from exhaustive test suites. Dragonfly Algorithm (DA), which is based on swarms, has also been effective in optimal test case generation.

Different metaheuristic algorithms show different characteristics while implementing for t-way testing, where the focus remains on test case weights most of the time. The existing t-way testing methods based on metaheuristic algorithms generate optimal test suites by focusing mostly on test case weights. These methods consider the t-way interactions having the same priority. But this isn't practical in the reality due to other factors. The testing methods have to distinguish between high priority combinations and low priority combinations so that we can generate highly prioritized test cases. However, when there's the question about prioritized test suite generation where there are more than one objectives (test case weight and test case priority), there aren't promising t-way strategies fulfilling the requirement. So, a t-way test suite generation strategy considering both test case weight and test case priority is necessary.

DA has been used in various fields and is capable of producing competitive results. DA has been effective while obtaining global optima due to its survival of the fittest strategy [4]. Also, in the field of multi-objective problems, there have been some uses of DA [5]. Therefore, proper implementation of DA can make multi-objective or bi-objective t-way testing methods for prioritized test suite generation (where test case priorities are considered) free of many of the existing complexities.

Due to the lack of existing works involving prioritized t-way test suite generation where there are more than one objectives (test case weight and test case priority), this paper has proposed a new t-way testing strategy. The major contributions of this research can be summarized below:

- We propose a new strategy to generate prioritized t-way test suites using Bi-objective Dragonfly Algorithm (BDA), and the test suite generation is based on two objectives: test case weight and test case priority.
- We compare the experimental results of BDA with the results of other existing t-way strategies. While other t-way strategies' performances are based on test case weights only, BDA's performance is based on both test case weight and test case priority. Hence, there's a superiority in BDA's performance.

FIGURE 1. Online payment system.

The rest of the paper will discuss how BDA generates prioritized test suites. Section II will discuss an overview and theoretical background of t-way testing and test case prioritization. Section III will discuss the related existing works. Section IV will discuss the Dragonfly Algorithm. Section V will explain the methodology of constructing prioritized t-way test suites using BDA. Section VI will discuss the results obtained from the implementation of BDA in t-way testing based on both test case weight and priority. Finally, section VII will present a summary on the overall scenario of this research.

II. BACKGROUND

A. T-WAY TEST SUITE GENERATION

T-way testing is a special type of sampling technique. This method tests software or hardware systems by generating optimal test cases. The advantage of this testing method is that it's not necessary to test all combinations of inputs and outputs. Mainly, based on the interaction strength (t), the testing method can cover every t-combination of the test cases.

T-way testing generates a reduced size of test cases for testing, thus avoiding exhaustive testing. As an example, we can consider an online payment system, as shown in Figure 1. It allows the payment or transfer of money through an electronic system. There's an online payment form, which the user has to fill up with necessary information, and then this information is submitted to the website of the merchant.

In the payment option, the user needs to submit four inputs or parameters in the merchant's website. These inputs are accepted card selection, card number, expiration date, and CVV. The system supports four payment methods: PayPal, American Express, MasterCard, and VISA. 'Card Number' option supports one string value. 'Expiration Date' usually supports two input values: month and year (for example, ranging from 2016 to 2031). 'CVV' accepts an input value. There will be 720 ($4 \times 1 \times 12 \times 15 \times 1$)

test cases for exhaustively testing all the test cases for this online payment system. But exhaustively testing all the test cases for this complex system isn't practical. Therefore, t-way testing can be a great solution for this problem. There can be a reduction of 80% test cases through two-way or pairwise testing, so it can save a lot of time and effort. For two-way testing, where interaction coverage is 2, around 93% bugs or software failures can be detected [6]. Again, fault detection becomes 98% for three-way testing [7] and 100% for 4-way to 6-way testing [8].

B. THEORETICAL BACKGROUND OF T-WAY TESTING

For t-way testing, the test suite (T) is usually an n × m array, where there are test cases in n rows. With the combination of each input parameter, the formation of every test case takes place. The test suite contains all combinations of input parameter values. One test case can contain multiple combinations of input parameter values.

Suppose, there's a set containing N parameters. These parameters are p_1, p_2, p_3, p_4, . . . , p_n. Here, each parameter contains v_i possible values, such as v_1, v_2, v_3, v_4, . . . , v_m. For interaction strength t, the t-way test suite is an array in the form of N × n, ensuring that every column gets elements from v_i. Also, at least once, each N × t sub-array has every combination of size t.

For describing the test suite of t-way, a great way is to use covering array (CA), which is a mathematical notation. The uniform covering array is represented by the notation CA (N, t, v^p). Here, p is the number of parameters, v denotes the parameters' values, and t is the interaction strength level. For instance, CA(12; 2, 3^{16}) has 12 rows of test cases, and these are constructed from 16 columns of parameters with 3 values for every parameter. In case it's not a uniform covering array, and the parameters' values aren't the same, we can represent it by MCA(N, t, $v_1{}^{p_1}$ $v_2{}^{p_2}$. . . . . $v_k{}^{p_k}$). As an instance, MCA (14, 3, 2^4, 3^1) has 14 final test cases, 4 parameters with 2 values and 1 parameter with 3 values.

C. PRIORITIZATION OF TEST CASES

In prioritized t-way test suite generation, it's necessary to determine the priority score of the test cases. The priority score of a test case depends on whether a test case contains at least one prioritized element or not. However, the conditions of a t-way test suite generation process decide which elements will be prioritized and which elements will be non-prioritized.

Suppose, in a t-way test suite generation, a generated test suite have 4 test cases: (a_1,b_1,c_1), (a_2,b_2,c_2), (a_3,b_3,c_3), (a_4,b_4,c_4). Here, only b_1 and c_2 are prioritized elements. Other elements are non-prioritized.

In this case, (a_1,b_1,c_1) and (a_2,b_2,c_2) are prioritized test cases because (a_1,b_1,c_1) contains the prioritized element b_1, and (a_2,b_2,c_2) contains the prioritized element c_2. So, each of these test cases will have a higher priority score.

However, (a_3,b_3,c_3) and (a_4,b_4,c_4) are non-prioritized test cases because each of these test cases doesn't contain any of the prioritized elements b_1 or c_2. So, each of these test cases will have a normal priority score.

III. RELATED WORK

T-way testing has 2 approaches; one is the algebraic approach, and the other is the computational approach [9]. The algebraic approach has some limitations. Usually, it's for small configurations. However, the computational approach uses greedy algorithms to generate test suites. As a result, this approach can cover a great number of interaction combinations.

In combinatorial testing, there has been adaptation and implementation of DA. Implementations of DA in the combinatorial testing has shown proofs for the efficiency of DA in test suite generation mainly in terms of test suite size [10].

Hybrid FPA has also been very effective in t-way testing. FPA's hybrid variants have been successful in overcoming the complexities due to slow convergence. In terms of test suite size, the hybrid FPA variants have shown superior performance against many existing t-way strategies [11]. However, an Elitist FPA (eFPA) was implemented in test suite generation for both sequence-less and sequence t-way testing. eFPA performed competitively against existing t-way strategies in terms of test suite size [12].

T-way strategies based on the Whale Optimization Algorithm (WOA) has also been effective in t-way testing. WOA is a modern AI-based and nature-inspired algorithm [13]. A WOA based strategy with the support of constraint has performed well in t-way test suite generation [14]. WOA is also good for t-way testing with higher interaction strength. Variants of WOA has successfully avoided local optima and conquered premature convergence to ensure good performance in t-way testing with higher interaction strength [15].

However, GA, an evolutionary search-based algorithm, has been implemented successfully in t-way testing. A t-way strategy based on a modified GA has shown high efficiency in variable and uniform t-way test suite generation. By the bit structure modification and quick access to test cases, there has been an upgrade in GA's performance [16]. The strategy has competed well against existing computational and AI-based t-way strategies. Also, the strategy has performed well in t-way with higher interaction strengths.

In pairwise testing, Harmony Search (HS) Algorithm has also been effective. HS gets inspired from harmony improvisation [17]. There have been developments of HS based strategies for generating pairwise test cases. The HS based t-way strategy also shows competitive results against existing pairwise test case generation tools [18]. HS has potential for further improvement in t-way testing.

ACA mimics an ant colony moving from its nest to discover food source using the shortest possible path [19]. ACA based t-way strategies are good enough in generating test suites with smaller sizes, and ACA based strategies have dominated many existing strategies while generating smaller test suites. [20].

SCA, a recent metaheuristic algorithm, has also shown promising results in t-way test suite generation.

In combinatorial test suite generation, SCA has been successful in minimizing test suite sizes. SCA has been superior compared to many t-way strategies in generating minimized combinatorial test suites [21].

Gravitational Search Algorithm (GSA) maintains two popular Issac Newton's laws: the universal gravitation law and the motion interaction law. GSA is also great in producing optimal t-way test suites. The performance of a GSA based t-way strategy was benchmarked against the currently established t-way strategies, and the GSA based t-way strategy produced competitive results against other strategies [22]. However, another t-way strategy based on the African Buffalo Optimization (ABO) algorithm ensured greater effectiveness and faster convergence in t-way testing [23].

CS gets inspired from some cuckoos' brood parasitic behaviours. CS based t-way strategies can perform well in t-way testing by tuning parameters like the nest size, the repetition, and the elitism probability. CS based t-way strategies have been able to outperform many existing t-way strategies in t-way testing [24].

Jaya Algorithm (JA) based strategies are also effective in t-way testing. JA is a simple but robust algorithm [25]. JA based strategies have overcome the complexities due to slow convergence and performed well in t-way testing [26].

T-way testing based on the Bee Colony algorithm is also efficient in generating optimal test cases; it can even outperform many existing computational and AI-based t-way strategies in terms of constructing optimum test cases [27]. Bee Algorithm based t-way strategies are also implemented in sequence-based testing [28]. However, Hybrid Artificial Bee Colony (HABC) Algorithm has been effective for a hamming based t-way strategy targeting variable strength test set generation [29]. This strategy has performed very well compared to its counterparts.

PSO has also been proved to be effective in combinatorial testing. Quantum Particle Swarm Optimization (QPSO) strategy has successfully generated constrained combinatorial test suites [30]. A synergic QPSO technique called QPIO enriches QPSO's application in the context of combinatorial testing.

IV. DRAGONFLY ALGORITHM

Dragonfly algorithm (DA) is a population-based approach. It has a few parameters to be tuned, which makes the implementation simple. Also, reasonable convergence time is ensured during its implementation. Regarding merging options, DA is very effective. It's firmer, and it's easy to merge this algorithm with other algorithms [31].

The DA operates being inspired by the hunting behavior, which is known as a static swarm (feeding). This approach follows the migration techniques of idealized dragonflies [32]. Dragonflies are small insects; they usually move in small groups. They search for food while roaming in small groups. This process of searching for food is known as the hunting mechanism of dragonflies. There are also larger groups of dragonflies. These dragonflies move in one direction while flying with each other, so the swarm migrates following the migration mechanism. The hunting behavior and feeding behavior of dragonflies can be of various types. The swarm can be a static swarm or a dynamic swarm. There are five operators, which characterize the swarming behavior of dragonflies. Also, Figure 2 shows the swarming behaviors of dragonflies.

A. SEPARATION

In the separation mechanism, it's ensured that the search agents are away from each other when they are in the neighborhood. The separation behavior has mathematical modelling as shown in equation 1.

$$S_{\mathrm{i}} = -\sum_{j=1}^{N} X - X_{\mathrm{j}} \tag{1}$$

Here, X is the current individual's position, X_{j} is the j^{th} neighbouring individual's or dragonfly's position. However, N indicates how many individual neighbours are present in the dragonfly swarm. S is the i^{th} individual's separation motion.

B. ALIGNMENT

Alignment is the indicator regarding how a specific search agent's velocity becomes similar with other search agents' velocity in the neighborhood. The alignment behavior has its mathematical modelling, as shown in equation 2.

$$A_{\mathrm{i}} = \frac{\sum_{j=1}^{N} V_{\mathrm{j}}}{N} \tag{2}$$

Here, in equation 2, $\mathrm{V_j}$ denotes j^{th} neighbour's speed.

C. COHESION

Cohesion explains how individuals fly to the center of the mass from the neighborhood area. It means individuals' tendency to fly towards the neighborhood mass's center. The cohesion behavior has its mathematical modelling as shown in equation 3.

$$C_{\mathrm{i}} = \frac{\sum_{j=1}^{N} X_{\mathrm{j}}}{N} - X \tag{3}$$

D. ATTRACTION

Attraction explains how individuals flying towards the food source get attracted to that food. Its mathematical modelling is shown in equation 4.

$$F_{\mathrm{i}} = F_{\mathrm{loc}} - X \tag{4}$$

Here, $\mathrm{F_i}$ indicates food attraction for i^{th} dragonfly. $\mathrm{F_{loc}}$ denotes the food source's position.

E. DISTRACTION

Distraction explains individuals' tendency to stay away from enemies. The enemy and the i^{th} solution will have a distraction, and it's shown mathematically in equation 5.

$$E_{\mathrm{i}} = E_{\mathrm{loc}} + X \tag{5}$$

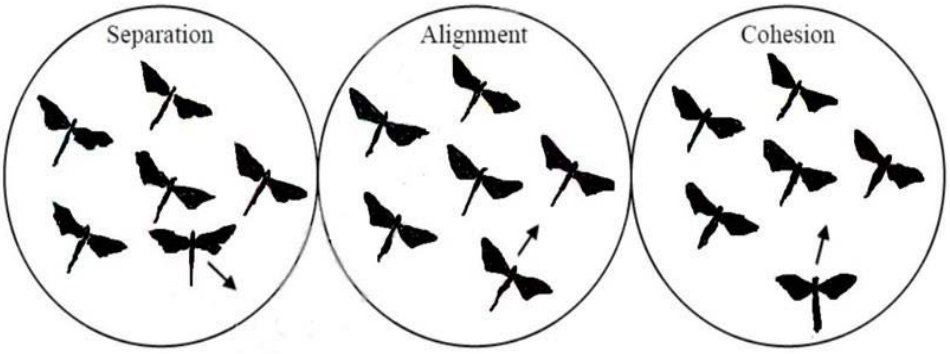

FIGURE 2. Swarming behaviors of dragonflies.

In equation 5, E_i means the distraction motion of the worst candidate or enemy for i^{th} dragonfly. E_{loc} means the position of the enemy.

F. FINDING OPTIMAL SOLUTION USING THE OPERATORS

When DA operates the search process, it uses the candidate with the strongest fitness to update the fitness of the location and the food source. Also, the worst candidate updates the enemy's location and fitness. This mechanism helps to diverge into favorable search spaces and stay away from unfavorable search spaces.

DA uses PSO's generic framework because it updates a dragonfly's position by utilizing two vectors. These two vectors are the position vector and the step vector ($\triangle$X). The step vector works similarly to the velocity vector of the PSO. The function of the step vector is to serve for altering the movement of dragonflies. The step vector is mathematically modelled in equation 6.

$$\triangle X_{t+1} = w\triangle X_t + (sS_i + aA_i + cC_i + fF_i + eE_i) \quad (6)$$

Here in equation 6, s denotes the weights of the separation (S_i) of the i^{th} individual, a denotes the weights of the alignment (A_i) of the i^{th} individual, c denotes the weights of the cohesion (C_i) of the i^{th} individual, f denotes the weights of the movement speed towards the food source (F_i) of the i^{th} individual, and e denotes the weights of the enemy disturbance level (E_i) of the i^{th} individual. The values of s, a, c, f, e, and w have been got from the existing works of DA [5], and we have also modified some of these parameters in order to improve DA's performance. In this research work, s = 0.1, a = 0.1, c = 0.7, f ranges from -1 to 1, e ranges from -1 to 1, and w ranges from -0.2 to 0.9.

We can update an individual's position according to equation 7. It also indicates the position vector calculation.

$$X_{t+1} = \triangle X_{t+1} + X_t \quad (7)$$

Here, t represents the present step.

We can enhance dragonflies' random behaviors using the levy flight in the search area [33]. For that, we modify equation 7, and get equation 8.

$$X_{t+1} = X_t + X_t \times Levy(d) \quad (8)$$

Here, d denotes dimension search space's dimension, and t represents the current iteration. We can explain levy flight by equation 9:

$$Levy(d) = \frac{r_1 \times \sigma}{|r_2|^{\frac{1}{\beta}}} \times 0.01 \quad (9)$$

Here, r_1 and r_2 represent the random numbers, and usually $0 \leq r_1 \leq 1$ and $0 \leq r_2 \leq 1$. For the adaptation purpose, for r_1, we have used the range $-1 \leq r_1 \leq 1$. β is a constant, and its value is 1.5. However, the explanation of σ is possible by the equation 10:

$$\sigma = \left(\frac{\Gamma(\beta+1) \times sin(\frac{\pi\beta}{2})}{\Gamma(\frac{\beta+1}{2}) \times \beta \times 2^{\frac{\beta-1}{2}}}\right)^{\frac{1}{\beta}} \quad (10)$$

Using these ideas and equations, we can run the iterations for t-way testing.

Algorithm 1 gives DA's pseudo-code. When DA starts its operation, there's a random initialization of a population of solutions from a given problem of optimization. The variables' upper and lower bounds define the random values which initialize the dragonflies' step and position vectors. Each iteration updates the step vector and position vector. For updating vectors X and $\triangle$X, the process chooses each dragonfly's neighborhood by calculating all dragonflies' Euclidean distance and selecting N of these. Unless the criterion is satisfied, DA keeps updating the position iteratively. DA's convergence rate is controllable by tuning the swarming weights [34].

V. THE DESIGN OF BI-OBJECTIVE DRAGONFLY ALGORITHM BASED T-WAY STRATEGY FOR PRIORITIZED TEST SUITE CONSTRUCTION

The developed and proposed t-way testing strategy generates the final test suite in several steps. It generates the search space with all available t-way combinations, updates the smallest number of test cases by covering all the t-way combinations while evaluating the test cases using

Algorithm 1 Dragonfly Algorithm's Pseudo-Code

```
Randomly generate population of dragonfly Xi (i=1,2,3,..., n)
Initializing step vectors ΔXi (i=1,2,3,..., n)
while the end condition isn't met, do
    Determine the dragonflies' objective values
    Update enemy and food source
    Update swarming weights (s, a, c, f, e, w)
    Determine S, A, C, F, E
    Update radius at neighborhood
    if a dragonfly has at least one dragonfly at its neighborhood then
        Update position vector and velocity vector
    else
        Update position vector
    end if
    Check the new positions and correct these based on the variables' boundaries
end while
```

the DA and Pareto optimality concept, and updates the final test suite. Figure 3 displays the process flow of the proposed testing strategy.

A. USER INPUT TO THE SYSTEM AND GENERATING THE LIST OF INTERACTION ELEMENTS

Firstly, the proposed strategy obtains the user inputs for the testing. The input values are analyzed. For input values, there are a set of parameters, and each parameter has one or more values. Figure 4 shows the developed t-way testing tool where inputs are provided by the user.

In this tool, the option ''Input'' asks the input parameters for t-way test suite generation. The next option ''Select prioritized elements'' is for mentioning the prioritized elements. The user has to mention the prioritized elements before running the test. When the test is running, the tool will consider higher priority scores for test cases having prioritized elements. According to this strategy's priority rule, interaction elements with one or more prioritized elements have higher priority scores. A test case's priority score is the summation of its all interaction elements' priority scores. In the tool, there are other options like ''Interaction strength'', ''Population size'', ''No. of updates of population in each iteration'', etc.

Through t-way testing with a specific interaction strength (t), the interaction elements are generated, which represent the search space. The t-way parameter combinations depend on the number of parameters, the interaction strength, and the number of values of each parameter. Every possible combination of inputs is taken to construct the complete search space of interaction elements.

Suppose, for the t-way testing's search space, there are combinations of p input parameters. In the search space, each element is an integer value's factor. A t-way combination can be $a_1, a_2 \ldots, a_t$, where t denotes interaction strength. From t-combinations of the input parameters, test cases can be generated.

Figure 5 shows how t-way interaction elements can be generated. Suppose there are 4 inputs (such as, A, B, C, D). If interaction coverage or strength (t) is 3, the 3-way combinations for the four inputs will be ABC, ABD, ACD, BDC. If interaction strength (t) is 2, then the 2-way combinations for the four inputs will be AB, AC, AD, CD, BC, BD. Then, based on the number of values of each parameter, all t-way combinations can be obtained.

B. GENERATE TEST POPULATION

As the strategy has the interaction elements now, the next step is the generation of the test population. Initially, a population is generated based on random selection from the input. The size of the population of test cases may vary depending on the input size and required optimality of the final test suite. In every iteration, the strategy selects one optimal test case. Every iteration goes through a fixed number of updates of the population. As an example, if the population size of the test is 20 and the number of updates of the population in each iteration is 100, then in every iteration, the strategy will update the population of 20 test cases 100 times based on BDA, and in each update, the strategy randomly selects one test case from the population. Thus, a list of 100 test cases is generated at the end of each iteration, and the strategy selects the best test case from that list. This list of 100 test cases generated in an iteration is deleted after the end of that iteration.

Usually, the test cases are an array of elements. These array elements come from the interaction elements that are parameter values; each parameter has one or more values within a specific range. As an example, there can be four parameters, where each parameter has i number of values. The parameters can be a, b, c and d. Here, the values of parameter 'a' are: $a_1, a_2, \ldots, a_i$. In the same way, for 'b': $b_1, b_2, \ldots\ldots, b_i$; for 'c': $c_1, c_2, \ldots\ldots.., c_i$; and for 'd': $d_1, d_2, \ldots\ldots.., d_i$. The parameters can have different numbers of values. From these parameter values, various test cases can be generated. As an example, one of the test cases is (a_1, b_1, c_1, d_1). Each population has such test cases.

C. PERFORM SEARCH USING BDA AND PARETO OPTIMALITY CONCEPT

The strategy targets to generate the smallest number of test cases covering all t-way combinations at least once in the search space. The test case covering the highest number of uncovered t-way combinations has the highest weight. Suppose, if a system has a specific number of inputs, all the t-way combinations of the inputs will be taken. Then while generating test cases, each test case will cover a specific number of uncovered t-combinations. The number of covered t-combination elements represents the weight and priority of the test case.

In equation 11, f(x) is the objective function, n is the number of combinations, t is the interaction strength.

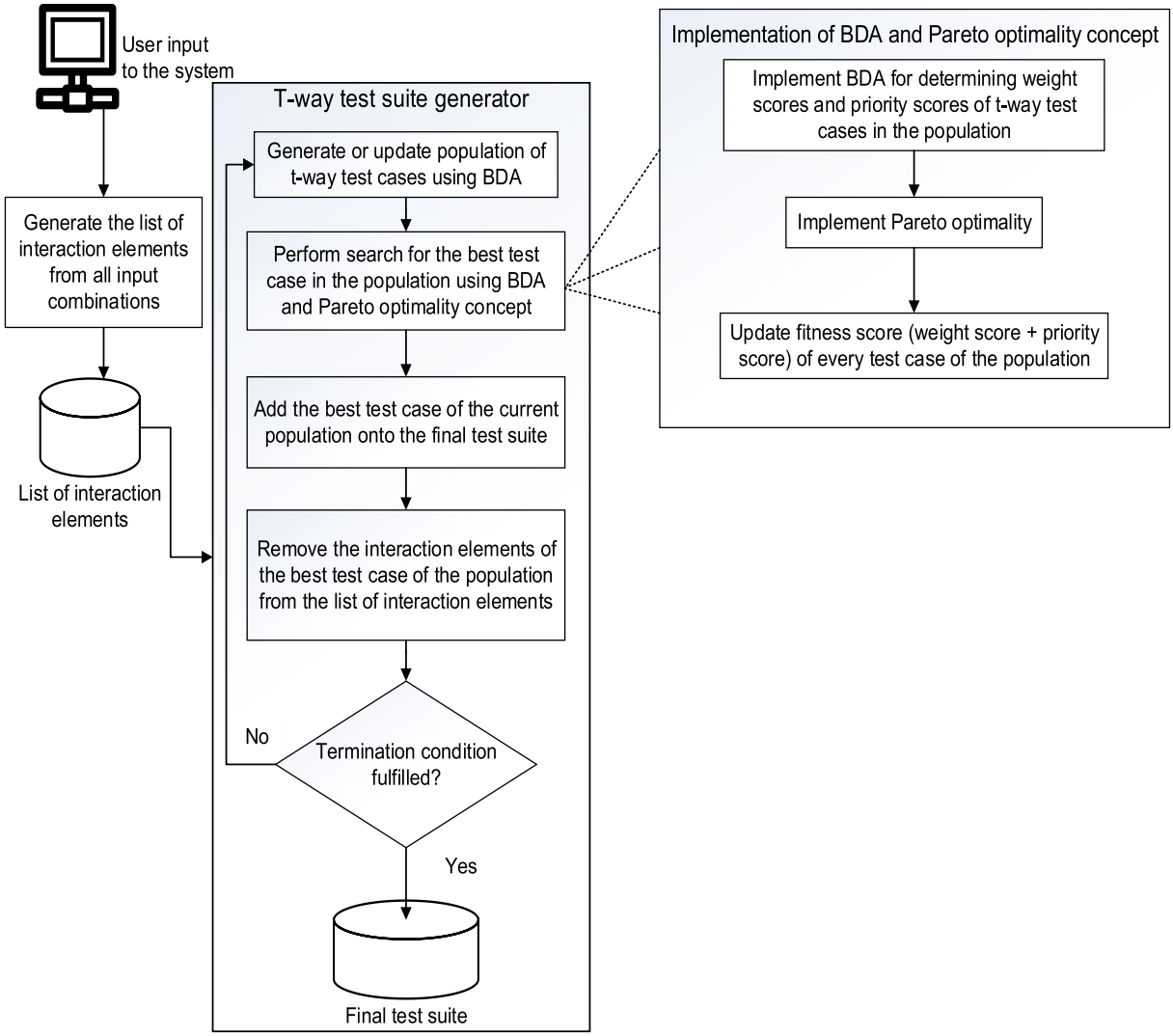

FIGURE 3. The process flow of prioritized t-way test suite construction utilizing BDA.

This function is optimized so that the test case's weight can be captured in terms of the number of covered interactions.

$$f(x) = \sum_{i=0}^{t} \sum_{j=0}^{n} (x_{i,j}, \ldots, x_{t,n}) \tag{11}$$

1) IMPLEMENT BDA

BDA helps to determine the test case's weight and priority scores. Each interaction element's weight is either 0 or 1. Suppose, if a previously uncovered interaction element is present in a test case, it adds a weight score of 1. But a covered interaction element adds a weight score of 0 to a test case.

In the case of priority, the score of an interaction element can be anything. Depending on priority conditions, the range of priority scores may vary. In our strategy, each prioritized interaction element has a priority score of 2, and for the rest of the interaction elements, each has a priority score of 1. For the priority score of a specific test case, the testing strategy adds the priority scores of all interaction elements of the test case.

2) IMPLEMENT PARETO OPTIMALITY

The strategy evaluates the test cases based on both test case weight and priority. The test cases are evaluated according to their fitness scores based on both weight and priority. For the implementation of this strategy, we combine BDA and Pareto optimality concept.

Comparison of two test cases in a bi-objective search space is possible using the Pareto dominance concept. According to Pareto dominance, if there are two vectors such as $\overrightarrow{x} = (x_1, x_2, x_3, \ldots.., x_k)$, $\overrightarrow{y} = (y_1, y_2, y_3, \ldots.., y_k)$, and $\overrightarrow{x}$ will dominate $\overrightarrow{y}$ if:

$$\forall i \in \{1, 2, \ldots, k\}, [f(x_i) \geq f(y_i)] \wedge [\exists i \in 1, 2, 3, \ldots, k : f(x_i)] \tag{12}$$

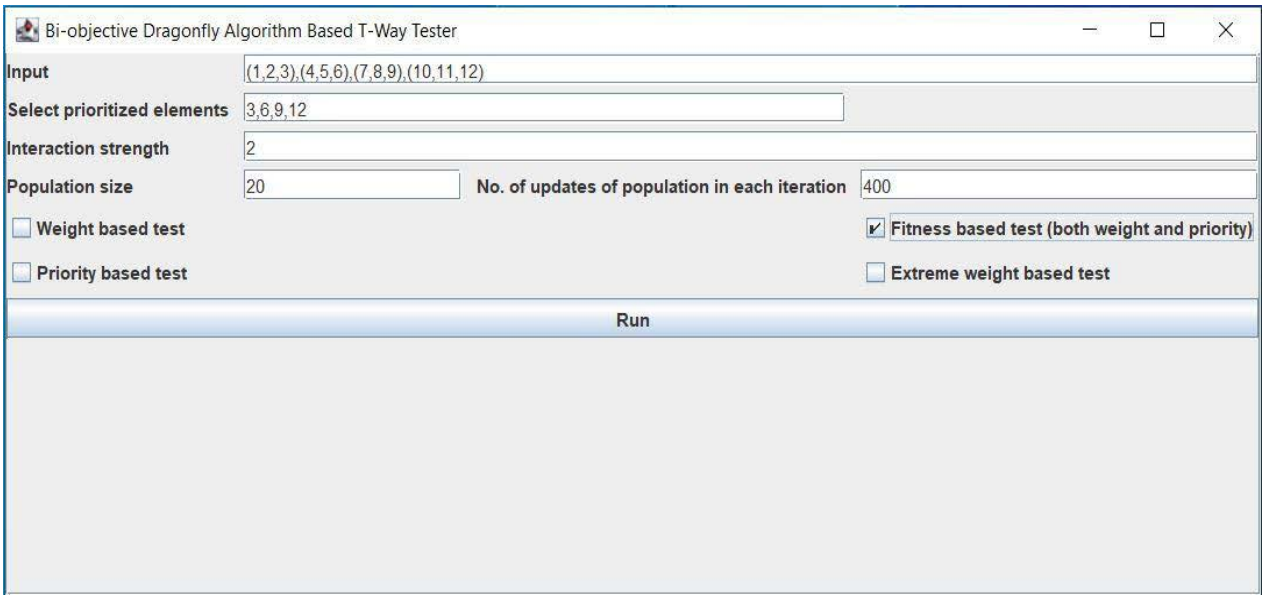

FIGURE 4. Provided inputs in the BDA based t-way testing tool.

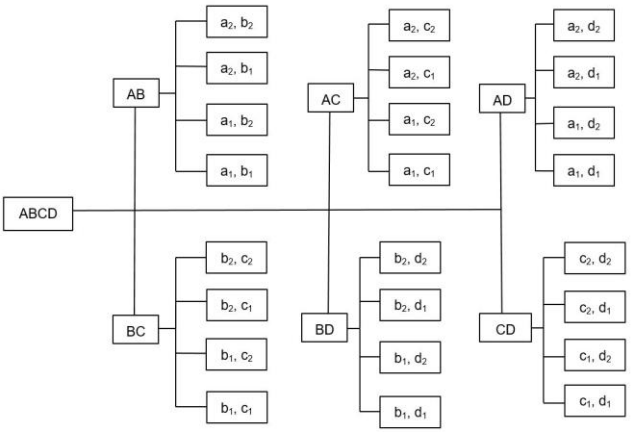

FIGURE 5. 2-way combinations for four inputs.

From equation 12, it can be said that test case 'a' can dominate test case 'b' if 'a' dominates 'b' in more number of objectives than the number of objectives in which 'b' dominates 'a'. The strategy partially use the Pareto optimality concept for test case selection in the bi-objective search space.

3) UPDATE FITNESS SCORES OF TEST CASES OF THE POPULATION AND ADD THE BEST TEST CASE ONTO THE FINAL TEST SUITE

The strategy updates the fitness score of each test case in the population by adding the weight score and priority score of the test case according to BDA.

In every population, we express the weight score of every single test case as the percentage of the highest weight score in the population. We do the same with priority scores also. Then we add the weight score and priority score of every test case, so we get the fitness score (priority score + weight score) of every test case.

After implementing BDA, the fitness scores of the population are available. After selecting the best test case from the population by partially using the Pareto optimality concept and using the fitness scores, the strategy removes the interaction elements of the best test case from all interaction elements determined from the inputs.

When the iteration ends, the strategy transfers the most superior test case onto the final test suite. Here, in every iteration, the test cases with lower fitness scores get eliminated, and the test case with the best fitness score gets included in the final test suite. BDA, along with the Pareto optimality concept, evaluates the test cases based on weight and priority scores. BDA uses its operators characterized by the swarming behaviors of dragonflies. The complete step finds the test cases with the most optimal weight and priority.

D. TERMINATION OF THE TEST AND UPDATING THE FINAL TEST SUITE

In an iteration, after finding the most optimal test case, that test case is transferred to the final test suite. Here, the final test suite gets updated based on the weight and priority of the test cases. While updating the test suite, the iteration keeps improving until fulfilling the termination condition. The test suite can be improved using BDA when selected test case variables don't provide satisfactory results to the benchmark functions. The termination condition is fulfilled when the strategy covers all the interaction elements.

In every iteration, based on fitness scores, the lower scoring test cases get eliminated, and the higher scoring test case gets included in the final test suite. Here, the t-way strategy with BDA combined with the Pareto optimality concept

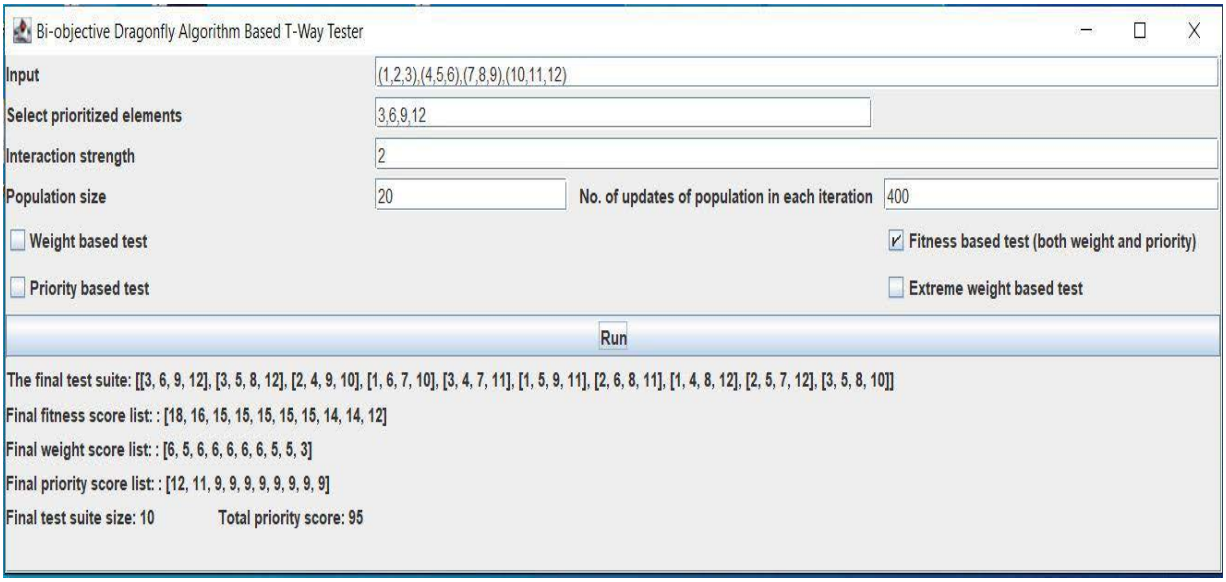

FIGURE 6. Achieved test suite in the BDA based t-way testing tool.

does the final test suite updating process. The final test suite is obtained after fulfilling the termination condition. Figure 6 shows the developed t-way testing tool where we can see the generated test suite.

VI. RESULTS AND DISCUSSIONS

After developing the design and implementation of prioritized t-way test suite construction using BDA, this section will discuss the results achieved from the implementation of BDA for various configurations. Also, there will be comparisons of BDA with other existing strategies.

A. EXPERIMENTAL SETUP

The coding of the tests have been done on Intellij IDEA Community Edition 2020.2.3 using Java. We have used Intel(R) Core(TM) i5-8250U (1.60 GHz, 3.4 GHz) with 8 GB of DDR4 RAM on the Windows 10 operating system. Many comparisons in the experiments used both average values and the best values of the test results. For determining average values, each tests were run 10 times.

The population size was 20 in all tests. For average results, the number of updates of the population in each iteration was 1000 in most tests. There were some configurations with larger inputs where some tests were run with lower numbers of updates (equal to 500) of populations in each iteration.

B. BENCHMARKING BI-OBJECTIVE DRAGONFLY ALGORITHM BASED STRATEGY WITH OTHER EXISTING STRATEGIES

From the results of our experiments, we have evaluated the performance of BDA against other existing metaheuristic algorithm based t-way testing techniques, and the evaluation has been on the basis of both test suite size and total priority score of the final test suite. However, the other existing strategies considered in this research are PSO, Harmony Search Strategy (HSS), FPA, All pairs, and Visual Pair-wise Test Array Generator (VPTag). There have been 6 sets of comparisons in our experiments. Existing test results of PSO, HSS, FPA have been collected from existing works [11], [12].

In each of the first 3 experiments (experiments 1 to 3), there have been 2 steps. Firstly, in weight based tests, there have been t-way test suite generations using BDA considering only test case weights, and these results have been compared with the results of other existing strategies. Secondly, in weight and priority based tests, there have been prioritized (considering both test case weight and test case priority) t-way test suite generations based on the developed strategy using BDA. The target is generating prioritized test suites using BDA, where test suite sizes will be competitive to test suite sizes obtained in the tests considering only test case weight. If the target is achieved, it will be a very good result because prioritization adds an extra objective (test case priority) in the t-way test suite generation, and this objective tends to increase the test suite size.

In the next 3 experiments (experiments 4 to 6), apart from BDA, there have also been implementations of the developed strategy on SCA and TLBO. Then there have been comparisons among SCA, TLBO, and BDA in terms of test suite sizes and priority scores.

1) EXPERIMENT 1: COMPARISON OF BDA WITH EXISTING STRATEGIES IN TERMS OF WEIGHT AND PRIORITY FOR 14 CONFIGURATIONS

In this experiment, there has been a comparison of BDA with other existing strategies in terms of both test suite size and prioritized test suite generation for 14 configurations. Table 1 shows the test results. There are 2 types of tests. Firstly, weight based tests consider only test case weight.

TABLE 1. Comparison of BDA with existing strategies in experiment 1.

Covering Arrays	Weight Based Tests							Weight and Priority Based Tests			
	Best						Average	Average		Best	
	Test Suite Size						Test Suite Size	Test Suite Size	Priority Score	Test Suite Size	Priority Score
	PSO	HSS	FPA	All pairs	VP Tag	BDA	BDA	BDA	BDA	BDA	BDA
CA(N; 3, 4^6)	102	70	101	NS	NS	97	101	96	3130	90	2952
CA(N; 2, 10^5)	NA	NA	125	138	136	123	125	134	2096	128	1998
CA(N; 2, 3^4)	9	9	9	10	9	9	10	10	95	9	84
CA(N; 2, 3^{13})	17	18	18	22	21	18	19	21	2578	19	2383
CA(N; 3, 3^6)	42	39	44	NS	NS	42	45	46	1577	44	1522
MCA(N;2,5^1, 3^8,2^2)	21	20	21	21	21	19	21	23	2098	21	1920
MCA(N;2,7^1, 6^1,5^1,4^6,3^8,2^3)	48	50	NA	51	NS	48	52	57	15876	53	16499
CA(N; 3, 6^6)	338	336	333	NS	NS	329	335	351	11938	338	11699
MCA(N;3,5^2, 4^2,3^2)	112	120	118	NS	NS	117	120	126	4249	118	3991
CA(N;3,5^6)	167	199	195	NS	NS	192	197	207	6352	200	6177
CA(N;2, 10^{10})	163	155	153	178	NS	190	195	201	14034	193	13496
CA(N;2, 15^{10})	NA	342	345	390	NS	445	451	479	33411	469	33800
CA(N;3, 5^7)	229	236	220	NS	NS	224	228	236	12638	229	12298
CA(N;2, 5^{10})	45	43	42	49	44	45	48	51	3369	48	3092

Secondly, weight and priority based tests consider both test case weight and test case priority.

In the weight based tests, the tests of BDA and other existing strategies are based on only test case weight. In the weight and priority based tests, the tests of BDA are based on both test case weight and test case priority.

In weight based tests, BDA has shown very good performance as compared to other strategies. BDA has outperformed PSO in 4 configurations, and PSO has been stronger in 4 configurations. BDA has outperformed HSS in 6 configurations, and HSS has outperformed BDA in 4 configurations. However, BDA outperforms FPA in 7 configurations, and FPA outperforms BDA in 4 configurations. BDA also heavily dominates All pairs and VPTag. All pairs and VPTag didn't support some configurations as these two strategies only support pairwise testing, and they don't even work for some large inputs in pairwise testing. We marked these test results by 'NS' (No support). Also, some test results were not available (marked by 'NA').

Also, in weight and priority based tests, there has been prioritized test suite generation using BDA. In weight based tests (where we consider only test suite size for evaluation), it's noticeable that BDA performs very well against the existing strategies. However, in prioritized test case generation, the test suite size usually becomes a little larger because the test considers both test case weight and priority while selecting test cases. Weight based tests consider only test case weight; that's why they can maintain a little smaller final test suite size.

From table 1, it's noticeable that the performance of BDA was satisfactory in terms of final test size in the prioritized test suite generations, where both test case weight and test case priority are considered. In the second configuration CA(N;2, 10^5), BDA produced a test suite with a size of 123 (best value) in the weight based test, and BDA also produced a test suite with a size of 128 (best value) in the weight and priority based test. So, the difference between the two test suite sizes is only 5. The results are similar for other configurations also. Apart from maintaining test suite sizes in weight and priority based tests, BDA also maintains good priority scores of the test cases, as shown in table 1. These t-way test results ensure that BDA generates prioritized test suites along with maintaining satisfactory test suite sizes. Figure 7 shows the achieved test suite sizes of BDA in weight based tests and weight and priority based tests.

From table 1, let's consider the test result (best value) of BDA in weight and priority based tests for the configuration CA(N; 3, 4^6), where the final test suite size is 90, interaction elements size (total number of interaction elements in the final test suite) is 1800, and priority score is 2952. Here, each test case has 20 interaction elements. A normal interaction element has a priority score of 1, and a prioritized interaction element has a priority score of 2. So, the minimum priority score of a test case can be 20, and the maximum priority score of a test case can be 40. Here, in the final test suite, the average priority score per test case = 2952/90 = 32.8.

2) EXPERIMENT 2: COMPARISON OF BDA WITH EXISTING STRATEGIES IN TERMS OF BOTH WEIGHT AND PRIORITY FOR CA(N; T, 2^{10}), $2 \leq t \leq 6$

The proposed strategy can function with high interaction strength, which is up to 6. Table 2 shows the performances of BDA and other strategies in terms of final test suite

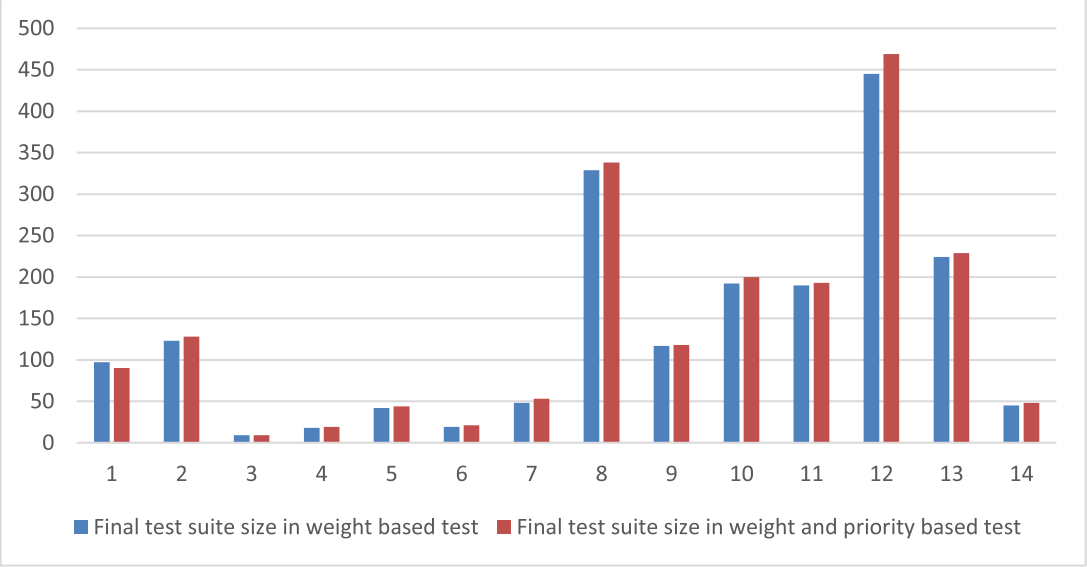

FIGURE 7. Sizes of final test suites generated by BDA in experiment 1.

TABLE 2. Evaluation of BDA with other strategies, for the configuration CA(N; t, 2^{10}), $2 \leq t \leq 6$.

t	Weight Based Tests							Weight and Priority Based Tests			
	Best Test Suite Size						Average Test Suite Size	Average		Best	
								Test Suite Size	Priority Score	Test Suite Size	Priority Score
	PSO	HSS	FPA	All pairs	VP Tag	BDA	BDA	BDA	BDA	BDA	BDA
2	8	7	8	10	9	8	9	10	805	8	643
3	17	16	16	NS	NS	16	18	20	4508	16	3644
4	37	37	35	NS	NS	37	39	44	17953	40	16365
5	82	81	81	NS	NS	80	84	87	43197	83	41262
6	158	158	158	NS	NS	158	166	164	66304	159	66365

TABLE 3. Evaluation of BDA against other strategies for the configuration CA(N; 4, 5^P), $5 \leq P \leq 10$.

P	Weight Based Tests							Weight and Priority Based Tests			
	Best Test Suite Size						Average Test Suite Size	Average		Best	
								Test Suite Size	Priority Score	Test Suite Size	Priority Score
	PSO	HSS	FPA	All pairs	VP Tag	BDA	BDA	BDA	BDA	BDA	BDA
5	779	751	784	NS	NS	780	788	789	6288	780	6248
6	1001	990	988	NS	NS	987	998	1029	25051	1017	24801
7	1209	1186	1164	NS	NS	1161	1194	1253	70129	1237	69268
8	1417	1358	1329	NS	NS	1409	1427	1459	162393	1447	161159
9	1570	1530	1478	NS	NS	1527	1571	1639	311836	1622	308716
10	1716	1624	1604	NS	NS	1691	1721	1769	586955	1746	579387

size (weight based tests) and also the performance of BDA in weight and priority based tests for the configuration CA(N; t, 2^{10}), where $2 \leq t \leq 6$.

For every strategy, the final test suite size increases with the increase of interaction strength, t. However, most of the strategies struggle while generating test suites for interaction strength $t > 6$. In weight based tests (where we consider only test suite size for evaluation), BDA, HSS, and FPA performed at a similar level. However, BDA outperformed PSO in weight based tests.

In weight and priority based tests, BDA generates test suites with sizes not much bigger than the sizes of test suites generated by BDA in weight based tests. As an example, for CA(N; t, 2^{10}) with $t = 5$, the test suite generated by BDA in the weight and priority based test has a size of 83 (best value), and the test suite generated by BDA in the weight based test has a size of 80 (best value). Here, the difference between these two test suite sizes is only 3. For $t = 2$ and $t = 3$, there's no gap between respective test suite sizes (best values) in weight based tests and prioritized (weight and priority based) tests. The results with other configurations are also impressive.

3) EXPERIMENT 3: COMPARISON OF BDA WITH EXISTING STRATEGIES IN TERMS OF BOTH WEIGHT AND PRIORITY FOR THE CONFIGURATION CA(N;4,5^P), $5 \leq P \leq 10$

Table 3 shows the performances of BDA and other present t-way techniques on the basis of test suite size (weight based tests) and also the performance of BDA in prioritized test suite generation (weight and priority based tests) for the configuration for CA(N; 4, 5^P), $5 \leq P \leq 10$.

Here, for all strategies, the final test suite size increases with the increase of P. In weight based tests, BDA has performed well against other strategies in terms of final test suite size. BDA dominated PSO and HSS in weight based tests. BDA and FPA performed at a similar level in weight based tests. However, in the weight and priority based tests, BDA has generated test suites that are not much bigger as compared to the test suites generated by BDA in the weight based tests. Also, BDA has generated highly prioritized test suites.

TABLE 4. Comparison of BDA with SCA and TLBO based on weight and priority.

Covering Arrays	Test Suite Size				Priority Score				Priority Score Per Test Case			
	Best			Average	Best			Average	Best			Average
	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA
CA(N; 3, 4^6)	103	85	90	96	3360	2769	2952	3130	32.62	32.57	32.80	32.60
CA(N; 2, 10^5)	131	130	128	134	2033	2032	1998	2096	15.52	15.63	15.61	15.64
CA(N; 2, 3^4)	9	9	9	10	84	84	84	95	9.33	9.33	9.33	9.50
CA(N; 2, 3^{13})	19	20	19	21	2384	2507	2383	2578	125.5	125.3	125.4	122.8
CA(N; 3, 3^6)	44	47	44	46	1519	1645	1522	1577	34.52	35.00	34.59	34.28
MCA(N;2,5^1, 3^8,2^2)	21	22	21	23	1894	2041	1920	2098	90.19	92.77	91.42	91.22
MCA(N;2,7^1, 6^1,5^1,4^6,3^8,2^3)	50	53	53	57	15159	16446	16499	15876	303.2	310.3	311.3	278.5
CA(N; 3, 6^6)	340	337	338	351	11705	11668	11699	11938	34.43	34.62	34.61	34.01
MCA(N;3,5^2, 4^2,3^2)	118	120	118	126	3971	4021	3991	4249	33.65	33.51	33.82	33.72
CA(N;3,5^6)	200	204	200	207	6162	6279	6177	6352	30.81	30.78	30.89	30.69
CA(N;2, 10^{10})	195	195	193	201	13735	13789	13496	14034	70.44	70.71	69.93	69.82
CA(N;2, 15^{10})	463	457	469	479	32247	31858	33800	33411	69.65	69.71	72.07	69.75
CA(N;3, 5^7)	228	232	228	236	12299	12563	12244	12638	53.94	54.15	53.70	53.55
CA(N;2, 5^{10})	48	50	48	51	3081	3282	3152	3369	64.19	65.64	66.06	65.67

4) EXPERIMENT 4: COMPARISON OF BDA WITH SCA AND TLBO IN TERMS OF BOTH WEIGHT AND PRIORITY FOR 14 CONFIGURATIONS

Table 4 displays the test results of BDA against SCA and TLBO on the basis of test case prioritization and test suite size.

Apart from BDA, there has also been the implementation of our proposed strategy of prioritized test suite generation on SCA and TLBO so that there can be an evaluation on the performance of BDA against these two metaheuristic algorithms.

In terms of test suite size (best value), BDA outperforms SCA in 4 configurations, and SCA outperforms BDA in 2 configurations. However, BDA outperforms TLBO in 9 configurations, and TLBO outperforms BDA in 3 configurations. So, considering the results in table 4, BDA can generate prioritized test suites along with maintaining smaller final test suite sizes as compared to SCA and TLBO.

BDA also performed well in terms of priority scores. Considering priority score per test case, BDA outperformed both SCA and TLBO. BDA heavily dominated SCA. Also, BDA was better than TLBO in 7 configurations, and TLBO was better in 6 configurations. Figure 8 also shows the final test suite sizes for the 14 configurations.

Considering both final test suite sizes and priority scores, BDA was ahead of both SCA and TLBO. BDA has been very effective in the prioritized test suite generation as well as maintaining smaller test suite sizes.

5) EXPERIMENT 5: COMPARISON OF BDA WITH SCA AND TLBO IN TERMS OF BOTH WEIGHT AND PRIORITY FOR CA(N; T, 2^{10}), 2≤t≤6

Table 5 shows the performance of BDA against SCA and TLBO in terms of test case weight and priority for the configuration CA(N; t, 2^{10}), $2 \leq t \leq 6$. We have applied our developed strategy on SCA and TLBO also so that there can be an evaluation on these algorithms against BDA in terms of prioritized test suite generation. In terms of final test suite size, BDA outperformed both SCA and TLBO.

BDA was also good in prioritized test suite generation because it produced test suites with a good priority score per test case as compared to SCA and TLBO. Figure 9 shows the final test suite sizes for all configurations from this experiment.

6) EXPERIMENT 6: COMPARISON OF BDA WITH SCA AND TLBO IN TERMS OF BOTH WEIGHT AND PRIORITY FOR CA(N; 4, 5^P), 5≤P≤10

Table 6 shows the performance of BDA against SCA and TLBO in terms of test case weight and priority for the configuration CA(N; 4, 5^P), $5 \leq P \leq 10$. Just like experiment 5, we have applied our developed strategy on SCA and TLBO also so that there can be an evaluation on these algorithms against BDA in terms of prioritized test suite generation.

In terms of final test suite size, BDA outperformed both SCA and TLBO. BDA was also good in prioritized test suite generation because it produced test suites with a better priority score per test case as compared to SCA. However, BDA and TLBO performed at a similar level in terms of priority score per test case in this experiment.

C. STATISTICAL ANALYSIS (FRIEDMAN AND WILCOXON SIGNED RANKED TESTS)

There have been statistical analysis and multiple comparisons of the test results. Here, the statistical analysis involves the Friedman test [35] and Wilcoxon signed-rank test. These tests analyze whether a proposed strategy has a statistical difference as compared to existing strategies. First, the Friedman

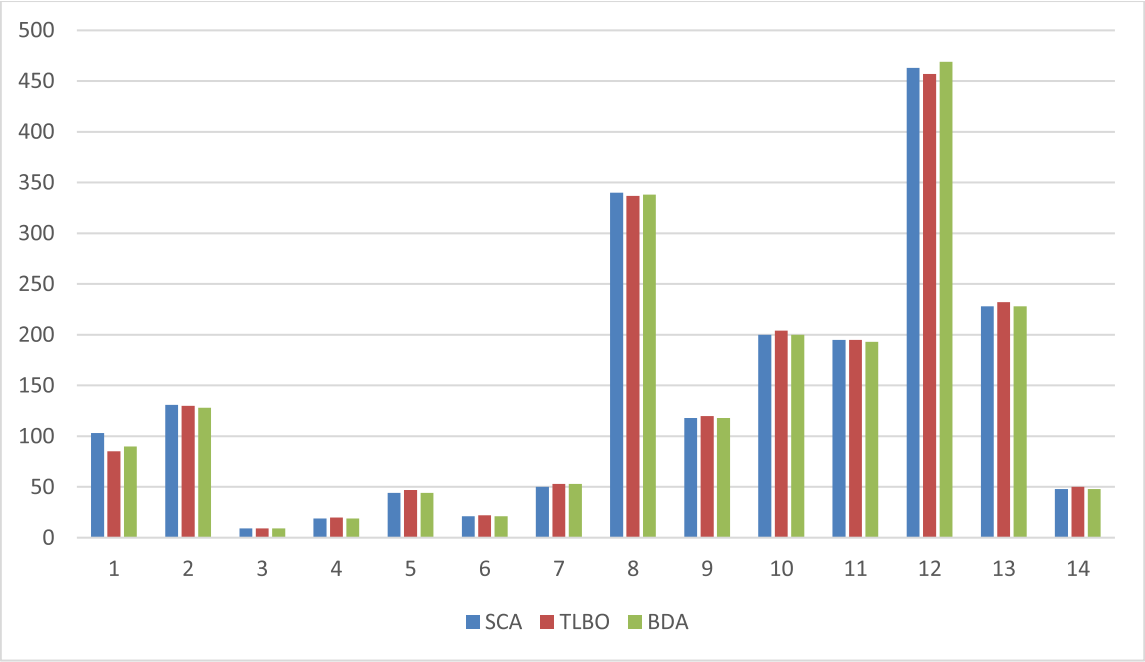

FIGURE 8. Final test suite sizes for the 14 configurations from experiment 4.

test detects differences among all strategies. Depending on the result from the Friedman test, there will be the Post-hoc Wilcoxon Rank-Sum test if required.

In the Friedman test, there can be two results. One result is rejecting the null hypothesis (H_0). Another result is retaining the null hypothesis, which is also known as the alternative hypothesis (H_1). If the Friedman test results in the rejection of the null hypothesis, there will be the Post-hoc Wilcoxon rank-sum test. Friedman and Wilcoxon signed rank tests follow some specific formulas [36], [37].

The Wilcoxon signed-rank test is a non-parametric analysis that compares two ordinal data sets subjected to different conditions. The objective is to investigate if there's a noticeable difference between the implementation results of other strategies and the developed strategy. There are two hypotheses. One is H_0, and the other is H_1.

Table 7, 8, 9, and 10 present the results of Friedman tests and Post-hoc Wilcoxon signed-rank tests. In the Post-hoc Wilcoxon signed-rank tests, there are positive ranks, negative ranks, and ties. If a p-value is less than 0.005, the compared results have no significant difference.

Friedman test works with completed samples. That's why the test ignores the NA (not available) entries in the test results. But the test considers the NS (no support) entries because it's a strategy's failure if the strategy can't support a configuration and run a test for that configuration.

There has been a set of Friedman test and Post-hoc Wilcoxon signed-rank test for experiments 1 to 3. Tables 7 and 8 show the test results. For experiments 1 to 3, we have taken BDA's test suite sizes from weight based tests only because other strategies' test suite sizes are also from weight based tests. The Friedman test rejects the null hypothesis, H_0, and proceeds to the Post-hoc Wilcoxon signed rank test. The statistical analysis shows that the performance of BDA has differences from the performances of other existing strategies. BDA has been significantly better than VPTag and All pairs, as shown in the Wilcoxon signed-rank test results in table 8. However, BDA's differences with PSO, HSS, and FPA weren't significant; the differences were small.

There has been another set of Friedman test and Post-hoc Wilcoxon signed-rank test for experiments 4 to 6. Tables 9 and 10 have the test results. Here, the Friedman test also rejects the null hypothesis, H_0, and proceeds to the Post-hoc Wilcoxon signed rank test. The statistical analysis shows that the performance of BDA has a significant difference from the performances of other existing strategies. BDA has been significantly better than SCA and TLBO, as shown in the test results in table 10.

Considering the results of both Post-hoc Wilcoxon signed-rank tests from table 8 and 10, null hypotheses were rejected in more cases as compared to the number of cases where null hypotheses were retained. These statistics prove that BDA has given a favorable result as compared to other existing strategies.

D. DISCUSSION

Developing a robust strategy for the construction of prioritized t-way test suites is quite a challenging task because most existing strategies have generated test suites based on test case weight. Test case priority has hardly been considered in the works of existing strategies. Moreover, prioritized test suite generation means the strategy has to consider both

TABLE 5. Comparison of BDA with other algorithms based on both test case weight and priority for CA(N; t, 2^{10}), 2 ≤ t ≤ 6.

t	Test Suite Size				Priority Score				Priority Score Per Test Case			
	Best			Average	Best			Average	Best			Average
	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA
2	8	8	8	10	643	621	643	805	80.38	77.73	80.38	80.5
3	17	19	16	18	3880	4253	3644	4049	228.2	223.8	227.8	225.0
4	43	45	40	44	17588	18458	16365	17953	409.0	410.2	409.1	408.0
5	82	83	82	87	40753	41365	40765	43197	497.0	498.4	497.1	496.5
6	160	159	158	164	66769	66377	65948	66304	417.3	417.4	417.5	404.3

TABLE 6. Comparison of BDA with other algorithms based on both test case weight and priority for CA(N; 4, 5^P), 5 ≤ P ≤ 10.

P	Test Suite Size				Priority Score				Priority Score Per Test Case			
	Best			Average	Best			Average	Best			Average
	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA	SCA	TLBO	BDA	BDA
5	783	787	780	789	6273	6232	6248	6288	8.01	7.92	8.01	7.97
6	1044	1037	1017	1029	24941	25324	24801	25051	23.89	24.42	24.39	24.34
7	1273	1239	1237	1253	69698	68219	69268	70129	54.83	55.06	56.00	55.97
8	1446	1469	1446	1459	1.59E5	1.65E5	1.61E5	1.62E5	110.2	112.5	111.4	111.3
9	1648	1598	1622	1639	3.15E5	3.01E5	3.09E5	3.11E5	191.7	188.2	190.3	189.7
10	1789	1753	1746	1769	5.96E5	5.84E5	5.79E5	5.86E5	333.1	332.9	331.8	331.5

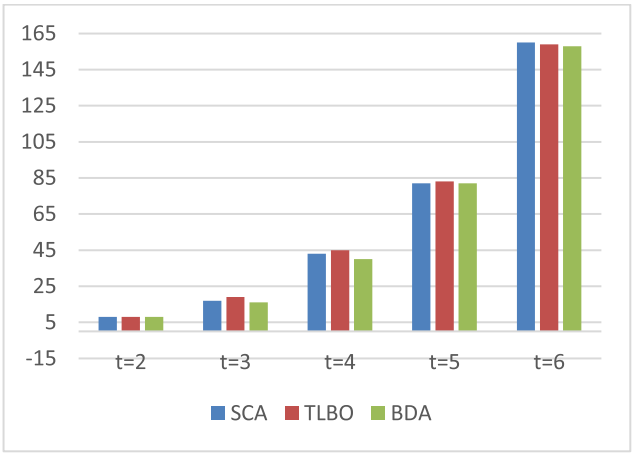

FIGURE 9. Test suite sizes for CA(N; t, 2^{10}), 2≤t≤6 in experiment 5.

TABLE 7. Friedman test for experiment 1 to 3.

Test Statistics		Conclusion
Number of samples	25	Reject H_0 and proceed to post-hoc test to see where the differences
Degree of freedom (df)	5	
Chi-Square	74.518	
Assymp. Sig.	<0.001	

TABLE 8. Wilcoxon signed ranked test for experiment 1 to 3.

Pairs	Neg. Ranks	Pos. Ranks	Ties	Total Ranks	Assymp. Syg. (2-tailed)	Bonferroni Holm Correction	Conclusion
VPTag-BDA	1	23	1	25	<0.001	0.01	Reject H_0
All pairs-BDA	2	23	0	25	<0.001	0.0125	Reject H_0
PSO-BDA	5	11	7	23	0.114	0.0167	Retain H_0
HSS-BDA	9	10	5	24	0.468	0.025	Retain H_0
FPA-BDA	8	11	5	24	0.493	0.05	Retain H_0

test case weight and priority during test case generation. It's because, only test case priority based test suite generation can't maintain smaller test suite sizes, and only test suite weight based test suite generation can't maintain highly prioritized test suites. So, a strategy is necessary which is effective in handling bi-objectives or multi-objectives. That's why this research work has selected BDA.

The selection of BDA for this research is worth discussing here. BDA has performed well in combinatorial test suite generation. It's very easy and simple to implement. There are only a few parameters for tuning, which is a reason for its simplicity in the implementation. Also, BDA's convergence time is reasonable during its implementation. Considering multi-objective optimization, BDA has shown good performance in the existing works. All these characteristics of BDA have indicated that BDA can be an efficient metaheuristic algorithm for prioritized t-way test suite construction. All these factors have been a reason for the development of BDA in this research.

Considering the overall performance of BDA against existing strategies in this research, BDA has performed very well as compared to existing strategies based on both test case weight and test case priority. In prioritized test suite generation, BDA has clearly dominated SCA and TLBO. In weight based tests, BDA has clearly dominated All pairs and VPTag. Also, BDA has been competitive against PSO, HSS, FPA in weight based tests. BDA's performance was a little better than PSO in weight based tests, but the domination of BDA wasn't much. BDA performed well against HSS and FPA, but BDA's performance didn't have a significant difference with the performances of HSS and FPA in weight based tests. So, the overall test results indicate that BDA has performed very well in prioritized test suite generation, and its performance has also been good in terms of test suite size in weight based tests.

TABLE 9. Friedman test for experiment 4 to 6.

Test Statistics		Conclusion
Number of samples	25	Reject H_0 and proceed to post-hoc test to see where the differences
Degree of freedom (df)	2	
Chi-Square	11.556	
Assymp. Sig.	0.003	

TABLE 10. Wilcoxon signed ranked test for experiment 4 to 6.

Pairs	Neg. Ranks	Pos. Ranks	Ties	Total Ranks	Assymp. Syg. (2-tailed)	Bonferroni Holm Correction	Conclusion
SCA-BDA	2	12	11	25	0.018	0.025	Reject H_0
TLBO-BDA	4	18	3	25	0.029	0.05	Reject H_0

Though the research has achieved its objectives, there are potentials to improve the work. There can be several future works, such as improving the performance the BDA, improving variable strength interaction, adding sequence based testing, considering constraints for BDA, etc.

VII. CONCLUSION

In this paper, we have proposed and implemented a t-way strategy based on BDA to generate prioritized test suites. The major objectives of our research work have been to design and implement the t-way strategy and compare it with the existing t-way strategies in terms of test suite size and test suite prioritization. Considering the test results, the research has achieved its objectives because BDA has performed well against existing t-way testing strategies in terms of final test suite size, and at the same time, BDA has generated prioritized test suites. So, the ability of BDA in generating prioritized test suites along with maintaining test suite sizes at accepted levels prove BDA's efficiency in handling multiple objectives. Due to handling two objectives (test case weight and test case priority) at the same time, BDA can be a great strategy in the field of software testing where more than one objectives need to be handled.

REFERENCES

[1] V. C. Prakash, S. Tatale, V. Kondhalkar, and L. Bewoor, "A critical review on automated test case generation for conducting combinatorial testing using particle swarm optimization," *Int. J. Eng. Technol.*, vol. 7, nos. 3–8, pp. 22–28, 2018.

[2] M. Z. Z. Ahmad, R. R. Othman, M. S. A. R. Ali, and N. Ramli, "A self-adapting ant colony optimization algorithm using fuzzy logic (ACOF) for combinatorial test suite generation," *Mater. Sci. Eng.* vol. 767, Dec. 2019, Art. no. 012017.

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

[4] A. I. Hammouri, M. Mafarja, M. A. Al-Betar, M. A. Awadallah, and I. Abu-Doush, "An improved dragonfly algorithm for feature selection," *Knowl.-Based Syst.*, vol. 203, Sep. 2020, Art. no. 106131.

[5] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Comput. Appl.*, vol. 27, pp. 1053–1076, May 2016.

[6] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Practical Combinatorial Testing*. Gaithersburg, MD, USA: NIST, 2010.

[7] K. Z. Bell, *Optimizing Effectiveness and Efficiency of Software Testing: A Hybrid Approach*. Raleigh, NC, USA: North Carolina State Univ., 2006.

[8] K. Z. Bell and M. A. Vouk, "On effectiveness of pairwise methodology for testing network-centric software," in *Proc. Int. Conf. Inf. Commun. Technol.*, Cairo, Egypt, 2005, pp. 221–235.

[9] 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.*, vol. 18, no. 3, pp. 125–148, Sep. 2008.

[10] B. S. Ahmed, "Generating pairwise combinatorial interaction test suites using single objective dragonfly optimisation algorithm," *J. Zankoy Sulaimani A*, vol. 19, no. 1, pp. 69–78, Oct. 2016, doi: 10.17656/jzs.10586.

[11] A. B. Nasser, K. Z. Zamli, A. A. Alsewari, and B. S. Ahmed, "Hybrid flower pollination algorithm strategies for t-way test suite generation," *PLoS ONE*, vol. 13, no. 5, May 2018, Art. no. e0195187.

[12] B. A. Nasser, K. Z. Zamli, A. R. Al-Sewari, and S. B. Ahmed, "An elitist-flower pollination based strategy for constructing sequence and sequence-less t-way test suite," *Int. J. BioInspired Comput.*, vol. 12, no. 2, pp. 115–127, 2018.

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

[14] 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.

[15] A. A. Hassan, S. Abdullah, K. Z. Zamli, and R. Razali, "Whale optimization algorithm strategies for higher interaction strength t-way testing," *Comput., Mater. Continua*, vol. 73, no. 1, pp. 2057–2077, 2022.

[16] 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, 2017.

[17] A. B. Nasser, A. A. Alsewari, A. A. Mu'azu, and K. Z. Zamli, "Comparative performance analysis of flower pollination algorithm and harmony search based strategies: A case study of applying interaction testing in the real world," in *Proc. 2nd Int. Conf. New Directions Multidisciplinary Res. Pract.*, Istanbul, Turkey, 2016, pp. 12–13.

[18] A. A. Muazu and U. D. Maiwada, "PWiseHA: Application of harmony search algorithm for test suites generation using pairwise techniques," *Int. J. Comput. Inf. Technol.*, vol. 9, no. 4, pp. 91–98, Jul. 2020.

[19] N. Ramli, R. R. Othman, and S. S. M. Fauzi, "Ant colony optimization algorithm parameter tuning for t-way IOR testing," *J. Phys. Conf.*, vol. 1019, Jun. 2018, Art. no. 012086.

[20] N. Ramli, R. R. Othman, and R. Hendradi, "A uniform strength t-way test suite generator based on ant colony optimization algorithm to produce minimum test suite size," in *Proc. AIP Conf.*, 2021, Art. no. 020021.

[21] K. Z. Zamli, F. Din, B. S. Ahmed, and M. Bures, "A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem," *PLoS ONE*, vol. 13, no. 5, May 2018, Art. no. e0195675.

[22] K. M. Htay, R. R. Othman, A. Amir, H. L. Zakaria, and N. Ramli, "A pairwise t-way test suite generation strategy using gravitational search algorithm," in *Proc. Int. Conf. Artif. Intell. Comput. Sci. Technol. (ICAICST)*, Jun. 2021, pp. 7–12.

[23] J. B. Odili, A. B. Nasser, A. Noraziah, M. H. A. Wahab, and M. Ahmed, "African buffalo optimization algorithm based t-way test suite generation strategy for electronic-payment transactions," in *Proc. Int. Conf. Emerg. Technol. Intell. Syst.*, 2021, pp. 160–174.

[24] A. B. Nasser, A. R. A. Alsewari, and K. Z. Zamli, "Tuning of cuckoo search based strategy for t-way testing," *ARPN J. Eng. Appl. Sci.*, vol. 10, no. 19, pp. 8948–8953, 2015.

[25] M. Ahmed, A. B. Nasser, K. Z. Zamli, and S. Heripracoyo, "Comparison of performances of Jaya algorithm and cuckoo search algorithm using benchmark functions," in *Proc. Int. Conf. Emerg. Technol. Intell. Syst.*, in Lecture Notes in Networks and Systems. Cham, Switzerland: Springer, 2021, pp. 114–125.

[26] 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 *Emerging Trends in Intelligent Computing and Informatics* (Advances in Intelligent Systems and Computing). Cham, Switzerland: Springer, 2020, pp. 352–361.

[27] A. K. Alazzawi, H. M. Rais, and S. Basri, "Artificial bee colony algorithm for t-way test suite generation," in *Proc. 4th Int. Conf. Comput. Inf. Sci. (ICCOINS)*, Kuala Lumpur, Malaysia, Aug. 2018, pp. 1–6.

[28] M. M. Zabil, K. Z. Zamli, and K. C. Lim, "Evaluating Bees algorithm for sequence-based t-way testing test data generation," *Indian J. Sci. Technol.*, vol. 11, no. 4, pp. 1–20, 2018.

[29] A. K. Alazzawi, H. M. Rais, S. Basri, Y. A. Alsariera, L. F. Capretz, A. A. Imam, and A. O. Balogun, "HABCSm: A Hamming based t-way strategy based on hybrid artificial bee colony for variable strength test sets generation," *Int. J. Comput. Commun. Control*, vol. 16, no. 5, pp. 1–18, Oct. 2021.

[30] X. Guo, X. Song, and J.-T. Zhou, "A synergic quantum particle swarm optimisation for constrained combinatorial test generation," *IET Software*, vol. 16, no. 3, pp. 279–300, 2022.
[31] C. M. Rahman and T. A. Rashid, "Dragonfly algorithm and its applications in applied science survey," *Comput. Intell. Neurosci.*, vol. 2019, pp. 1–21, Dec. 2019.
[32] J. Too and S. Mirjalili, "A hyper learning binary dragonfly algorithm for feature selection: A COVID-19 case study," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106553.
[33] M. Abedi and F. S. Gharehchopogh, "An improved opposition based learning firefly algorithm with dragonfly algorithm for solving continuous optimization problems," *Intell. Data Anal.*, vol. 24, no. 2, pp. 309–338, Mar. 2020.
[34] Y. Meraihi, A. Ramdane-Cherif, D. Acheli, and M. Mahseur, "Dragonfly algorithm: A comprehensive review and applications," *Neural Comput. Appl.*, vol. 32, no. 21, pp. 16625–16646, Nov. 2020.
[35] M. Friedman, "A comparison of alternative tests of significance for the problem of *m* rankings," *Ann. Math. Statist.*, vol. 11, no. 1, pp. 86–92, 1940.
[36] D. W. Zimmerman and B. D. Zumbo, "Relative power of the Wilcoxon test, the Friedman test, and repeated-measures ANOVA on ranks," *J. Exp. Educ.*, vol. 62, no. 1, pp. 75–86, Jul. 1993.
[37] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*. Hoboken, NJ, USA: Wiley, 2010.

MASHUK AHMED received the B.Sc. degree in electrical and electronic engineering from the Khulna University of Engineering and Technology (KUET), Bangladesh, in 2017. He is currently pursuing the M.Sc. degree in software engineering with Universiti Malaysia Pahang, Malaysia. He is doing research on software testing. He has worked on t-way testing using metaheuristic algorithms.

ABDULLAH B. NASSER received the B.Sc. degree from Hodeidah University, Yemen, in 2006, the M.Sc. degree from Universiti Sains Malaysia, Malaysia, in 2014, and the Ph.D. degree in computer science (software engineering) from Universiti Malaysia Pahang, in 2018. He is currently a University Lecturer in programming and software engineering with the University of Vaasa, Finland. He is also the author of many scientific articles published in renowned journals and conferences. His research interests include software engineering and artificial intelligence software testing and search-based computing, specifically, the use of optimization methods for solving different problems.

KAMAL Z. ZAMLI (Member, IEEE) received the degree in electrical engineering from 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 the University of Newcastle, Tyne, U.K., in 2003. He is currently a Professor with the Faculty of Computing, Universiti Malaysia Pahang, Malaysia. His research interests include combinatorial t-way, software testing, and search-based software engineering.