Received 30 August 2024, accepted 25 October 2024, date of publication 13 November 2024, date of current version 27 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3497321

RESEARCH ARTICLE

CS-FuzGA-PTS: Maximizing Fault Detection Through Optimizing T-Way Test Suite Prioritization Based on Boundary Value Analysis

ABDULLAH B. NASSER[1], MD. ABDUL KADER[2], (Member, IEEE), ABDULRAHMAN A. ALSEWARI[3], (Senior Member, IEEE), SULIMAN MOHAMED FATI[4], (Senior Member, IEEE), NIBRAS ABDULLAH[5], SAYAWU YAKUBU DIABA[6], (Senior Member, IEEE), AND WAHEB A. JABBAR[7], (Senior Member, IEEE)

[1]School of Technology and Innovations, University of Vaasa, 65200 Vaasa, Finland
[2]Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Pahang 26600, Malaysia
[3]College of Computing, Birmingham City University, B5 5JU Birmingham, U.K.
[4]College of Computer and Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia
[5]Faculty of Computer Studies, Arab Open University, Riyadh 84901, Saudi Arabia
[6]VTT Technical Research Center of Finland, 02150 Espoo, Finland
[7]College of Engineering, Faculty of Computing, Engineering and the Built Environment, Birmingham City University, B5 5JU Birmingham, U.K.

Corresponding author: Abdullah B. Nasser (Nasser.abdullah@uwasa.fi)

ABSTRACT In the realm of software testing, resource limitations pose a significant challenge to ensuring comprehensive testing coverage. While there are numerous attempts to systematically generate test cases that maximize input coverage and fault detection, there remains an essential need for prioritizing test cases to ensure efficient utilization of resources. Given the important role of each individual test case in the overall testing process, a Prioritized Test Suite (PTS) plays a vital role in optimizing testing resources, achieving maximum fault detection, and providing comprehensive test coverage. This research addresses this need by proposing and implementing a new testing strategy called Cuckoo Search with Adaptive Fuzzy Logic-Controlled Genetic Algorithm Operators for Generating PTS (CS-FuzGA-PTS). CS-FuzGA-PTS aims to systematically generate PTS by utilizing t-way testing, boundary value analysis (BVA), and optimization techniques. CS-FuzGA-PTS employs T-way testing for test case reduction and ensures maximum input coverage. CS-FuzGA-PTS incorporates BVA to prioritize test cases based on their boundary values to identify potential defects that occur at the boundaries of input ranges, thereby optimizing the test execution efforts by focusing on high-priority cases. The core of CS-FuzGA-PTS lies in a new optimization algorithm called CS-FuzGA as a search algorithm. The algorithm integrates adaptive fuzzy logic-controlled Genetic Algorithm (GA) operators with Cuckoo Search (CS). By dynamically adjusting search behavior based on solutions diversity, CS-FuzGA enhances both exploration and exploitation, achieving an optimal balance between them through integrating GA operators into CS according to search requirements. The results obtained from the experiments provide insights into the effectiveness of CS-FuzGA-PTS in generating a PTS that can identify potential defects occurring at input boundaries. Moreover, CS-FuzGA-PTS outperforms existing strategies in terms of test reduction.

INDEX TERMS Adaptive fuzzy logic control, boundary value analysis, cuckoo search, fault detection, genetic algorithm, prioritized test cases, optimization, software testing, T-way testing.

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

I. INTRODUCTION

As advancements in technology drive rapid evolution in software development, creating innovative technologies and applications has become increasingly complex. This complexity highlights the critical importance of rigorous software testing to ensure the quality, reliability, and integrity of these applications in meeting user needs and market requirements. Numerous existing test suite generation strategies attempt to systematically ensure comprehensive test coverage that covers a wide range of software inputs, while others aim to reduce the size of the generated test suite. In general, when designing a testing tool for generating test suites, several objectives are considered. These objectives include minimizing test cases, maximizing input coverage, and maximizing fault detection, to name a few. To produce high-quality software, different testing tools and approaches have investigated these objectives.

In the literature, many techniques aim to maximize testing coverage and minimize test suite size objectives, such as state transition testing, use case testing, and combinatorial testing. These techniques ensure that the software is tested against a wide range of input combinations that maximizing fault detection while using the minimum test suite size. T-way testing, a combinatorial test design technique, systematically generates the test suite from a large input space in a way that covers all possible t-size combinations of input parameters, thus ensuring maximum coverage. To minimize the test suite size, several t-way testing strategies utilize optimization algorithms as search engines for generating optimal test suites, such as Simulated Annealing (SA) [1], Genetic Algorithm (GA) [1], [2], Ant Colony Algorithm (ACA) [2], Harmony Search (HS) [3], Particle Swarm Optimization (PSO) [4], and Cuckoo Search (CS) [5], to find the minimum test cases that cover the maximum inputs. Due to their efficiency, many researchers adopt hybridization or adaptation of meta-heuristic algorithms as search engines, such as high-level hyper-heuristic (HHH) [6], elite-FPA [7], Improved-JA [8], Learning-CS [9], Modified ABC [10], Modified FPA [11], Hybrid HS with Grey Wolf Optimizer [12], and Hybrid ABC [13].

These strategies have proven their efficiency as they take advantage of the hybridization of algorithms to produce an optimal test suite rather than relying solely on a single algorithm. CS is an optimization algorithm inspired by the brood parasitic behavior of cuckoo species. However, its search capabilities, like other optimization algorithms, are limited, and it may trap in local optima due to a lack of exploration or exploitation or a balance between the two. To address these limitations, this research proposes a new optimization algorithm called CS-FuzGA by integrating CS, fuzzy logic control, and GA operators to overcome the limitations of standard CS. Integrating GA operators, such as mutation and crossover, into CS enhances its performance by enabling a more thorough exploration of the search space and by exploiting promising solutions. On the other hand, adaptive fuzzy logic control monitors the diversity of the potential solutions and dynamically adjusts search behaviour to achieve a good balance between exploration and exploitation by integrating GA operators into CS according to search requirements. Additionally, by adjusting genetic operators based on the population diversity, the CS-FuzGA is able to mitigate the premature convergence issue, which is commonly encountered in optimization algorithms.

Beyond combinatorial testing, other software testing techniques, such as equivalence partitioning, boundary value analysis (BVA), and decision table testing, offer high potential to identify critical defects. BVA, a black box testing technique, focuses on testing the values at the boundaries of input domains, where errors are more likely to occur. BVA plays a crucial role in software quality by thoroughly examining critical points on the boundaries. Integrating t-way testing and BVA can enhance the robustness and efficiency of the testing process, leading to high software quality and reliability. To effectively leverage these insights, this research aims to design a software testing strategy for generating a Prioritized Test Suite (PTS) by integrating t-way testing and BVA techniques.

PTS is a set of test cases that are organized and designed in a practical sequence, indicating the importance of each individual test case, including test case cost, test case coverage, test case history, and customer requirements [14], [15]. In general, prioritizing test cases plays a vital role in maximizing fault detection, optimizing testing resources, and achieving comprehensive test coverage. However, an effective PTS is designed to maximize the probability of fault detection based on certain criteria [16]. Although the importance of different test case prioritization techniques for maximizing fault detection is known, prioritizing the test case based on the BVA adds another layer of effectiveness to the testing process. A PTS ordered by BVA helps to optimize the test execution efforts by executing high-priority test cases first. By prioritizing test cases based on these boundary conditions, testers can detect hidden defects that may remain undetected in other testing approaches.

Given these challenges, our research proposes a novel software testing strategy called Cuckoo Search with Adaptive Fuzzy Logic-Controlled Genetic Algorithm Operators for Generating PTS (CS-FuzGA-PTS), which aims to systematically generate PTS by utilizing t-way testing, BVA, and CS-FuzGA techniques. CS-FuzGA-PTS incorporates the BVA technique to prioritize test cases based on their boundary values to identify potential defects that occur at the boundaries of input ranges, while t-way testing is used as a test reduction technique to reduce the number of test cases and ensure maximum input coverage. CS-FuzGA-PTS utilizes the CS-FuzGA algorithm as the core search algorithm for maximizing t-way testing coverage.

Putting the pieces together, the contributions of this research can be summarized as follows:

- **Enhanced Optimization with CS-FuzGA**: This research contributes to the field of optimization by proposing a new variant of the CS algorithm that utilizes the strengths of both CS and GA operators. CS-FuzGA is controlled by fuzzy logic to be able to adjust and improve its performance, leading to better exploration of the solution space and exploiting promising regions. This combined approach provides an efficient optimization process compared to using CS alone.

- **Prioritized Test Suite Generation with CS-FuzGA-PTS**: Proposes a new software testing method for generating PTS based on CS-FuzGA called CS-FuzGA-PTS. CS-FuzGA-PTS employs t-way testing to systematically reduce the number of test cases, and BVA to prioritize test cases based on their boundary values, which helps to optimize the test execution efforts and identify potential defects occurring at input range boundaries.

The rest of the paper is organized as follows: Section II reviews the related work, including t-way testing strategies and prioritized test strategies. Section III presents the two proposed works: CS-FuzGA and CS-FuzGA-PTS, while Section IV evaluates the proposed works against the existing works and discusses the results. Lastly, Section V concludes the work and presents recommendations for future work.

II. RELATED WORKS

As far as literature is concerned, a significant amount of study has been conducted on test case generation. This section provides an overview of both early t-way testing strategies and the most recent research, especially focused on prioritized test case generation. These strategies come from a diverse range of fields, including optimization algorithms, mathematical approaches, and heuristic search techniques. Broadly speaking, the existing t-way testing strategies can be categorized into either the One Parameter at a Time (OPAT) strategy, where the test cases are generated by adding a column per iteration, or the One Test at a Time (OTAT) strategy, where the test cases are generated by adding one row per iteration until all the t combinations are covered. Examples of OPAT strategies are the in-parameter-order (IPO) strategy and its variants [17], [18]. OPAT strategies optimize the process of generating test cases for each individual parameter instead of the entire set of test cases for all parameters. These strategies are fast and efficient as they can produce the minimum number of test cases; however, they are computationally intensive for large systems.

On the other hand, Automatic Efficient Test Generator (AETG) [19], GTWay [20], Jenny [21], TConfig [22], and WHITCH [23] are examples of OTAT strategies. While these strategies are manageable and efficient, especially for systems with complex interactions between parameters, managing the complexity of these strategies is still challenging [24]. Under the OTAT, several strategies treat the problem of test case generation as an optimization problem. These strategies utilize meta- heuristic algorithms such as SA [1], GA [1], [2], ACA [2], HS [3], PSO [4] and CS [5] to find the optimal test case that covers the maximum number of t combinations. The strategies start by generating all the t combinations, and then iteratively, the meta-heuristic algorithm is used to find the optimal test case that covers the maximum number of combinations. This process is repeated until all the t combinations are covered. These strategies are efficient in terms of reducing the total size of the test suite and the time taken to generate the test cases compared to the exhaustive search. Due to its efficiency, many researchers adopt adaptive or hybridized meta-heuristics algorithms as search engines, such as high-level hyper-heuristic (HHH) [6], elite-FPA [7], Improved-JA [8], Learning-CS [9], Modified ABC [10], Self-adaptive FPA, Hybrid HS with Grey Wolf Optimizer [12], and Hybrid ABC [13]. These strategies have proven their efficiency as they take advantage of the hybridization of algorithms to produce the optimal test cases in a reasonable time.

In addition to addressing the test suite reduction problem, several studies have responded to other software testing problems, including test suite prioritization. Rechtberger et al. [25] examined and analyzed the prevalent presence of Finite State Machines (FSMs) in contemporary systems and the necessity of testing these systems through the use of trained sequences of transitions in FSMs. The study focused on generating test paths in the most efficient manner, considering many factors such as the lowest test cost or the highest likelihood of identifying flaws in the System Under Test (SUT). This study also emphasized the necessity of reaching a specific test coverage criterion and the effectiveness of the model-based testing (MBT) strategy. The specific criteria for generating test pathways from the FSM model are provided. These capabilities encompass the ability to initiate and conclude test paths in predetermined states, select the order in which states and transitions are traversed, and specify the range of path lengths. The methodology described here is the Prioritized State Machine Test (PSMT) strategy, which tries to produce sets of test pathways. PSMT uses the FSM model and two categories of test coverage requirements to represent the problem. Furthermore, the method includes a description of six algorithm alternatives, with a baseline N-switch reduction algorithm serving as a point of reference. The efficacy of the six algorithm variants is evaluated by assessing their ability to generate test pathways and activate simulated flaws as specified in the FSMs. The study evaluates the characteristics of the produced test pathways and their efficacy in various problem configurations, employing a combination of actual automotive and defense projects and artificially generated FSMs. The study demonstrates that the PSMT technique outperforms the baseline in most problem scenarios. Additionally, it discusses the distinct behaviors of the six algorithm variations, which vary based on the utilization of the FSM. The primary conclusions of the research highlight the adaptability and possible practicality of the PSMT technique in actual industrial experiments. This statement highlights the importance of simultaneously fulfilling the defined criteria and the significance of the approach and evaluation in testing both functional and non-functional software requirements.

Ahmed et al. in [26] put forth a Bi-objective Dragonfly Algorithm (BDA), as a means of generating a T-way PTS. When producing test suites, it is frequently necessary to differentiate between high-priority and low-priority test cases; therefore, the research emphasizes the significance of considering both the weighting and priority of test cases. The objective of the BDA strategy is to produce PTSs by considering two key factors: the priority and weight of the test cases. Comparing the effectiveness of BDA to that of existing T-way strategies, the research demonstrates that BDA generates PTSs effectively and is competitive in terms of test suite size. In addition to discussing the possibility of future enhancements, the research concludes that BDA is a valuable strategy in the field of software testing due to its effectiveness in managing multiple objectives. Furthermore, the paper identifies prospective future research directions the addition of sequence-based testing, the consideration of constraints for BDA, and the improvement of variable strength interaction. These factors suggest there is still room for enhancement in the algorithm capabilities and performance. In light of the research paper accomplishments, there are prospects for additional refinements pertaining to the performance, scope, and functionalities of the algorithm.

Michaels et al. in [27] presented a method for creating test cases for open-source Android applications. They developed a test generation tool using Java and compiled it in Eclipse. The research seeks to address many research inquiries, such as determining the rate of event coverage and code coverage for test suites prioritized based on element and event sequences. Additionally, it intends to discover the generation in which an element or event sequence obtains 100% code coverage initially. Test generation involves the random execution of the program to be tested using a Java-based test generating tool that is built and compiled in Eclipse. The utility communicates with an Android emulator by utilizing the Appium server to transmit events and obtain verification of event execution. The test cases are analyzed to obtain data regarding the element and event sequence criteria. Scripts are then used to build a comprehensive set of each generation covered by the entire test suite. The study assesses the order in which tests are conducted by analyzing the code coverage achieved. It also quantifies the rates at which items, events, and code are covered using different metrics. The study focuses on the prioritizing of test suites based on sequence-based coverage criteria and reports on the extent to which 100% sequence coverage is achieved by the application. The study examines the implications of the findings, including the effectiveness of prioritizing test suites with extensive sequence spaces and their potential use for bigger applications. Overall, the study offers valuable insights into the process of generating and prioritizing test cases for Android applications, specifically emphasizing event and code coverage.

Kali and Murthy in [28] introduced a hybrid Firefly Algorithm (FA) to prioritize regression test cases in software development. The methodology used the K-means clustering algorithm to distinguish pertinent test cases from irrelevant ones. It then optimizes the process using a hybrid firefly algorithm (HFFA), which combines the Artificial Bee Colony algorithm and the FA. The evaluation of the performance involves the use of several metrics, with a specific focus on the average percentage of faults detected (APFD). The results demonstrate that the suggested hybrid Firefly ABC technique surpasses the current methods in terms of accuracy and APFD values. The study demonstrates the efficacy of the hybrid FA in prioritizing test cases, and the conclusion underscores the superiority of the suggested method compared to existing techniques.

Choi et al. [29] examined the efficacy of fault detection by prioritizing combinatorial test generation. The study investigates the relationship between Kullback-Leibler (KL) divergence, weight coverage, and fault detection effectiveness, particularly when order-based and frequency-based prioritization methods are employed. The study seeks to ascertain if order-oriented prioritized combinatorial test suites with higher weight coverage yield superior fault detection effectiveness and if frequency-oriented prioritized combinatorial test suites with better KL divergence yield superior fault detection effectiveness. Moreover, the study assesses the relationship between the effectiveness of fault detection and the prioritization strategies employed.

Badanahatti and Murthy in [30] employed the test case prioritization technique to do regression testing on web applications. This paper examines many methodologies and strategies for test cases, including input-driven methods, point-of-interest (POI)-based methods, and clustering and optimization techniques. The suggested methodology comprises three primary stages: test case generation, test case clustering utilizing the Kernel Fuzzy C Means (KFCM) algorithm, and test case prioritization employing the Gray Wolf Optimization (GWO) algorithm for test case prioritization. In addition, this study employs performance and coverage metrics, such as loop coverage, statement coverage, line coverage, and comment coverage, to generate test cases. Additionally, it assesses the efficiency of the suggested approach by considering factors such as the time it takes to execute and the amount of memory it consumes. It also compares this to other strategies that are already in use. The objective is to efficiently determine the order of importance for test cases in the regression testing of online applications. An opportunity exists to enhance the method by employing diverse clustering and optimization strategies, with a specific emphasis on security in cloud systems.

Biswas et al. in [31] employed Ant Colony Optimization (ACO) to automate software testing and developed two algorithms utilizing the optimal pathways and test sequence data. The initial technique seeks to produce ordered paths in a control flow graph (CFG) by having the ants compute the likelihood of potential nodes and build paths accordingly. The likelihood is calculated by using the pheromone value and heuristic information. This method guarantees that the paths are ranked according to likelihood and allows for the early identification of faults throughout the testing procedure. The second approach employs ACO to provide test data, which is subsequently utilized as input for the created pathways. The approach is specifically designed to generate test data sequences that span the full domain and avoid incomplete searches. This comprehensive methodology ensures thorough software coverage while minimizing duplication, enhancing the quality of testing, and reducing testing costs. The talks emphasized the efficacy of the proposed approach in automating structural software testing. The prioritized paths and the provision of relevant test data aid in covering all program paths and effectively detecting faults at an early stage. The research also showcased the utilization of algorithms in evaluating a software module, achieving comprehensive coverage of all possible paths while minimizing redundancy. The study revealed that the ACO algorithm is highly effective in achieving optimum outcomes in software testing. Additionally, the integration of several meta-heuristic methodologies can enhance the performance of software testing applications.

Wang et al. in [32] devised a technique for creating test cases using prioritized pair-wise testing. Firstly, the approach entails allocating priority weights to each parameter, which represents the parameter priority information. As weight increases, priority also increases. The criteria for determining weights were established according to code coverage, test case cost, recently modified code domain, and testers' inclination. Additionally, a composite binary group was formed by utilizing weighted parameters. The weights of the test cases were determined by summing the values of the binary groups covered by them. The objective of this strategy was to enhance testing efficiency by giving precedence to high-important test cases. Furthermore, this approach generates a priority combination model known as a biased covering array. This model is used to aggregate pair-wise test cases with priority. The test cases are arranged in descending order according to their priority weights. The objective is to create a biased covering array that satisfies pair-wise criteria while maximizing the total weight of the first N test cases. Ultimately, the study included a greedy approach to the GA to enhance the efficiency of test case development. The objective of this integrated strategy was to tackle the issue of combinatorial testing with prioritization and devise an improved global search algorithm.

Qu et al. in [33] examined the efficacy of combinatorial interaction testing (CIT) in assessing software topics across various versions. The objective of the research is to assess the performance of CIT in detecting intentionally introduced defects, in comparison to a comprehensive test set. The study examines various methods for prioritizing CIT test suites and compares them to a method for re-generating and prioritizing them. The goal is to determine whether prioritized and re-generated/prioritized CIT test suites are more effective in detecting defects earlier than non-prioritized test suites. An issue with the study on Combinatorial Interaction Testing (CIT) is its failure to assess the effectiveness of CIT tests in selecting or prioritizing problems across various software versions. Furthermore, the study did not evaluate the effectiveness of CIT in regression testing when applied to successive iterations of a software program. Moreover, there is a lack of inquiry into the implementation of prioritization approaches in this particular setting. The study also lacks empirical testing on actual software topics and fails to address the crucial aspect of prioritizing weighting.

Summing up, the reviewed research papers proposed different methods for generating PTS by considering different objectives, including minimizing test cost, maximizing fault detection probability [25], coverage weight and priority of test cases [25], [32], and coverage of the sequences of events or elements of the program [27]. This section also shows that several techniques have been used for generating PTS; however, optimization algorithms such as BDA [26], HFFA [28], KFCM with GWO [30], and ACO [31], are the most common techniques used for generating PTS. The existing studies applied the generated PTS across different domains of software testing, including regression testing, web application testing, Android application testing, and general test suite prioritization. The studies highlight the significance of integrating PTS into the testing development process to enhance the efficiency of the overall testing process, leading to high-quality and reliable software. Building upon these foundational works, our research introduces a new method for generating PTS aimed at achieving key objectives, including increasing fault detection and improving testing coverage. In the next section, we introduce our proposed method, CS-FuzGA-PTS, which integrates BVA and t-way testing techniques to provide a more effective solution to achieve these objectives.

III. PROPOSED WORKS

Building upon the foundations laid by the prior research discussed in the previous section, this section presents the design of the two proposed works: CS-FuzGA-PTS as a testing strategy for generating PTS and the CS-FuzGA algorithm as an optimization algorithm. The method comprises primary components that consider the base of the method. Before we start presenting the CS-FuzGA-PTS, the following is a brief description of the CS-FuzGA-PTS components:

1. **T-way Testing Technique**: CS-FuzGA-PTS utilizes t-way testing as a sampling technique to minimize the size of the test suite by selecting representative test cases that ensure all the t-combinations of inputs are included in the test cases. The test case coverage or weight, which is considered the first objective of CS-FuzGA-PTS, is calculated based on the t-way testing technique. The test case coverage measures the number of t-combinations covered by the test case.
2. **Boundary Value Analysis Technique:** In order to identify critical defects that occur on the boundaries of input ranges, unlike other t-way test generation strategies where the test cases are generated only based on the interaction strength of the inputs, CS-FuzGA-PTS utilizes the BVA technique to prioritize the test cases based on their

boundary values. By focusing on the boundaries of inputs, prioritized test cases can enhance the efficiency of test execution efforts by executing high-priority test cases first. The test case priority, which is the second objective to be optimized during the test case generation, measures the number of boundary values covered by the test case.

3. **Cuckoo Search Algorithm:** The CS Algorithm is used as the core search algorithm for searching for the optimal test cases. generating prioritized test cases. The CS optimizes test cases based on the coverage score and boundary value score of the test cases.
4. **Adaptive Fuzzy Logic Control with GA Operators:** CS alone may get stuck in the local optima. To overcome this limitation, adaptive fuzzy logic control with GA operators is used to continuously monitor the diversity of the potential solutions and enhance the search quality. Adaptive fuzzy logic control dynamically adjusts search behavior to achieve a good balance between exploration and exploitation by integrating GA operators into CS according to search requirements.

A. THEORETICAL BACKGROUND

1) T-WAY TESTING

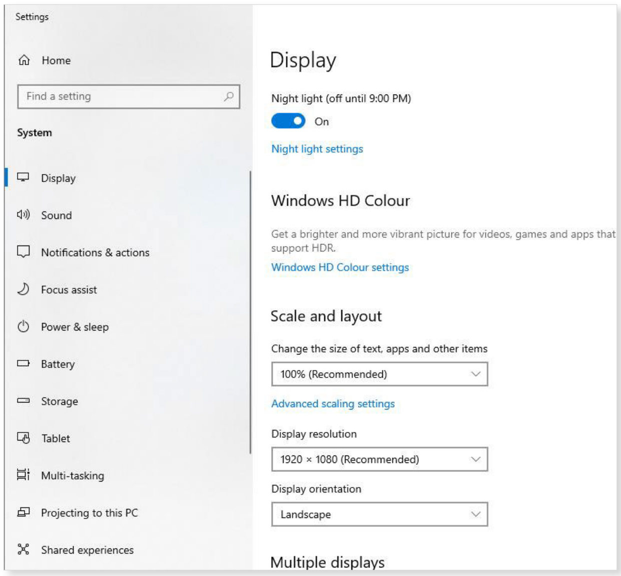
T-way testing, also known as Combinatorial testing, is a minimization technique used to generate a test suite based on interaction fault detection. The t in T-way stands for the interaction length. For a better understanding, let's consider an example of a Scale and Layout setting under the Display setting in Windows 10. For this example, there are three parameters which are:

1. Night light (On, Off)
2. Change the size of text, apps and other items has two values (100% and 125%)
3. Display resolution has six values (1366 × 768, 1280 × 768, 1280 × 720, 1280 × 600, 1024 × 768, 800 × 600)
4. Display orientation has two values (Landscape and Portrait)

All the variations of the inputs are considered for testing the environment. There are 2 (Night light) × 2 (Text size) × 6 (Display resolution) × 2 (Display orientation) = 48 test cases that need to be checked that can cover all the interactions in **FIGURE 1.** However, the size of the test suite can be reduced from 48 to 12 test cases by considering two-way interaction. Two-way interaction testing guarantees that any two- combination of parameters covers at least one time in the final test suite. It should be noted that the size of the test suite is 75% reduction, but the fault detection rate can reach 90%. Three-way testing unlike two-way testing, improves the identification of faults to 99% which is a great chance of finding faults effectively in a reasonable time.

2) BOUNDARY VALUE ANALYSIS

In software testing, partition testing divides the input space into smaller partitions then, from each partition, test cases are selected. If any test case is selected from one partition and



Display

Night light (off until 9:00 PM)

On

Night light settings

Windows HD Colour

Get a brighter and more vibrant picture for videos, games and apps that support HDR.

Windows HD Colour settings

Scale and layout

Change the size of text, apps and other items

100% (Recommended)

Advanced scaling settings

Display resolution

1920 × 1080 (Recommended)

Display orientation

Landscape

Multiple displays

FIGURE 1. Scale and layout setting.

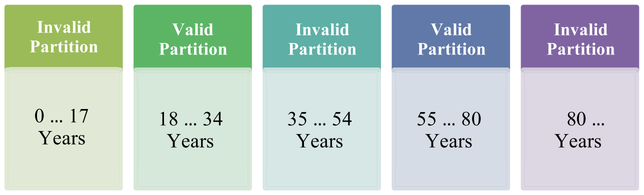


FIGURE 2. Boundary partitioning example.

tested, the results should be the same every time. For example, if we have a system that **FIGURE 2** shows. Then only one value is selected from each partition. shows. Then only one value is selected from each partition.

BVA is similar to partition testing however, the difference between them is, in BVA, the boundary values between partitions (i.e. maximum and minimum values) are included in test cases to identify potential defects that usually occur at the boundaries of input ranges. In addition, BVA considers some values as output of input space. In our earlier example, the values -1, 0, 17, 18, 34, 35, 54, 55, 88, and 81 must be included in the test cases. Partition testing is an easy way to reduce the test cases while BVA has a great ability to identify potential defects that occur at the boundaries of input ranges [34].

3) CUCKOO SEARCH

CS is a population-based algorithm inspired by brood parasite behavior in some birds like Guira cuckoos. CS provides an ideal balance between local intensification and global diversification by intensifying the search for solutions in the incumbent solution neighborhood, as well as efficiently expanding the search space by using Levy flights. CS has only one parameter, P_a, to be tuned, less than the number of parameters in the GA and PSO.

```
Cuckoo Search via Lévy Flights
begin
  Objective function $f(\mathbf{x})$,  $\mathbf{x} = (x_1, ..., x_d)^T$
  Generate initial population of
      $n$ host nests $\mathbf{x}_i$ $(i = 1, 2, ..., n)$
  while ($t$ <MaxGeneration) or (stop criterion)
    Get a cuckoo randomly by Lévy flights
      evaluate its quality/fitness $F_i$
    Choose a nest among $n$ (say, $j$) randomly
    if  $(F_i > F_j)$,
        replace $j$ by the new solution;
    end
    A fraction ($p_a$) of worse nests
        are abandoned and new ones are built;
    Keep the best solutions
        (or nests with quality solutions);
    Rank the solutions and find the current best
  end while
  Postprocess results and visualization
end
```

FIGURE 3. Pseudocode of the cuckoo search [35].

The algorithm begins by generating an initial nest population. There are two operations performed within each generation of the algorithm. First, create a new nest by carrying out a levy flight, and then evaluate the new nest and remove the existing nest if a better evaluation is obtained. The second part of the algorithm finds the bad nests and eliminates them with probability, P_a, [35]. The pseudocode in **FIGURE 3** summarizes the steps of the CS algorithm.

B. CUCKOO SEARCH BASED ON ADAPTIVE FUZZY LOGIC CONTROL OF GENETIC ALGORITHM OPERATORS FOR PRIORITIZED TEST CASE GENERATION

The CS-FuzGA-PTS is designed to generate the PTS. PTS is a set of ordered test cases designed to maximize the probability of fault detection based on certain criteria. The CS-FuzGA-PTS follows a systematic approach to generate PTS. The CS-FuzGA-PTS optimizes the PTS by incorporating both t-way test coverage and test BVA. The combination of these two testing techniques enables efficient, targeted test case generation. In general, to generate the PTS, the CS-FuzGA-PTS starts by generating all the t-combinations of the inputs and boundary values of input ranges, and then it iteratively generates the test cases that maximize the test coverage and test priority. The procedure of generating PTS using CS-FuzGA-PTS can be seen as two main phases:

1) GENERATING T-COMBINATIONS AND BOUNDARY VALUES

Based on the SUT configuration, the CS-FuzGA-PTS starts by generating a list of all the t-combinations of the inputs and lists the boundary values of the input ranges. CS-FuzGA-PTS receives the SUT configuration, including the number of inputs, the boundaries of the input ranges, and the interaction level (t value). Based on the t-way testing concept as the base of the test case reduction techniques, the CS-FuzGA-PTS generates all the t-combinations of the inputs according to the configuration of the SUT. The t-combination is a list of input combinations such that each combination covers the interaction of a specific number of inputs. For example, 2-combinations (or t-way combinations) cover all pairs of the inputs, and 3-combinations cover all triplets of the inputs. CS-FuzGA-PTS also identifies the boundary values of the inputs and stores them in a boundary values list. The boundary value list consists of crucial boundaries for the input. In addition to the edges of the input, the subranges can be involved in the boundary value list.

2) PRIORITIZED TEST SUITE GENERATION

The core part of the CS-FuzGA-PTS method is to generate a set of test cases that maximize test coverage and fault detection. To achieve this objective, CS-FuzGA-PTS uses CS-FuzGA as an optimization algorithm for finding the optimal test case that covers the maximum number of t-combinations and boundary values. During the test case generation, the CS-FuzGA explores the search space, which is the list of t-combinations, and the boundary values list to find a set of test cases that meet the objective of maximizing test coverage and fault detection. Each candidate solution of CS-FuzGA, which is a set of possible inputs, presents a test case. CS-FuzGA-PTS follows OTAT for generating the test suite; therefore, it iteratively attempts to generate one test case at each iteration that optimally fulfils the two objectives of CS-FuzGA-PTS. The best test case generated by CS-FuzGA is added to the final test suite list. Subsequently, the t-combinations and boundaries covered by this test case are deleted from the respective lists. This process of finding the optimal test case and removing the covered elements are repeated until the two lists are devoid of elements.

The output of these two phases is the final test suite accompanied by two testing metrics. The final test suite includes the set of test cases, t-way coverage weight of each test case, and boundary values analysis weight of each test case. The t-way coverage weight measures the extent to which each test case covers different interaction combination inputs with size t, which basically measures how well the test is able to explore software interaction inputs. On the other hand, the boundary values analysis weight measures the efficiency of the test case in exposing potential issues near boundary ranges. To construct the PTS, testers have the flexibility to organize the PTS based on their preference, either based on the test case coverage weight or the boundary values weight. **FIGURE 4** summarizes the steps of CS-FuzGA-PTS.

C. CS-FUZGA OPTIMIZATION ALGORITHM

CS-FuzGA, as an optimization algorithm, serves as the core implementation search algorithm used in CS-FuzGA-PTS for finding optimal test cases. The CS-FuzGA is an enhancement algorithm of the CS algorithm, it attempts to enhance the balance between exploration and exploitation and premature

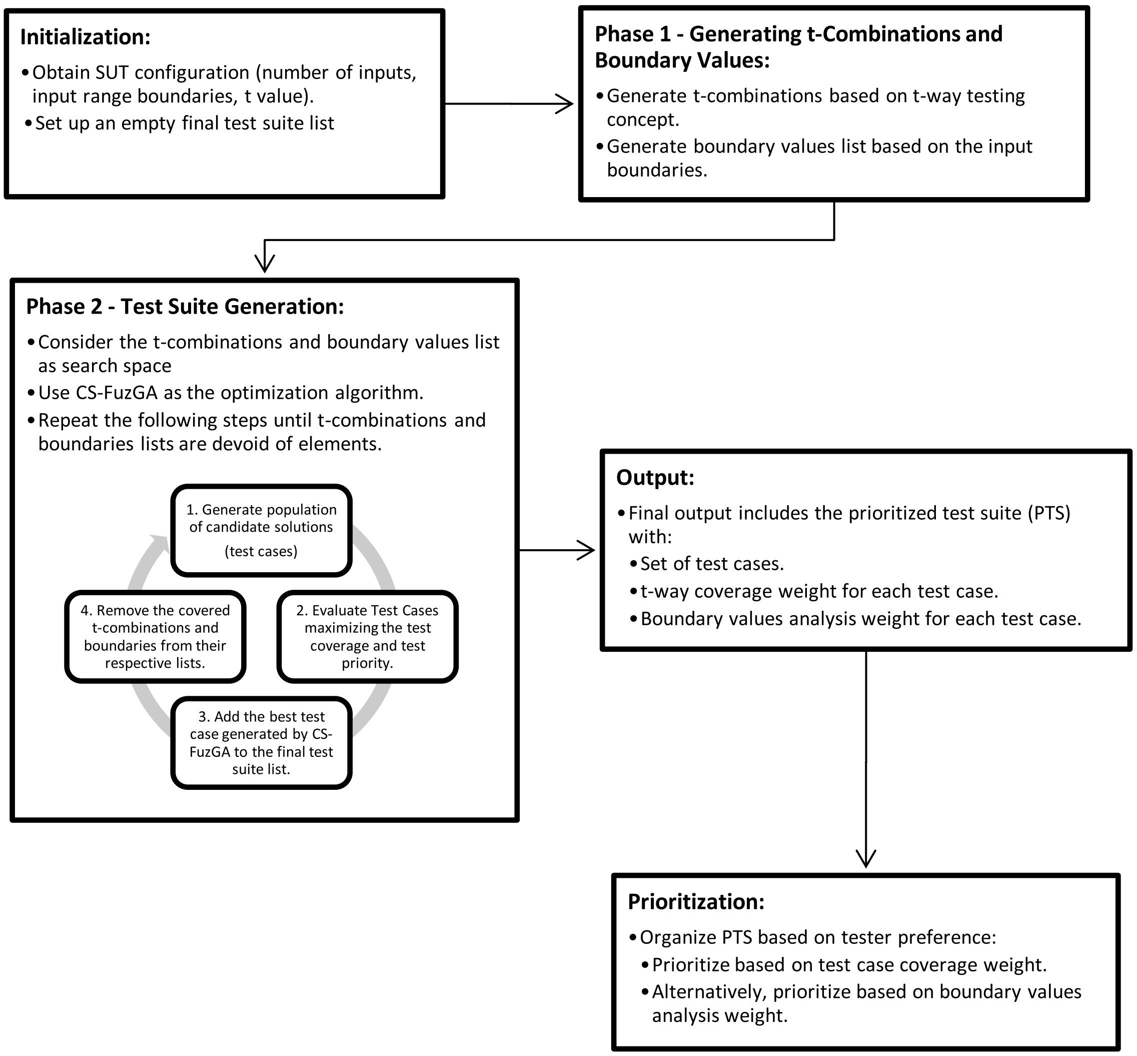

Initialization:
• Obtain SUT configuration (number of inputs, input range boundaries, t value).
• Set up an empty final test suite list
Phase 1 - Generating t-Combinations and Boundary Values:
• Generate t-combinations based on t-way testing concept.
• Generate boundary values list based on the input boundaries.
Phase 2 - Test Suite Generation:
• Consider the t-combinations and boundary values list as search space
• Use CS-FuzGA as the optimization algorithm.
• Repeat the following steps until t-combinations and boundaries lists are devoid of elements.
1. Generate population of candidate solutions (test cases)
2. Evaluate Test Cases maximizing the test coverage and test priority.
3. Add the best test case generated by CS-FuzGA to the final test suite list.
4. Remove the covered t-combinations and boundaries from their respective lists.
Output:
• Final output includes the prioritized test suite (PTS) with:
• Set of test cases.
• t-way coverage weight for each test case.
• Boundary values analysis weight for each test case.
Prioritization:
• Organize PTS based on tester preference:
• Prioritize based on test case coverage weight.
• Alternatively, prioritize based on boundary values analysis weight.

FIGURE 4. CS-FuzGA-PTS steps.

convergence. To achieve this goal, CS-FuzGA integrates fuzzy logic control and genetic operators in CS. CS-FuzGA monitors and enhances the performance of CS-FuzGA based on the performance of the algorithm by injecting the required GA operators to enhance the performance. As each GA operator has its own search capabilities, the integration of GA operators and fuzzy logic control in CS allows it to dynamically balance exploration and exploitation during the search process. Unlike the standard CS which relies on fixed parameter settings to balance between exploration and exploitation, one of the most unique features of CS-FuzGA is the use of fuzzy logic to manage this balance and control the algorithm behavior overall. The risk of premature convergence to local optima is a common challenge in metaheuristic algorithms. The CS-FuzGA attempts to mitigate this through a synergistic combination of fuzzy logic control and GA operators. First, the fuzzy logic dynamically balances exploration and exploitation by adjusting CS parameters based on population diversity, therefore preventing excessive focus on one area. Additionally, the integration of GA operators into CS enhances CS-FuzGA global search ability, thus ultimately reducing the likelihood of becoming trapped in local optima.

Here are the steps of the CS-FuzGA algorithm for generating test cases:

Step 1: Initialization of Cuckoo Eggs

According to the initial setting of the algorithm, CS-FuzGA starts by initializing the population of cuckoo eggs, which it generates randomly based on the search space. CS-FuzGA also sets up common parameters such as maximum iteration and size of the population. In order to reduce the parameters of CS-FuzGA that need initial values, CS-FuzGA has been designed to set up some parameters such as mutation rate of GA and switch probability of CS based on the performance of CS-FuzGA. This step ends by evaluating the coverage and boundary value weights of each cuckoo egg.

Step 2: Improvement process of CS-FuzGA

After the population of the cuckoo egg is initialized, the population is iteratively subject to an improvement process.

The improvement process of CS-FuzGA can be seen in two parts:

- First, the cuckoo eggs are subjected to CS improvement process, including moving the eggs to new nests using levy flight or abandoning and replacing them.
- Secondly, the population is subjected to the enhanced part of CS. In this part, CS-FuzGA integrates the GA operators, including mutation and crossover, into its search process. The operators are wisely injected into the search process to maintain the balance between exploration (mutation) and exploitation (crossover). The decision to favor either exploration or exploitation is influenced by the adaptive fuzzy logic control based on population diversity and performance. To select the favor operator, CS-FuzGA calculates the diversity of the population.

CS-FuzGA calculates the diversity within the population and then bases the diversity score, CS-FuzGA selects the genetic operator that favors either exploration or exploitation. To do that, CS-FuzGA starts calculating the membership of the population members for low, medium, and high diversity levels, and based on the membership, the algorithm selects a genetic operator that either emphasizes exploration (mutation) or exploitation (crossover).

IV. RESULTS AND DISCUSSION

This section aims to present the experimental evaluation of the proposed works against the existing works. The performance of CS-FuzGA-PTS is compared against the existing works in terms of test case reduction and defect identification. This section is divided into several subsections: the first section presents the experiment setup, while the second section evaluates the CS-FuzGA-PTS against the existing strategies in terms of test case reduction. The third section evaluates the effectiveness of CS-FuzGA-PTS in terms of defect detection. Lastly, the fourth section evaluates the proposed optimization algorithm CS-FuzGA against the standard CS and GA in terms of convergence rate.

A. EXPERIMENTAL SETUP

To conduct the experiment, the CS-FuzGA-PTS, standard CS, GA and random search are implemented using Python to generate test cases for software input. Additionally, results of existing works such as SA, GA, PSO, FPA, and CS will be collected from the published papers. Specifically, the efficiency of the proposed method will be evaluated against the existing works in terms of generating the optimal test cases. Second, the effectiveness of defect identification using the proposed method will be evaluated. Lastly, the performance of CS-FuzGA as an optimization algorithm will be evaluated against the standard CS and GA solution quality and convergence rate. The experiments were performed on a laptop with specifications of an Intel (R) Core (TM) i7-3770 CPU@ 3.40 GHz - 3.40 GHz, 8GB of RAM, Windows 10, and 64-bit operating system. For the parameter setting, CS-FuzGA-PTS has been tuned, resulting in a maximum iteration of 200 and a population size of 30, as indicated by the convergence rate analysis in this study. while the other parameters, such as mutation rate, and the probability of CS, used the recommended default values. Meanwhile, for other optimization methods, the recommended settings from the original papers are used. Table 1 represents the parameters that are adopted for the meta-heuristic strategies [3], [5], [6], [36].

TABLE 1. Parameters for meta-heuristic strategies of interests.

Algorithm	Parameter	Values
PSO	Maximum iteration	100
	Population size	80
	Inertia weight	0.3
	Acceleration coefficients	1.375
CS	Maximum iteration	100
	Population size	100
	Probability *ep*	0.25
FPA	Maximum iteration	300
	Population size	100
	Probability *pa*	0.25
HSS	Maximum iteration	1000
	Harmony memory size	100
	Harmony memory consideration rate	0.7
	Pitch adjustment rate	0.2
CS-FuzGA	Maximum iteration	400
	Population size	40
	Probability *ep*	0.25
	Random crossover	0.75
	Mutation rate	0.03

B. TEST CASE REDUCTION

This experiment evaluates the ability of CS-FuzGA-PTS to minimize the test cases of the SUT while maintaining sufficient coverage. In this section, CS-FuzGA-PTS is compared with existing t-way testing methods such as mAETG, AETG, IPOG, Jenny, TVG, PSO, HSS, FPA and CS. The results of these methods are collected from the available research [3], [5], [36], [37].

For a comprehensive comparison, this section presents experiments employing three well-known real-world system configurations that are commonly used in t-way testing research. These experiments aim to evaluate the performance and effectiveness of the CS-FuzGA-PTS test case generation method across a range of complexities.

1. Experiment 1: A system with a configuration of 10 inputs, each input with 2-values while the coverage strength, t, is varied from 2 to 10.

TABLE 2. Comparison with existing strategies for experiment 1.

t	Computational-based Strategies								Meta-heuristic Strategies					
	IPOG	TVG	Jenny	TConfig	PICT`	GTWay	ITCH	CTE-XL	PSO	HSS	CS	FPA	CS-FuzGA-PTS	
													Avg	Best
2	10	10	10	9	NA	NA	6	NA	8	7	8	8	8.4	8
3	19	17	18	20	NA	NA	18	NA	17	16	16	16	16.7	16
4	49	41	39	45	NA	NA	58	NA	37	37	36	35	38.7	36
5	128	84	87	95	NA	NA	NS	NA	82	81	79	81	83.0	75
6	352	168	169	183	NA	NA	NS	NA	158	158	157	158	158.9	155
7	NS	302	311	NS	NA	NA	NS	NA	NS	298	NS	292	293.0	290
8	NS	514	521	NS	NA	NA	NS	NA	NS	498	NS	500	503.7	493
9	NS	651	788	NS	NA	NA	NS	NA	NS	512	NS	592	627.8	552
10	NS	NS	1024	NS	NA	NA	NS	NA	NS	1024	NS	1024	1024	1024

TABLE 3. Comparison with existing strategies for experiment 2.

v	Computational-based Strategies								Meta-heuristic Strategies					
	IPOG	TVG	Jenny	TConfig	PICT	GTWay	ITCH	CTE-XL	PSO	HSS	CS	FPA	CS-FuzGA-PTS	
													Avg	Best
2	46	40	39	45	43	46	58	NA	34	37	28	36	40.5	36
3	229	228	221	235	231	224	336	NA	213	211	211	211	212.3	211
4	649	782	703	718	742	621	704	NA	685	691	698	661	668.8	664
5	1843	1917	1719	1878	1812	1714	1750	NA	1716	1624	1731	1605	1634.2	1631
6	3808	4159	3519	NA	3735	3514	NA	NA	3880	3475	3894	3354	3487.3	3406
7	7061	7854	6462	NA	NA	6459	NA	NA	NA	6398	NA	6205	6586.6	6415

TABLE 4. Comparison with existing strategies for experiment 3.

p	Computational-based Strategies								Meta-heuristic Strategies					
	IPOG	TVG	Jenny	TConfig	PICT	GTWay	ITCH	CTE-XL	PSO	HSS	CS	FPA	CS-FuzGA-PTS	
													Avg	Best
5	908	849	837	773	810	731	625	NA	779	751	776	784	761.7	749
6	1239	1128	1074	1092	1072	1027	625	NA	1001	990	991	988	996.2	992
7	1349	1384	1248	1320	1279	1216	1750	NA	1209	1186	1200	1164	1208.2	1128
8	1792	1595	1424	1532	1468	1443	1750	NA	1417	1358	1415	1329	1330.4	1394
9	1793	1795	1578	1724	1643	1579	1750	NA	1570	1530	1562	1478	1472.4	1448
10	1965	1971	1791	1878	1812	1714	1750	NA	1716	1624	1731	1605	1677.2	1624
11	2091	2122	1839	2038	1957	1852	1750	NA	1902	1860	2062	1739	1802.3	1742
12	2285	2268	1964	NA	2103	2022	1750	NA	2015	2022	2223	1879	1861.6	1852

2. Experiment 2: A system with a configuration of 10 inputs, each input with p-values where p is varied from 2 to 7, and the coverage strength is 4.
3. Experiment 3: A system with a configuration of i inputs, where i is varied from 5 to 10, each input with 5-values and the coverage strength 4.

In analyzing the above experimental results, the results reveal distinct trends among computational-based and meta-heuristic testing strategies. The results of experiment 1 in Table 2 show that computational-based strategies, such as IPOG, TVG, Jenny, and TConfig, demonstrate reasonable performance in terms of test case reduction, where TVG and Jenny outperform other strategies across different values of t. Concerning meta-heuristic strategies, CS-FuzGA-PTS achieves the best performance in most of the cases; it consistently outperforms other meta-heuristic strategies.

The results of computational-based strategies for experiment 2 show varied and competitive performance across different levels of input. GTWay consistently outperforms the other strategies, while Jenny produces competitive results. In this experiment, meta-heuristic strategies demonstrate strong performance in minimizing test cases compared to computational-based strategies. Both CS-FuzGA-PTS and FPA consistently outperform other meta-heuristic strategies in most cases, achieving the best test suite size across various input levels.

Similarly, the results of Experiment 3 show that ITCH and GTWay consistently maintain competitive performance, demonstrating their robustness across different configurations. On the other hand, meta-heuristic strategies show strong performance compared to computational-based strategies. While CS-FuzGA-PTS is not the best in this experiment, it shows very competitive results. The FPA strategy outperforms the other strategies in most cases.

Transitioning into the broader discussion, the initial observation underscores that meta-heuristic strategies exhibit strong performance compared to computational-based strategies. In the same context, the experiments involve a variety of system configurations, including different numbers of inputs, values, and coverage strengths. It has been observed that varying system configurations influence the performance of both computational-based and meta-heuristic strategies. However, it is noteworthy that none of the strategies achieve optimal results across all configurations.

Considering computational-based strategies, we can observe that TVG, Junny, ITCH, and GTWay consistently perform better than other computational-based strategies. This might be because most of these strategies employ advanced techniques that allow them to identify and eliminate redundant test cases more effectively. For example, TVG utilizes static analysis by analyzing the control flow graph of the program, then employs heuristic search and redundant test vector elimination. Similarly, Junny employs adaptive heuristic search and a merging technique that combines test cases covering similar interactions for further reduction. The outstanding performance of ITCH and GTWay in reducing test cases stems from their efficient algorithms, due to their dynamic adaptability and scalability to higher-order interactions. ITCH's heuristic and incremental approach starts with pairwise tests and gradually increases the complexity of test cases, allowing it to generate competitive results. GTWay integrates test generation with execution, enabling the identification and elimination of redundant test cases, further reducing size.

On the other hand, FPA and CS-FuzGA-PTS consistently outperform other meta-heuristic strategies in most cases, while CS stands out in some cases. The results show the robustness of FPA and CS-FuzGA-PTS under different configurations, where both are able to achieve the best results across different configurations. This might be because most of these strategies employ techniques that allow them to adapt their performance to different configurations. The authors believe that the combination of GA operators in CS-FuzGA-PTS, controlled by the fuzzy technique plays a vital role in achieving good results. Also, the utilization of the Levy flight in both FPA and CS-FuzGA-PTS may be identified as another contributing factor to achieving good results. The results also show that CS and HSS produce a competitive result, while PSO exhibits competitive performance as well, but it fails to record any best results across all the configurations. It might face challenges in achieving an optimal balance between exploration and exploitation within the search space. It is worthwhile to mention that CS-FuzGA-PTS is designed to self-adapt between exploration and exploitation to achieve an optimal balance between the two, as the selection of GA operators is based on the performance of CS-FuzGA-PTS.

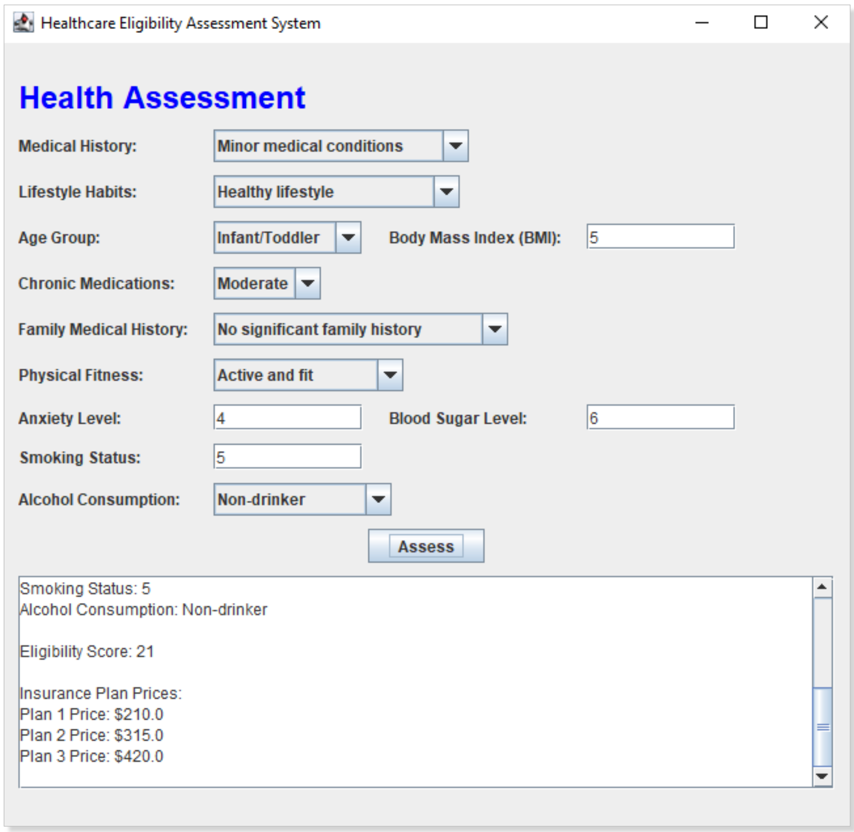
C. PRIORITIZATION AND DEFECT ANALYSIS

This section aims to evaluate the effectiveness of CS-FuzGA-PTS in prioritizing test cases and identifying potential defects. The section exhibits the prioritization capabilities of CS-FuzGA-PTS compared to existing methods. The evaluation is structured as two experiments. The effectiveness of CS-FuzGA-PTS is evaluated in terms of identifying bugs using a real-world application.

In this evaluation, we use the Healthcare Eligibility Assessment System (HEAS) as a real-world application to evaluate the proposed method. HEAS is a GUI-based software used to assess the health status of customers to determine their eligibility for certain services, such as medical treatments, job eligibility, insurance coverage, or other services. The application has been selected because it involves various input types, user interactions, calculations, decision-making processes, and stakeholder impacts within a manageable context. As **FIGURE 5** shows, HEAS calculates the customer's score based on various health factors and criteria, including:

- Medical History: [No significant medical history, Minor medical conditions, Chronic conditions, Serious illnesses, Terminal conditions]
- Lifestyle Habits: [Healthy lifestyle, Occasional unhealthy habits, Regular unhealthy habits, Sedentary lifestyle]
- Age Group: (Enter numerical age value) [Infant/Toddler, Child/Teenager, Adult, Senior Citizen]
- Body Mass Index (BMI): (Enter BMI value) [Underweight, Normal, Overweight, Obese]
- Chronic Medications: [None, Few, Moderate, Many]
- Family Medical History: [No significant family history, Family history of certain conditions]
- Physical Fitness: [Active and fit, Moderately fit, Low physical activity]
- Blood Pressure: (Enter numerical blood pressure value) [Normal, Prehypertension, Hypertension]
- Blood Sugar Level: (Enter numerical blood sugar level value) [Normal, Prediabetes, Diabetes]
- Smoking Status: [Non-smoker, Ex-smoker, Current smoker]
- Alcohol Consumption: [Non-drinker, Occasional drinker, Regular drinker].

To evaluate the effectiveness of CS-FuzGA-PTS in identifying software defects, we execute the PTS generated by CS-FuzGA-PTS during the testing process on HEAS



Medical History:	Minor medical conditions
Lifestyle Habits:	Healthy lifestyle
Age Group:	Infant/Toddler
Body Mass Index (BMI):	5
Chronic Medications:	Moderate
Family Medical History:	No significant family history
Physical Fitness:	Active and fit
Anxiety Level:	4
Blood Sugar Level:	6
Smoking Status:	5
Alcohol Consumption:	Non-drinker

```
Smoking Status: 5
Alcohol Consumption: Non-drinker

Eligibility Score: 21

Insurance Plan Prices:
Plan 1 Price: $210.0
Plan 2 Price: $315.0
Plan 3 Price: $420.0
```

FIGURE 5. Healthcare eligibility assessment system.

TABLE 5. HEAS defect identification result using CS-FuzGA-PTS and random generation.

Test Case Configuration	Tool	Mutation Kill Rate	Line Coverage	Total Time (seconds)
5 test cases	Random Generation	29%	83%	6
	CS-FuzGA	54%	95%	5
10 test cases	Random Generation	29%	92%	12
	CS-FuzGA	79%	98%	5
15 test cases	Random Generation	50%	92%	11
	CS-FuzGA	83%	98%	5
20 test cases	Random Generation	67%	95%	9
	CS-FuzGA	83%	98%	6
25 test cases	Random Generation	83%	97%	8
	CS-FuzGA	100%	98%	6

as the system under test. To measure defect identification, we employed Mutation Testing with PIT (PITest). PITest is a powerful tool used to assess the effectiveness of test suite for identifying defects in the source code. It accomplishes this by generating mutated versions of the source code that include software defects. PITest generates different kinds of defects in the source code, such as Conditional Changes, Arithmetic Mutations, Variable Manipulation, Exception Handling, Boundary Changes, Statement Deletion and Operator Alterations. As the proposed method emphasizes prioritizing test cases through BVA, Boundary Changes mutation is exclusively used to mitigate any potential influences on the results.

Table 5 shows the comparison results of defect identification using the PTS generated by CS-FuzGA-PTS compared with random generation. The table displays two important metrics for the evaluation, including mutation kill rate (a.k.a. mutator coverage) and average line coverage. In addition to that, it displays the average analysis time. Mutation kill rate presents the percentage of the number of killed mutations to

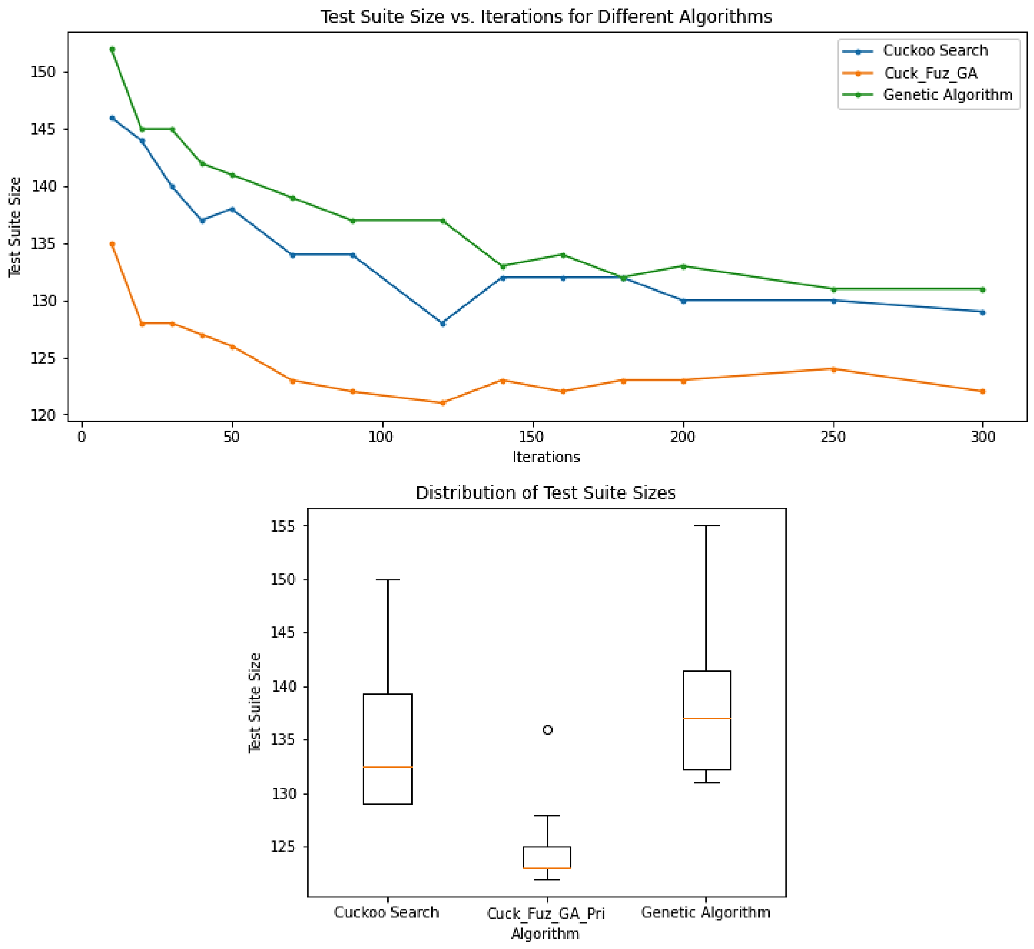


FIGURE 6. **Comparison of convergence rate for system 1.**

the total number of mutations, while line coverage presents the percentage of covered lines of code to the total lines of code. The comparison shows that CS-FuzGA always tends to achieve a higher mutation kill rate and line coverage compared to random generation, and it is able to detect all the defects by executing a certain number of high-priority test cases. While exhaustive testing requires up to ($30 \times 10\times10 \times 20\times4 \times 5\times4 \times 4\times2 \times 3\times3$) 3715 test cases, only 25 test cases can achieve 100% mutation coverage and kill all the generated mutations.

The results obtained from this experiment provide insights into the effectiveness of CS-FuzGA-PTS in generating a test suite that is able to identify potential defects occurring at input boundaries. By optimizing and prioritizing the test cases based on coverage weight and BVA, the proposed method outperforms the other methods in terms of coverage weight, test suite size, and defect detection.

D. CONVERGENCE RATE

The aim of this experiment is to evaluate the performance of CS-FuzGA as an optimization algorithm. The evaluation aims to observe and analyze the convergence rate of CS-FuzGA compared to standard CS and GA. The convergence rate measures how fast the algorithm can reach the optimal solution as the number of iterations increases. To measure the convergence rate, the results of the three algorithms are tracked over a series of iterations as shown in Table 6. The results of the experiment are also presented in **FIGURES 6** and **7** of two systems configurations:

- System 1 has 5 inputs, each with 10 values, and a coverage strength of 2,
- System 2 is HCS, with a coverage strength of 2.

From the line plots in the figures, we track the obtained results over the iterations, while in the box plot on the right side, we see the distribution of the obtained test suite size.

For both systems, the performance of CS-FuzGA is superior to the other two algorithms, as shown in the line plot. Overall, the iterations, the proposed method produces an optimal number of test cases than CS and GA. Box plots also show that the distribution of CS-FuzGA test suite sizes is always smaller than the other two algorithms. It also shows that the average of test suite sizes (as displayed by the orange line in the boxes) is almost at the bottom of the boxes,

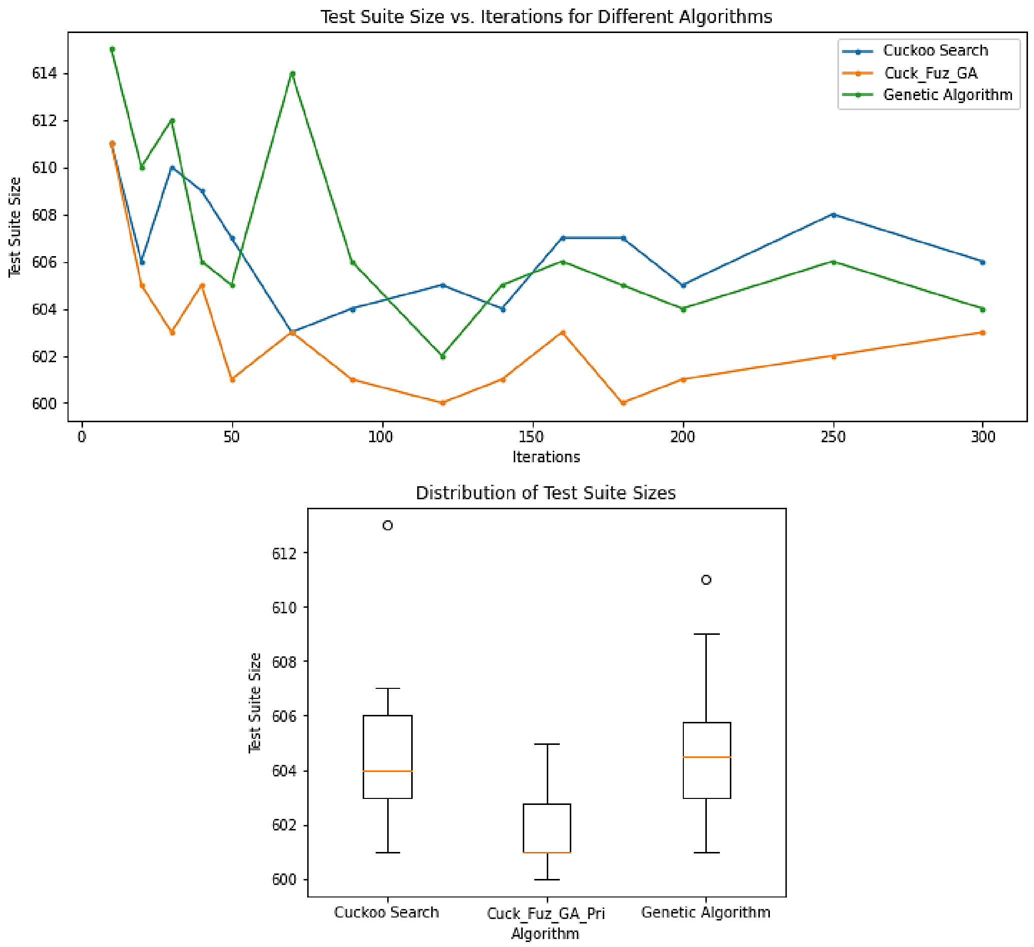


FIGURE 7. Comparison of convergence rate for HCS system.

meaning that the average of test suite sizes is the same as the minimum test suite size. Box plots also show that the worst results obtained by CS-FuzGA are still better compared to the results obtained by CS and GA.

The results indicate that the integration of the CS and GA operators with fuzzy logic control is able to accelerate the search process of CS-FuzGA compared to standard CS and GA. By continuously monitoring the diversity of population and coverage rate, fuzzy logic control is dynamically able to adjust the contributions of CS and GA operators throughout the optimization process. This dynamic adaptation enables the CS-FuzGA algorithm to strike an optimal balance between exploration and exploitation, enhancing the search quality.

E. COMPLEXITY ANALYSIS

The complexity of the algorithm is important for evaluating its scalability and performance. Especially when it applies to large-scale problems. This section analyses the CS-FuzGA algorithm in comparison to standard cuckoo search (CS) and genetic algorithms (GA) in terms of time complexity. complexity of space and overall efficiency.

The time complexity of CS depends on the number of iterations and the number of solutions or nests. Each iteration creates a new solution and replaces the worst performing nest. This can generally be expressed as follows:

$$O(n \cdot I)$$

where I is the number of iterations and n is the number of nests.

The time complexity of GA depends on the number of iterations, and time complexity of each GA operators. The time complexity of GA can generally be expressed as follows:

$$O(n \cdot I \cdot (s + c + m))$$

where I is the number of iterations or generation, n is the number of chromosome, s is the time complexity of selection, c is the time complexity of crossover, and m is the time complexity of mutation.

The time complexity of CS-FuzGA depends on the number of iterations, time complexity of one selected GA operator and Overhead from the adaptive fuzzy logic control. The time complexity of CS-FuzGA can generally be expressed as

TABLE 6. Analyzing the convergence rate for system 1 AND HCS system.

Algorithm	Iteration	Test suite size System 1	Test suite size HCS system
Cuck_Fuz_GA	10	609	136
	20	604	128
	40	603	128
	60	602	124
	60	602	125
	100	601	123
	120	602	123
	140	604	123
	160	600	123
	180	601	125
	200	602	122
	220	602	123
	240	600	123
	260	602	123
	280	600	123
	300	603	122
Cuckoo search	10	616	150
	20	612	145
	40	609	141
	60	606	140
	60	604	137
	100	607	136
	120	603	134
	140	604	131
	160	604	129
	180	605	129
	200	604	129
	220	602	129
	240	603	129
	260	604	129
	280	605	129
	300	604	128
Genetic algorithm	10	613	155
	20	613	145
	40	609	142
	60	609	140
	60	604	143
	100	604	138
	120	605	136
	140	604	138
	160	606	134
	180	611	133
	200	606	132
	220	602	132
	240	603	131
	260	604	131
	280	604	131
	300	606	131

follows:

$$O(I \cdot (n + (s + c + m) + k + d))$$

where:

O(I): The cost of I iterations, encompassing both CS and GA operator execution.

O(n): The cost of managing n nests in the CS.

O(s + c + m): This reflects the cost of applying one of the GA operators (s: selection, c: crossover, or m: mutation) per iteration, chosen based on algorithm performance. It avoids the full complexity of a complete GA cycle.

O(k): The overhead of the adaptive fuzzy logic control, generally considered a small constant.

O(d): The computational cost of evaluating population diversity and selecting the appropriate GA operator. This is also expected to be relatively small compared to the other terms.

This analysis indicates CS-FuzGA has a significant scalability advantage over standard GA. Although CS has a simpler time complexity, CS-FuzGA improves its search capabilities by integrating GA operators. However, CS-FuzGA maintains a lower cost by executing only one GA operator per iteration within CS loop. The additional costs of fuzzy logic control (k) and different computations (d) are relatively small. In general, the time complexity of CS-FuzGA is favorable when compared to the potential performance and scalability benefits of CS-FuzGA. As a result, CS-FuzGA is well balanced between the simplicity of CS and the power of GA, making it suitable for large-scale problems.

V. CONCLUSION

In this paper, we have proposed and implemented a method called CS with Adaptive Fuzzy Logic-Controlled GA Operators for PTS Generation (CS-FuzGA-PTS). CS-FuzGA-PTS aims to generate a PTS that combines t-way testing for test case reduction and boundary analysis for prioritizing test cases based on potential defects at boundaries of input ranges. CS-FuzGA-PTS optimizes the test cases using the CS-FuzGA algorithm as the core search algorithm. CS-FuzGA integrates CS with adaptive fuzzy logic control and GA operators to continuously monitor the diversity of potential solutions enhancing the search capability. Considering the test results, the research achieved its objectives, as the PTS generated using CS-FuzGA-PTS demonstrates superior performance in identifying defects. Furthermore, the experiment shows CS-FuzGA-PTS ability to surpass existing t-way strategies in terms of test reduction. From an optimization perspective, the proposed CS-FuzGA accelerates the search process and obtains a better convergence rate compared to standard CS and GA. Overall, the results suggest that CS-FuzGA-PTS is a promising approach for test reduction and fault detection due to its crafting design that integrates t-way testing and boundary analysis techniques from the software testing domain and CS, GA operators, and fuzzy logic control from the optimization domain. Furthermore, CS-FuzGA-PTS is able to monitor and adjust its performance based on the results obtained, which can boost either exploration or exploitation by measuring the diversity of the population.

The proposed CS-FuzGA-PTS has a significant implication for the field of software testing, as the software systems become increasingly complex. There are practical applications of the proposed method across various industries where software failures can lead to significant to substantial losses, including Finance and Banking, Healthcare, and E-commerce systems, to name a few. In these applications, the CS-FuzGA-PTS can enhance the reliability of applications by effectively identifying the critical defects that may occur at input limits, thereby improving system robustness. Furthermore, by using T-way testing and boundary value analysis, CS-FuzGA-PTS reduces the number of required test cases. while maintaining comprehensive coverage. This reduction directly reduces the overall costs, including the resources required for testing and time and manpower. On the other hand, there are some limitations raised when applying CS-FuzGA-PTS to different complexities of software projects. As the software system gets more complex, CS-FuzGA-PTS may face challenges to efficiently generate and prioritize all relevant test cases due to the combinatorial explosion of possible test cases. Additionally, the CS-FuzGA-PTS relies heavily on BVA to prioritize test cases, which require domain expertise to identify these critical boundary values.

For future work, there is still room for improving the performance of CS-FuzGA-PTS by exploring the integration of other metaheuristic algorithms beyond CS. Furthermore, due to its adaptive nature, CS-FuzGA can be utilized for solving other real-world optimization problems, such as travel salesman problem, software product line, and global optimization problems.

REFERENCES

[1] J. Stardom, *Metaheuristics and the Search for Covering and Packing Arrays*. Burnaby, BC, Canada Simon Fraser Univ., 2001.

[2] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf.*, 2004, pp. 72–77.

[3] 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.

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

[5] 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.

[6] 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.

[7] A. B. Nasser, K. Z. Zamli, A. A. Alsewari, and B. S. Ahmed, "An elitist-flower pollination-based strategy for constructing sequence and sequence-less t-way test suite," *Int. J. Bio-Inspired Comput.*, vol. 12, no. 2, p. 115, 2018.

[8] K. Z. Zamli, "An improved Jaya algorithm-based strategy for T-way test suite generation," in *Emerging Trends in Intelligent Computing and Informatics (IRICT)* (Advances in Intelligent Systems and Computing), vol. 1073. Cham, Switzerland: Springer, 2019, pp. 352–361.

[9] A. B. Nasser, A. Alsewari, and K. Z. Zamli, "Learning cuckoo search strategy for t-way test generation," in *Proc. Int. Conf. Comput., Anal. Netw.*, 2017, pp. 97–110.

[10] M. S. A. R. Ali, R. R. Othman, Z. R. Yahya, and M. Z. Zahir, "A modified artificial bee colony based test suite generation strategy for uniform T-way testing," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 767, no. 1, Feb. 2020, Art. no. 012020.

[11] E. Nabil, "A modified flower pollination algorithm for global optimization," *Expert Syst. Appl.*, vol. 57, pp. 192–203, Sep. 2016.

[12] A. A. Alomoush, A. A. Alsewari, H. S. Alamri, K. Aloufi, and K. Z. Zamli, "Hybrid harmony search algorithm with grey wolf optimizer and modified opposition-based learning," *IEEE Access*, vol. 7, pp. 68764–68785, 2019.

[13] A. K. Alazzawi, H. M. Rais, and S. Basri, "Parameters tuning of hybrid artificial bee colony search based strategy for t-way testing," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 5S, pp. 204–212, 2019.

[14] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, no. 9, pp. 1041–1054, Nov. 2021.

[15] Z. Sultan, R. Abbas, S. N. Bhatti, and S. A. A. Shah, "Analytical review on test cases prioritization techniques: An empirical study," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 2, pp. 293–302, 2017.
[16] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Jun. 2002.
[17] Y. Lei and K. C. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *Proc. 3rd IEEE Int. High-Assurance Syst. Eng. Symp.*, Nov. 1998, pp. 254–261.
[18] M. I. Younis and K. Z. Zamli, "MIPOG—An efficient t-way minimization strategy for combinatorial testing," *Int. J. Comput. Theory Eng.*, vol. 3, no. 3, pp. 388–397, 2011.
[19] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 437–444, Jul. 1997.
[20] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah, "Design and implementation of a t-way test data generation strategy with automated execution tool support," *Inf. Sci.*, vol. 181, no. 9, pp. 1741–1758, May 2011.
[21] B. Jenkins. (2003). *Jenny Tool*. Accessed: May 5, 2024. [Online]. Available: http://www.burtleburtle.net/bob/math
[22] A. Williams. (1996). *TConfig Tool*. Accessed: May 5, 2024. [Online]. Available: http://www.site.uottawa.ca/~awilliam
[23] A. Hartman, T. Klinger, and L. Raskin, "IBM intelligent test case handler," *Discrete Math.*, vol. 284, no. 1, pp. 149–156, 2010.
[24] J. Zhang, Z. Zhang, and F. Ma, *Automatic Generation of Combinatorial Test Data*. Cham, Switzerland: Springer, 2014.
[25] V. Rechtberger, M. Bures, B. S. Ahmed, Y. Belkhier, J. Nema, and H. Schvach, "Prioritized variable-length test cases generation for finite state machines," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2022, pp. 11–20.
[26] M. Ahmed, A. B. Nasser, and K. Z. Zamli, "Construction of prioritized T-way test suite using bi-objective dragonfly algorithm," *IEEE Access*, vol. 10, pp. 71683–71698, 2022.
[27] R. Michaels, M. K. Khan, and R. Bryce, "Test suite prioritization with element and event sequences for Android applications," in *Proc. IEEE 11th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2021, pp. 1326–1332.
[28] S. Kale and Y. S. S. R. Murthy, "Hybrid firefly algorithm based regression testcase prioritisation," *Int. J. Bus. Intell. Data Mining*, vol. 12, no. 4, p. 340, 2017.
[29] E.-H. Choi, S. Kawabata, O. Mizuno, C. Artho, and T. Kitamura, "Test effectiveness evaluation of prioritized combinatorial testing: A case study," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur. (QRS)*, Aug. 2016, pp. 61–68.
[30] M. University, S. Badanahatti, Y. Murthy, and M. University, "Optimal test case prioritization in cloud based regression testing with aid of KFCM," *Int. J. Intell. Eng. Syst.*, vol. 10, no. 3, pp. 96–105, Apr. 2017.
[31] S. Biswas, M. S. Kaiser, and S. A. Mamun, "Applying ant colony optimization in software testing to generate prioritized optimal path and test data," in *Proc. Int. Conf. Electr. Eng. Inf. Commun. Technol. (ICEEICT)*, May 2015, pp. 1–6.
[32] Y. Wang, H. Wu, and Z. Y. Sheng, "A method of testing generation based on prioritized pair," *Appl. Mech. Mater.*, vols. 241–244, pp. 2696–2700, Dec. 2012.
[33] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Oct. 2007, pp. 255–264.
[34] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*. New York, NY, USA: Springer Inc, 2003.
[35] 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.
[36] 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.
[37] M. B. Cohen, "Designing test suites for software interaction testing," Ph.D. dissertation, Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, 2004.

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 the author of many scientific papers published in renowned journals and conferences. His research interests include optimization algorithms, software testing, and artificial intelligence optimization, including software defect prediction and feature selection.

MD. ABDUL KADER (Member, IEEE) received the bachelor's degree in computer science and engineering from Khulna University of Science and Technology (KUET), Bangladesh, the M.Sc. degree in computer engineering from Universiti Malaysia Perlis (UniMAP), Malaysia, and the Ph.D. degree in computer science (computational intelligence) from Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA), Malaysia. He is currently a Senior Lecturer with the Faculty of Computing, UMPSA. He is the author of many scientific papers published in renowned journals and conferences. His research interests include computational intelligence, optimization algorithms, image processing, and software engineering.

ABDULRAHMAN A. ALSEWARI (Senior Member, IEEE) received the B.Eng. degree in computer engineering from the Military College of Engineering, Baghdad, Iraq, in 2002, and the M.Sc. degree in electronic system design engineering and the Ph.D. degree in software engineering from Universiti Sains Malaysia, in 2008 and 2012, respectively. He was an Associate Professor and a Researcher with the Software Engineering Department, Faculty of Computing, Universiti Malaysia Pahang, Malaysia, from 2013 to 2022. Currently, he is the Machine Learning Laboratory Leader, the Deputy course lead of computer science with AI, a Seminar Coordinator, and a Senior Lecturer with the College of Computing, Faculty of Computing, Engineering, and the Built Environment, Birmingham City University, U.K. He has authored or co-authored one book, more than 80 articles, and more than ten inventions. His research interests include software engineering, software testing, optimization algorithms, artificial intelligence, soft computing, embedded systems, and image processing.

SULIMAN MOHAMED FATI (Senior Member, IEEE) received the B.Sc. degree from Ain Shams University, Egypt, in 2002, the M.Sc. degree from Cairo University, Egypt, in 2009, and the Ph.D. degree from Universiti Sains Malaysia (USM), Malaysia, in 2014. He is currently an Assistant Professor and the Chairperson of the Information Systems Department, Prince Sultan University (PSU), Saudi Arabia. His research interests include the Internet of Things, machine learning, social media mining, cloud computing, cloud computing security, and information security.

NIBRAS ABDULLAH received the B.Sc. degree in computer science from Hadhramout University of Science and Technology, Yemen, in 2003, and the M.Sc. degree in computer science and the Ph.D. degree in multimedia networks protocols from Universiti Sains Malaysia, in 2010 and 2017, respectively. Currently, he is an Assistant Professor with the Faculty of Computer Studies, Arab Open University, Saudi Arabia. He has been a member of the Technical Program Committee for more than 20 international conferences worldwide. He has been appointed as a reviewer for various peer-reviewed international journals. His research interests include multimedia network protocols, computer networks, wireless networks, internet security, the Internet of Things (IoT), information and knowledge engineering, multimedia information systems, handwriting recognition, optimization, artificial intelligence, and artificial neural networks.

SAYAWU YAKUBU DIABA (Senior Member, IEEE) was born in Suhum, Ghana. He received the Bachelor of Engineering and Master of Science degrees in telecommunications engineering from Kwame Nkrumah University of Science and Technology, Kumasi, Ghana, and the Doctor of Science (Tech.) degree (Hons.) in telecommunication engineering from the University of Vaasa, Finland. He gained invaluable knowledge in managing and improving electrical systems throughout his 13 years of employment as a power distribution specialist with the Electricity Company of Ghana. He was employed at Finland's renowned VTT Technical Research Center as a Research Scientist. He is a renowned Engineer and a Researcher with strong academic and professional experience. He is also actively researching the possibilities of 6G and 5G technology. His research interests include the forefront of innovation and technology, such as the application of machine learning to smart grids, the creation of cybersecurity attack detection algorithms, and improvements to carbon neutrality and future energy systems.

WAHEB A. JABBAR (Senior Member, IEEE) received the B.Sc. degree (Hons.) in electrical engineering from the University of Basrah, Iraq, in 2001, and the Master of Engineering degree (Hons.) in communication and computer engineering and the Ph.D. degree in electrical, electronic, and systems engineering from Universiti Kebangsaan Malaysia (UKM), Malaysia, in 2011 and 2015, respectively. He is registered as a Charted Engineer (C.Eng.) under the Engineering Council and IET, U.K., and a Professional Technologist under Malaysia Board of Technologists (MBOT). He is currently an Associate Professor of electrical/electronic engineering with the College of Engineering, Faculty of Computing, Engineering, and the Built Environment, Birmingham City University (BCU), U.K. He is also the College Academic Lead (Subject Lead) of Electronic and Electrical Engineering and the Course Lead of Electronic Engineering. His research interests include the Internet of Things, cyber-physical systems, B5G/6G, wireless networks, and smart city. He also has a keen interest in low power wide area networks, such as LoRa networks and NB-IoT. He is an MIET member of IET U.K., and a member of IEM and BEM, Malaysia. He served as the Chair of the Educational Activities of the IEEE-UKM Student Branch, from 2012 to 2013.