IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 September 2022, accepted 10 October 2022, date of publication 14 October 2022, date of current version 25 October 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3214841*

RESEARCH ARTICLE

# Object-Oriented Test Case Generation Using Teaching Learning-Based Optimization (TLBO) Algorithm

**OHOOD AL-MASRI AND WEDAD AL-SORORI**
University of Science and Technology Sana'a, Sana'a, Yemen
Corresponding authors: Ohood Al-Masri (ohoodtariq23@gmail.com) and Wedad Al-Sorori (w.alsorori@gmail.com)

**ABSTRACT** Researchers are currently seeking effective methods for automated software testing to reduce time, avoid test case redundancy, and create comprehensive test cases to cover (paths, benches, conditions, and statements). Generating a minimum number of test cases and covering all code paths is challenging in automated test case generation. Therefore, the use of optimization algorithms has become a popular trend for generating test cases to achieve many goals. In this study, we used a teaching-learning-based optimization algorithm to generate the minimum number of test cases. We compared our results with those of other state-of-the-art methods based on the path coverage for ten Java programs. The motive for using this algorithm is to optimize the number of test cases that cover all code paths in the unit test. The results emphasize that the proposed algorithm generates the minimum number of test cases and covers all paths in the code at a full-coverage rate.

**INDEX TERMS** Test suite generation, unit testing, object-oriented test case generation, coverage-based optimization.

## I. INTRODUCTION

Software testing is an important phase in the software development process because it represents the quality of the software product [1], [2], [3], [4], [5], [6]. The primary goal of testing is to identify software flaws. This stage is considered even more important nowadays as programs are becoming increasingly complex, essential to safety, and extremely important in daily activities, thus requiring an increase in quality [7], [8], [9]. According to [10] and [11], testing accounts for more than 50% of the cost of software development. In software testing, two methodologies are often used: black box and white box [12]. The former is concerned with testing the functionality of the software under test without knowledge of the structure or implementation specifics. The latter is a technique for testing with knowledge of a program's core structure and coding. Therefore, testers must completely understand the source code and consider its behavior using testing coverage requirements (e.g., path coverage). The white box shows two types of testing, an integration test and a unit test. The integration test uses input and output file pairs to test the overall function of the software. Each integration test is specified in a configuration file with one line for each test. On the other hand, the unit test is the smallest testable portion of the software. The development team is responsible for this type of test. A developer performs this type of testing and must be well versed in the code design. Unit testing is the most basic sort of testing [9]. In this study, we focused on generating test cases for the unit testing. A unit testing approach can be either manual or automated. The former involves manually writing test cases and is more susceptible to human mistakes, whereas the latter involves using tools to perform test cases, depending on user input. Automated software testing can significantly lower the cost of software development. The goal of automated test case generation is to identify a suitable number of test cases to cover all conceivable targets (paths, statements, and branches) [13]. Path coverage tests all paths in the code. The major goal

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


This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

of the path-coverage criteria is to ensure that all paths are covered and to avoid redundant tests. The difficulty in generating data for a unit test can be viewed as an optimization problem that can be handled using a search-based software testing (SBST) approach [14], [15], [16], [17]. SBST aims to move software engineering problems from human-based to machine-based searches [18], [19]. The increase in SBST in recent years has been attributed to its significant contributions to the domain of software testing, such as reducing maintenance costs, prioritizing test cases, reducing human costs, verifying software models, and validating real-time properties. It also generates and minimizes the number of software tests using meta-heuristic searching algorithms, such as genetic algorithms, hill climbing, particle swarm optimization, and teaching-learning-based optimization) [11]. The teaching-learning-based optimization algorithm (TLBO) was inspired by the teaching-learning process [19]. An iterative learning algorithm has several characteristics compared to other evolutionary computation (EC) techniques. The algorithm simulates a teacher's and student's capacity to teach and learn in a classroom. The TLBO method has acquired widespread support among optimization experts because it does not require any specific parameters; it simply requires common regulating parameters, such as population size [19]. Therefore, it enhances the performance of the algorithm. The TLBO algorithm was used in the SBST in two studies. The first study Shahabi et al. [7] proposed TLBO for generating test cases. The algorithm was implemented in EvoSuite, which is a reference tool for search-based software testing. Empirical investigations on the SF110 dataset demonstrate that TLBO provides competitive results of 90.08% in method coverage when compared to standard and monotonic genetic algorithms. However, they did not evaluate the path coverage because EvoSuite did not adopt this criterion. The second study Kumar and Rajeev [20] suggested using TLBO for generating test cases based on branch coverage in procedural programming. Therefore, the motivation for this study is to implement TLBO in object-oriented programs to optimize the number of test cases based on the path coverage criterion. We then compared the results to other state-of-the-art methods. The paper is organized as follows: Section II describes the relevant past research in the field and provides an overall review of past literature. A preliminary description of the TLBO algorithm is in section III. Section IV describes the methodology of the proposed algorithm. Section V details the experimental results and compares other meta-heuristic algorithms. Section VI discusses the conclusion, as well as the theoretical and practical contributions of the study and suggestions for future work.

## II. RELATED WORK

The field of automated test data generation was first developed in the early 1970s. **Clarke's (1976)** [21] was considered the first research on automated test data generation. **Parther (1987)** [21] presented a new concept for test data generation called the path prefix method. **Korel (1990)** [21] provided a revolutionary change by dynamically generating test data. During the '90s, researchers focused on object-oriented programs, such as **Lakhotia et al.** [21]. Souza et al. [22] proposed a multi-objective optimization process based on particle swarm optimization (PSO) to optimize test case selection for functional tests. Moreover, various algorithms, such as (swarm intelligence and evolutionary algorithms, among other meta-heuristic methodologies) have been employed in the development of software test case generation. Khari et al.[23] created a tool that includes two primary automated software testing components: test-suite generation and test-suite optimization. Boundary value testing, robustness testing, worst-case testing, robust worst-case testing, and random testing are the five test suite-generating methods offered in the tool. The generated test suite was further optimized to the desired fitness level using the artificial bee colony algorithm or the cuckoo search algorithm. The two algorithms were applied to ten sample Java programs. The average value of the path coverage for ABC was 90.3% and that for CSA was 75.4%. Hamad [24] developed an artificial bee colony algorithm (ABC) to test data generation for software structural testing in two programs. The results demonstrate the success and ability of the ABC algorithm in software path testing by determining the optimal fitness values. Saber et al. [25] proposed a composing method: a greedy algorithm to quickly find good solutions, a genetic algorithm to increase the search space covered, and a local search algorithm to refine the solutions. The proposed method is 178% better than the state-of-the-art algorithms. Rani et al. [26] implemented an elitist genetic algorithm (GA) with an improved fitness function to expose maximum faults while also minimizing the cost of testing by generating fewer complex and asymmetric test cases. It uses a selective mutation strategy to create low-cost artificial faults that result in fewer redundant and equivalent mutants. This study used 14 Java programs of significant sizes to validate the efficacy of the proposed approach in comparison with initial random tests and a widely used evolutionary framework in academia, namely EvoSuite. The approach was a significant improvement in the test case optimization. Khari et al. [27] examined the performance of six meta-heuristic algorithms, including the hill-climbing algorithm (HCA), particle swarm optimization (PSO), firefly algorithm (FA), cuckoo search algorithm (CS), bat algorithm (BA), and artificial bee colony algorithm (ABC), using their standard implementation to optimize the path coverage and branch coverage produced by the test data. Each algorithm was implemented to generate test cases. Subsequently, the performance of each approach was evaluated for five Java programs. Process measures, such as average time, best time, and worst time, were used to compare the algorithms as well as product metrics, such as path coverage and objective function values of the resulting test suites. The BA was found to be the best-suited algorithm because it produced the most optimal test suites in the shortest amount of time, and the average coverage path of all five programs was approximately 80%. BA was

found to be the most rapid. FA was found to be the slowest algorithm. The CS, PSO, and HCA fall somewhere in the middle. Bidgoli and Haghighi [28] ant colony optimization (ACO) was adapted and improved to provide a test data generation strategy for covering prime paths. In comparison, test suites generated by an automatic tool can be used in a metaheuristic algorithm to generate test cases called EvoSuite. The results indicate that ACO had a 9% higher mutation score. Sharma et al. [29] developed a framework to optimize test cases using a cuckoo search algorithm. Geetha and Mala [14] proposed a tabu search hybrid to the BAT algorithm to choose test cases. The metric criterion for comparison is code coverage. The proposed BAT with TABU search yielded a 0.04875% improvement over the tabu algorithm. Anh [13] developed and enhanced a GA-based method for generating test cases for unit and integration testing. They implemented the algorithm in two classes for the unit test and in six classes for the integration test. The results showed that the GA obtained the highest coverage when compared to state-of-the-art algorithms based on branch coverage and execution time. Damia and Esnaashari [31] combined the firefly algorithm (FA) and the asexual reproduction optimization algorithm (ARO). FA is a bio-inspired algorithm that excels in exploitation and local searches but struggles with exploration and is prone to the local optima problem. On the other hand, ARO gets out of local optima. As a result, they have teamed up to incorporate ARO into the FA phases to boost population variety. This combination was used to generate automatic test cases for the six tested programs to cover all finite paths. FA-ARO achieves 100% path coverage when compared to the traditional genetic algorithm (TGA), adaptive genetic algorithm (AGA), adaptive particle swarm optimization (APSO), hybrid genetic tabu search algorithm (HGATS), random search (RS), differential evolution (DE), hybrid cuckoo search, and genetic search. Jaiswal and Prajapati [32] proposed a Particle Swarm Optimization (PSO) based test case selection approach for the basis path testing. They used the improved fitness function (IFF) as a fitness function that can direct the PSO-based optimization process toward optimal test case selection. They implemented the proposed algorithm using two programs. The results suggest that the proposed approach can generate better outcomes 100% in terms of control-flow graph coverage of all linearly independent paths than the traditional fitness function. Esnaashari and Damia [33] presented a structure for generating test cases based on path coverage. They proposed a mimetic algorithm that employs reinforcement learning as a local search approach within a genetic algorithm. Experiments have shown that this method generates test data faster than the standard genetic algorithm, various genetic algorithm upgrades, random search, particle swarm optimization, bee algorithm, ant colony optimization, simulated annealing, hill climbing, and tabu search. Furthermore, the algorithm provides 100 percent path coverage while requiring fewer evaluations. Lakshminarayana and Kumar [34] developed the cuckoo search and bee colony algorithm (CSBCA) to optimize the automated test cases. Using an example of the ATM withdrawal procedure. According to an experimental investigation, the proposed CSBCA technique produced path coverage in 16.4 seconds. The cuckoo search and bee colony algorithm (CSBCA) achieved a higher fitness function value of 0.7 to 1.0 in 65 percent of test cases/test data than the particular swarm optimization (PSO), cuckoo search algorithm (CS), firefly algorithm (FA), and bee colony algorithm (BCA). Sahoo and Ray [35] proposed a new approach, the forest optimization algorithm (FOA) with metamorphic relations (MRS), to cover multiple paths at a time in one run. The initial test case was created with FOA, and the subsequent test cases were created with metamorphic relations without going through many runs. The reason for utilizing the FOA is that its search process is similar to that of branch/path coverage techniques. The algorithm was implemented in MATLAB and its performance was assessed using six programs. The results show that FOA based on metamorphic relations is more efficient than particular swarm optimization based on metamorphic relations in terms of time consumption and the number of paths covered. For instance, FOA-based metamorphic relations cover five paths, whereas PSO-based metamorphic relations cover four paths. Gupta and Goyal [36] conducted a systematic review to generate test cases. The duration of the systematic review was 2010–2020. They presented all studies that showed the coverage standards, different datasets, and testing levels from 2010 to 2021. Path and branch coverage are common criteria used to generate test cases. Moreover, researchers have focused on the genetic algorithm (GA), some of which used particle swarm optimization (PSO). Moreover, they mentioned all the studies that presented hybrid algorithms. Furthermore, the results emphasize that only a few articles used hybrid algorithms and the ant colony algorithm (ACO), and they showed research that used manual and automatic processes of test case generation. The results indicated that only a few studies used manual test cases. By contrast, one study used both manual and automatic techniques. However, there is a need to reduce the time complexity of software testing and save money for automatic generation test cases.

## III. PRELIMINARY

Rao et al. developed the TLBO algorithm [19]. It is based on the teacher's influence on the effect of the student's output in a class (teaching-learning process). The algorithm shows two primary ways of learning: (1) learning from a teacher as the teacher phase, and (2) learning from other learners as the learner phase. In this optimization algorithm a group of learners is considered as a population and different subjects supplied to the learners are considered as different design variables of the optimization problem and a learner's result is analogous to the 'fitness' value of the optimization problem [30]. This method only requires general control parameters, such as population size and generation number, but no algorithm-specific control parameters are required. The instructor is considered the finest answer for the entire

**Initialize** the number of students and termination condition.
**While** (unmet termination condition)
    **Calculate** the mean of decision variables
    **Identify** the best solution as a teacher
    **Identify** the movement percentage
    **Modify** solution based on the best solution
    *Xnew = Xold+ r (Xteacher-(TF) Mean)*
**If** the new solution is better than existing
**Accept** the solution
    **Mutate** the solution
**Else**
    **Reject** the solution
**End If**
**Select** two solutions randomly: Xi and Xj
**If** Xi is better than Xj
*Xnew = Xold + r (Xi – Xj)*
**Else**
*Xnew = Xold + r (Xj – Xi)*
**End If**
**If** the new solution is better than the existing
    **Accept** the solution
    **Mutate** the solution
**Else**
    **Reject** the solution
**End If**
**End While**
**Return** the best solution

**FIGURE 1.** The discrete TLBO algorithm [7].

population. The design variables are the factors involved in the objective function of the given optimization problem, and the optimal value of the objective function is the optimum solution. Note: The teaching learning-based optimization (TLBO) algorithm has been presented to optimize continuous problems. However, in the research Shahabi [7], they adapt the algorithm to solve the discrete search problems. Therefore, we used a modified version of the algorithm that solves the discrete search problem as shown in Figure.1.

*1. Initialization:* The algorithm receives the number of individuals and termination conditions as inputs. The process begins with a randomly generated initial population.

*2. Teacher phase:* This is the first stage of the algorithm, in which students learn from the teacher. During this phase, the teacher seeks to enhance the class's average result in the subject based on his abilities. At any iteration, ı assumes that there are 'm' number of subjects and '*n*' number of learners. *Mji* is the mean result of the learners in a particular subject 'j' (j=1,2, .., m), and $kbest, i$ is the best learner. The best overall result X total $kjbest, i$ is the best learner that is calculated by adding all the subjects achieved over the full population of learners. However, because a teacher is often thought of as a highly educated person who trains students to get higher results, the algorithm considers the best learner identified as the instructor. The difference between each subject's existing mean result and the teacher's comparable result for each topic as:

$$Meanj, ki = ri(Xj, kbest, i - TFMji) \tag{1}$$

where, $Xj, kbest, i$ is the result of the best learner in the subject $j$ and $ri$ is a random number in the range [0, 1]. $TF$ is a teaching factor that determines the value of either one or two. The value is determined randomly with the same probability as:

$$TF = round\,[1 + rand\,(0, 1)\,2 - 1] \tag{2}$$

The $TF$ value range is between one and two. In the teacher's phase, the present answer is updated based on the $Difference\ Mean_{j,k,i}$

$$X'_{j,k,i} = X_{j,k,i} + Difference\ Mean_{j,k,i} \tag{3}$$

where, $X'_{j,k,i}$ is the updated value of $X_{j,k,i}$. $X'_{j,k,i}$ is accepted if it gives a better function value. The learner phase uses all the accepted function values from the teacher phase as input.

*3. Learner phase:* It is the second step of the algorithm, in which learners interact with one another to gather information. A learner connects with other learners at random to improve his or her knowledge. When another student has more knowledge than others do, the learner picks up new information.

*4. Termination:* At the end of every iteration, the entire population is evaluated, and if the minimum requirement (i.e., a specific coverage percentage) is found in a member, the algorithm ends. If there are no such members, the algorithm chooses the best individual as the teacher and continues into its evolutionary iterations. In addition, there is another stopping condition, such as a time limit.

## IV. METHODOLOGY

This section describes our proposed approach; more precisely, it describes our fitness function. We also describe the implementation of the proposed algorithm for generating and minimizing test cases.

### A. OVERVIEW

Search-based software testing is a random or directed search technique used to address problems in the software testing, verification, and validation domains. Figure.2 shows a search-based software test input generation approach.

Search-based techniques are becoming increasingly common in software testing and are particularly beneficial for generating test data [11]. In all SBST approaches, a suitable representation of the problem is first used to encode the solutions. The type of search operator employed in a search-based algorithm is influenced by the representation. A more significant aspect of the SBST approach is the development

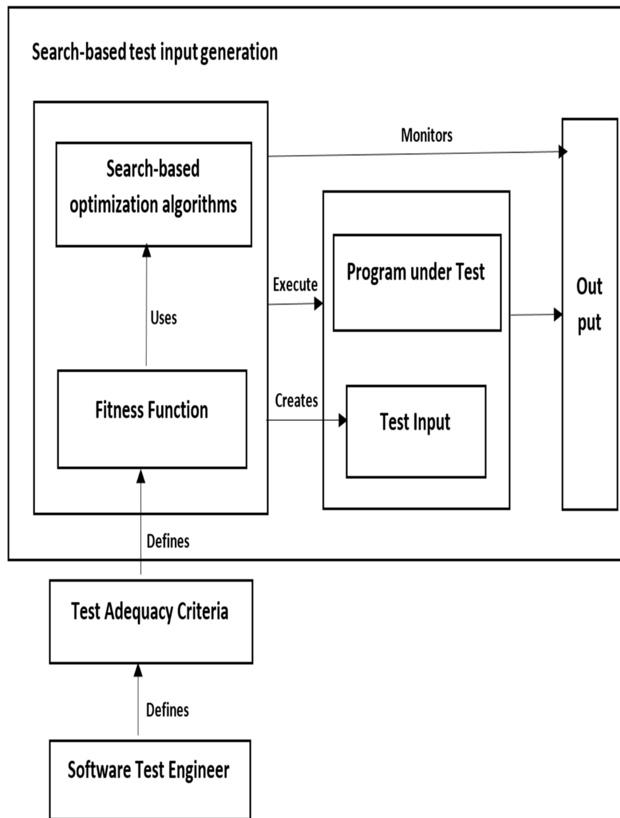


**FIGURE 2.** Search-based software test input generation method Khari and Kumar [11].

of a fitness function that evaluates the quality of the meta-solutions.

### B. DATASET

This study focuses on ten Java programs to generate the test cases. Table. 1 listed the ten programs, which were named P1 until P10. The programs range in length from approximately 10 to 75 lines of code (LOC). These programs differ in terms of programming paradigms and data types. P1 and P5 use nested if-else conditionals, while P2, P6, P7, P8, and P9 use basic if-else conditionals; P3 uses a switch case conditional, P4 is an if-else conditional, and P5 and P10 are nested loops.

### C. PERFORMANCE STUDY

In this subsection, we explain the steps of the methodology by using one of the tested programs P2 (the greatest number problem), as an example. The steps involved in the execution of the TLBO algorithm are shown in Figure.3.

#### 1) PREPARING CLASS UNDER TEST

Figure.4 shows the pseudocode for the greatest number program.

##### a: INDIVIDUAL ENCODER

A test case is constructed from a set of input values supplied to the program under test during execution. However, testing an object-oriented program requires additional information

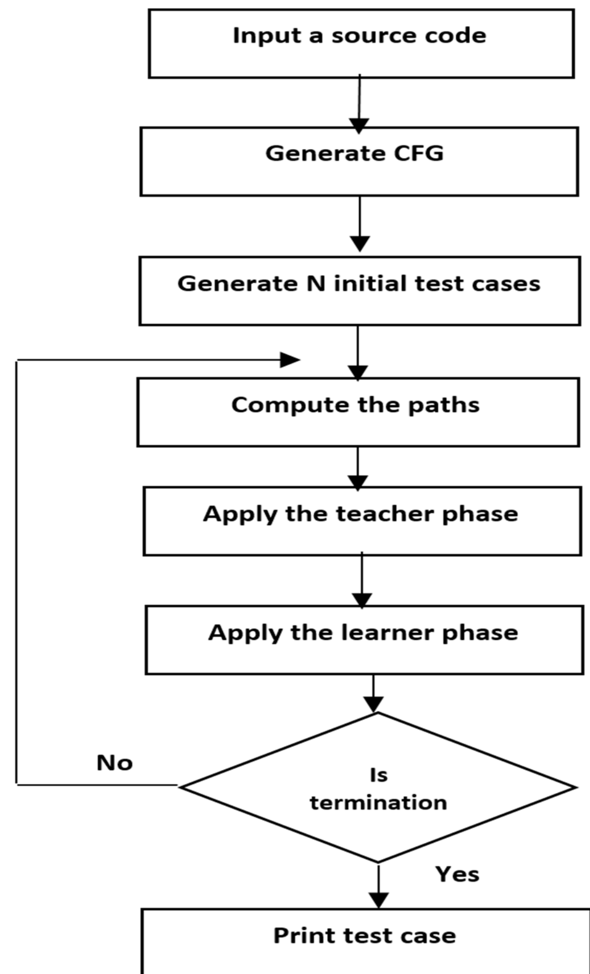


**FIGURE 3.** TLBO execution.

```
Begin
Int n1, n2, n3;
 IF (n1 > n2 && n1 > n3)
        Print ("n1 is the biggest")
Else If (n2 > n1 && n2 > n3)
        Print ("n2 is the biggest")
Else If (n3 > n1 && n3 > n1)
        Print ("n2 is the biggest")
Else
        Print ("They are equal")
End
```

**FIGURE 4.** Pseudocode for the greatest number program.

about the object's construction, the use of supplemental methods for attribute setting, and the values of the parameters supplied to these methods. A test case is defined as a collection of statements to be executed (separated by a colon) and the related parameter values (separated by a comma). Four types of statements in the encoding representation have

**TABLE 1.** Java programs under test.

| Label | Program | Description | LOC | Paths | Data Type | Variables |
|---|---|---|---|---|---|---|
| P1 | Triangle classification | It identifies the type of triangle. | 24 | 5 | Integer | 3 |
| P2 | Greatest number | It finds the largest number. | 18 | 4 | Integer | 3 |
| P3 | Days in a month | Given a month and a year. Moreover, it finds the number of days. | 75 | 13 | Integer | 2 |
| P4 | Prime number | It identifies if the given input is prime or not. | 25 | 4 | Integer | 1 |
| P5 | Bubble sort | The bubble sort algorithm. | 40 | 3 | Array | 2 |
| P6 | Even-Odd | It checks if a number input is even or odd. | 10 | 2 | Integer | 1 |
| P7 | Leap year | It checks if a given year is a leap year or not. | 10 | 2 | Integer | 1 |
| P8 | Quadratic-Equation | It checks if inputs can form a quadratic or not and finds it. | 15 | 4 | Double | 3 |
| P9 | Remainder | It checks whether the remainder is zero. After dividing the dividends and the divisor. | 10 | 3 | Double | 2 |
| P10 | Insertion sort | Insertion algorithm. | 34 | 5 | Array | 2 |

**TABLE 2.** Target path of P2.

| Path No. | Target path | | | | | |
|---|---|---|---|---|---|---|
| 1 | Start | 1 | 2 | 8 | End | |
| 2 | Start | 1 | 3 | 5 | 8 | End |
| 3 | Start | 1 | 4 | 6 | 8 | End |
| 4 | Start | 1 | 7 | 8 | End | |

been clarified [13]: The constructor is to generate instances of a given class. The field is to access public attributes of objects (identified as $obj); the method is to invoke methods on objects, and the assignment is to assign values to variables (indicated as $var) or public attributes of objects. Invocation parameters for methods and constructors can be an integer, boolean, string, double, or array. In the tested program P2, there were three integer inputs (n1, n2, and n3). The input values were selected randomly from the data type range until all target paths were passed.

### b: CODE INSTRUMENTATION

We generate a control flow graph (CFG) by using an abstract syntax tree (AST) as a copied file of the source code by the Junit and Open Java plugin in Eclipse. Figure.5 illustrates the control flow graph of tested program P2 as an example. Table. 2 shows all the paths of the tested program P2.

### 2) THE TLBO ALGORITHM IS USED TO GENERATE THE TEST CASES

### a: INITIAL POPULATION

The initial population of solutions is randomly produced from their data type domain. For instance, the size of the initial population of p2 is 100.

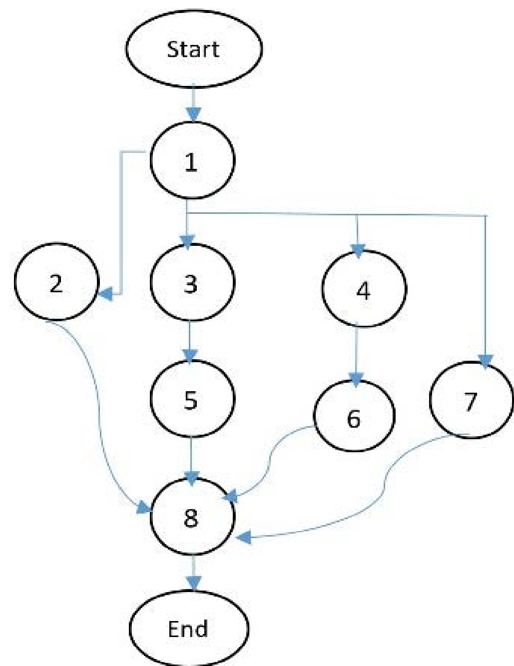


**FIGURE 5.** The CFG of P2.

### b: FITNESS FUNCTION

We used *Korel's fitness function* [12] to compute the distance between the target path and the considered path. Table. 3 shows the Korel function for P2.

Furthermore, the evaluation of the objective fitness value is conducted as

$$f = (w1 * f1 + w2 * f2) + K \quad (4)$$

**TABLE 3.** Korel function of greatest number program.

| Branch predicts | Branch distance |
|---|---|
| n1>n2 | n2-n1 |
| Max >n3 | n3-max |
| N3>max | max-n3 |

Equation 4 shows the linear secularization of the two functions f1 and f2 with weights w1 and w2 = one, and K is the offset. The value of K is 100. The two functions that are equivalent to the path and branch coverage are listed below in Equation 5 and Equation 6:

$$f1 = \frac{number\ of\ covered\ paths}{Total\ number\ of\ all\ paths} * 100 \quad (5)$$

$$f2 = \frac{Total\ branch\ covered}{Total\ branches\ in\ all\ test\ cases} * 100 \quad (6)$$

### c: THE TEACHER PHASE

Every path moves toward the teacher during the teaching phase. The average of the choice variables was determined for this purpose, and each path was updated as:

$$X_{new} = X_{old} + r\,(X_{teacher} - (T_F)\,Mean) \quad (7)$$

### d: LEARN PHASE

Each participant was assigned a classmate at random, and their fitness levels were compared. In reality, in the teaching phase, the path is coupled with the instructor in the movement operator, which means that it will become more like the teacher throughout that phase. If the fitness value of a random classmate is greater, the path travels toward it using the following equation:

$$X_{new} = X_{old} + r\left(X_i - X_j\right) \quad (8)$$

If the randomly chosen classmate has a lower score, the path moves away from it and closer to the teacher as:

$$X_{new} = X_{old} + r\left(X_j - X_i\right) \quad (9)$$

### e: TERMINATION

In our example P2, the entire population is examined at the end of each iteration, and if a member meets the target path, the algorithm ends. There is another stopping condition, which is that the time limit equals 1000000 seconds.

## V. RESULT

In this section, we present the results of implementing the TLBO-based test data generator. Then, we compared the proposed algorithm results with those of other state-of-the-art algorithms.

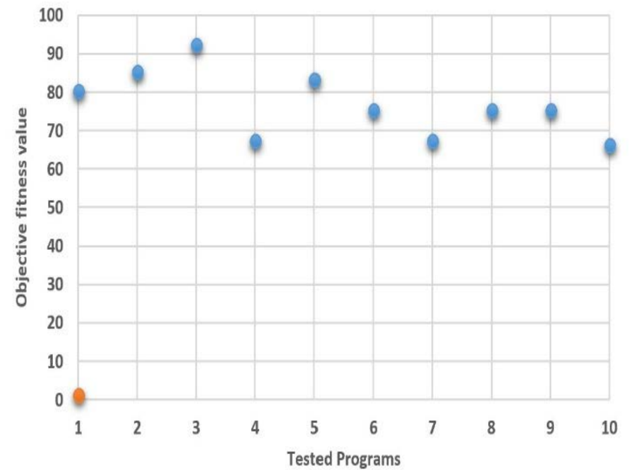


**FIGURE 6.** The objective fitness value.

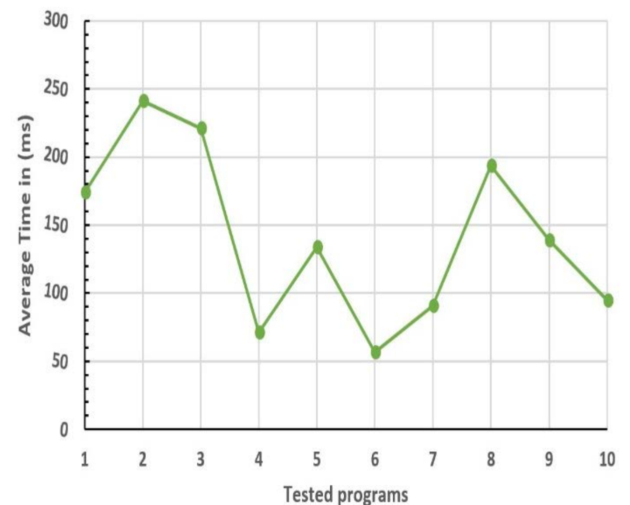


**FIGURE 7.** Average execution time of tested programs.

### A. EXPERIMENT RESULTS

We implemented the algorithm for ten Java programs to generate test cases. The results emphasize that the TLBO algorithm generates test cases with full 100% path coverage. The objective fitness value for P1 is 80, P2 is 85, P3 is 92, P4 is 67, P5 is 83, P6 is 75, P7 is 67, P8 is 75, P9 is 75, and P10 is 66, as illustrated in Figure.6. As observed from the results, P3 obtains the highest objective value of 92 because it has 13 paths to be visited.

Furthermore, Millie Second measures the execution time for the ten tested programs. The results of time consumption for generating test cases are (P1 = 174.3 (ms), P2 = 241.1 (ms), P3 = 221.2 (ms), P4 = 71.3 (ms), P5 = 133.8 (ms), P6 = 56.7 (ms), P7 = 90.8 (ms), P8 = 193.8 (ms), P9 = 138.6 (ms), and P10 = 94.6 (ms)). As it might be clear from Figure.7, the TLBO algorithm presents great results in terms of time complexity in the programs

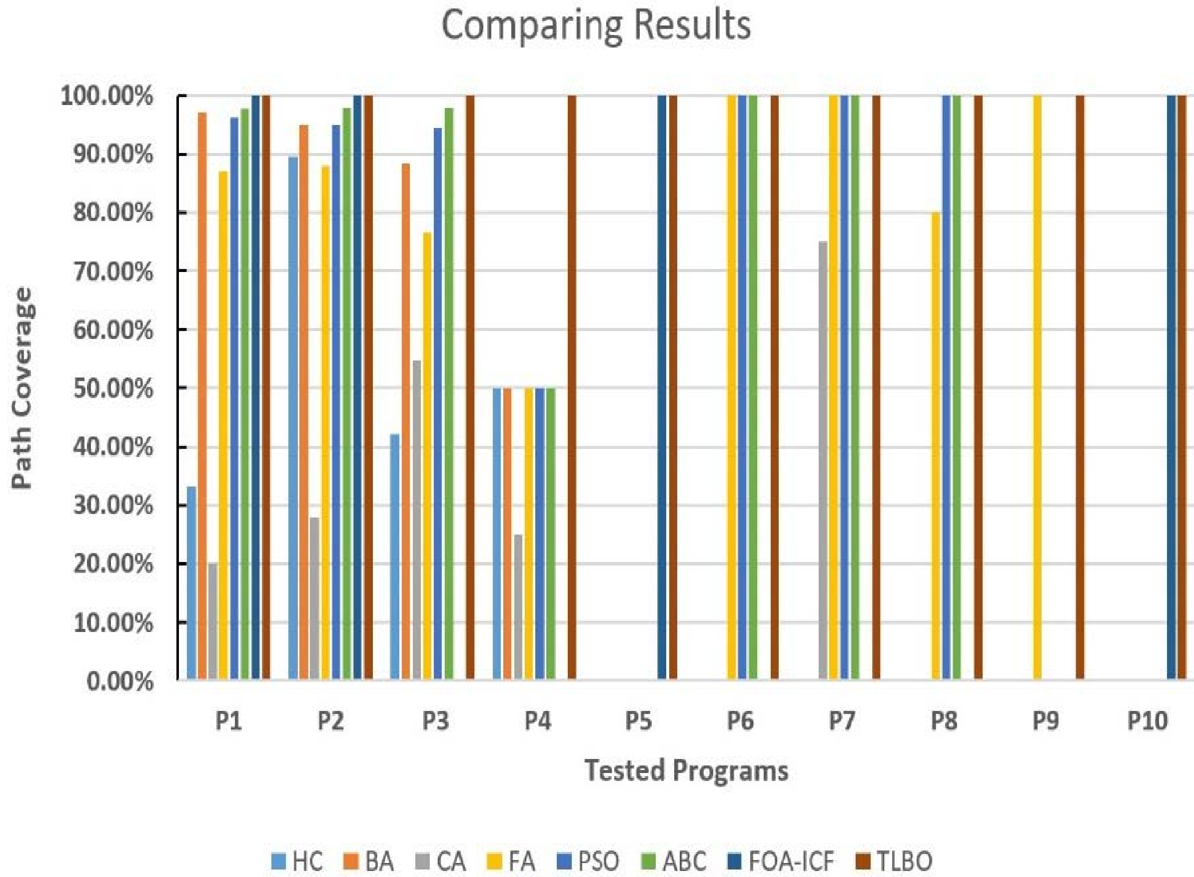


**FIGURE 8.** Compared results with the state of the art.

under test. For example, the execution time of P6 was 75.0 (ms). If we converted 75.0 (ms) to seconds, it would be 0.750 (s). This means less than one second. Therefore, obtaining test cases and covering code paths in less than one second has proven the efficiency of the proposed algorithm. Table. 4 lists the objective fitness value and the execution time for each program.

### B. COMPARED THE OUTCOMES TO THOSE OF OTHER ALGORITHMS

Regarding the research results Sahoo and Ray [35]; Khari et al. [27]; Khari and Kumar [15]), we compared the results of hill climbing (HC), bat algorithm (BA), cuckoo algorithm (CS), firefly algorithm (FA), particle swarm algorithm (PSO), artificial bee colony algorithm (ABC), forest optimization with improved combined fitness (FOA-ICF), and TLBO based on the path coverage criteria. The global parameters of all compared algorithms are illustrated in Table. 5.

Table. 6 and Figure.8 show the results of compared algorithms based on path coverage criteria. The result of P1 indicates that TLBO achieves full coverage of paths; however, the CS obtains the lowest percentage of path coverage (20.0%). The result of P2 shows that TLBO obtains the highest result with full coverage of 100%; however, the CA gets the lowest percentage of path coverage (20.0%). P3 result indicates that TLBO obtains the highest result with full

**TABLE 4.** Objective fitness and the time execution.

| Label | Objective fitness value | Time execution |
|---|---|---|
| P1 | 80 | 174.3 |
| P2 | 85 | 241.1 |
| P3 | 92 | 221.2 |
| P4 | 67 | 71.3 |
| P5 | 83 | 133.8 |
| P6 | 75 | 56.7 |
| P7 | 67 | 90.8 |
| P8 | 75 | 193.8 |
| P9 | 75 | 138.6 |
| P10 | 66 | 94.6 |

**TABLE 5.** Compared parameters.

| Algorithm | Lower bound | Upper bound | Population size | Condition stop |
|---|---|---|---|---|
| HC | The range of data types | The range of data types | 100 | The number of paths |
| BA | The range of data types | The range of data types | 100 | The number of paths |
| CS | The range of data types | The range of data types | 10 (random choice) | The number of paths |
| FA | The range of data types. | The range of data types | 100 | The number of paths |
| PSO | The range of data types | The range of data types | 100 | The number of paths |
| ABC | The range of data types | The range of data types | 10 (random choice) | The number of paths |
| FOA-ICF | 1 | 1000 | 100 | The number of paths |
| TLBO | The range of data types | The range of data types | 100 | The number of paths or time limit equals 1000000 (*S*). |

coverage; in contrast, the HC obtains the lowest percentage of path coverage (42.15%). P4 result emphasizes that TLBO obtains full coverage of 100%; nevertheless, the CS obtains the lowest percentage of path coverage (25.0%). P5 and P10 results illustrate that TLBO gets the same result with full coverage of paths 100% to the FOA-ICF. The P6 result shows that TLBO, PSO, ABC, and FA achieve full coverage. The P7 result indicates that TLBO, PSO, ABC, and FA achieve the full coverage of paths 100%; however, CS obtains the lowest percentage of path coverage (75.0%). The result of P8 indicates that TLBO, PSO, and ABC obtain the highest result with full coverage of paths 100%; on the contrary, the FA obtains the lowest percentage of path coverage of 80.0%. The P9 result indicates that TLBO and the FA obtain the same result with full coverage of paths 100%. The 10 result indicates that TLBO and the FOA-ICF obtain the same result with full coverage of paths 100%.

## VI. CONCLUSION

The purpose of this study is to generate test cases in a unit test for object-oriented programming. We implemented the generation of test cases in ten Java programs and covered all code paths using the TLBO algorithm. Therefore, the results of the practical experiments indicate that the TLBO algorithm obtains the minimum number of test cases and full path coverage of 100% for all tested programs when compared to hill climbing, the bat algorithm, the cuckoo algorithm, the firefly algorithm, the particle swarm algorithm, and the artificial bee colony algorithm. However, the forest optimization algorithm achieved the same full coverage of 100%. This study focused on object-oriented programs. The proposed method can also be used in procedural programming. The proposed approach is adapted to generate test cases for unit testing. However, it can also be applied to integration testing of multiple classes.

**TABLE 6.** Comparison based on path coverage.

| Label | Program Name | HC | BA | CS | FA | PSO | ABC | FOA-ICF | TLBO |
|---|---|---|---|---|---|---|---|---|---|
| P1 | Triangle classification | 33.2 | 97 | 20.0 | 87 | 96.2 | 97.8 | 100 | 100 |
| P2 | Greatest number | 89.5 | 95 | 28.0 | 88 | 95.0 | 98.0 | 100 | 100 |
| P3 | Days in a month | 42.1 | 88.2 | 54.7 | 76.6 | 94.4 | 97.9 | - | 100 |
| P4 | Prime number | 50 | 50 | 25 | 50 | 50 | 50 | - | 100 |
| P5 | Bubble Sort | - | - | - | - | - | - | 100 | 100 |
| P6 | Even-Odd | - | - | - | 100 | 100 | 100 | - | 100 |
| P7 | Leap Year | - | - | - | 75 | 100 | 100 | - | 100 |
| P8 | Quadratic Equation | | - | - | 80 | 100 | 100 | - | 100 |
| P9 | Remainder | - | - | - | 100 | - | - | - | 100 |
| P10 | Insertion Sort | - | - | - | - | - | - | 100 | 100 |

## REFERENCES


[1] H. V. Gamido and M. V. Gamido, "Comparative review of the features of automated software testing tools," *Int. J. Electr. Comput. Eng.*, vol. 9, no. 5, pp. 4473–4478, Oct. 2019.

[2] X. Yao, D. Gong, B. Li, X. Dang, and G. Zhang, "Testing method for software with randomness using genetic algorithm," *IEEE Access*, vol. 8, pp. 61999–62010, 2020, doi: 10.1109/ACCESS.2020.2983762.

[3] O. Sahin, B. Akay, and D. Karaboga, "Archive-based multi-criteria artificial bee colony algorithm for whole test suite generation," *Eng. Sci. Technol., Int. J.*, vol. 24, no. 3, pp. 806–817, Jun. 2021, doi: 10.1016/j.jestch.2020.12.011.

[4] D. Bruce, H. D. Menéndez, E. T. Barr, and D. Clark, "Ant colony optimization for object-oriented unit test generation," in *Proc. Int. Conf. Swarm Intell.* Cham, Switzerland: Springer, Oct. 2020, pp. 29–41, doi: 10.1145/3287324.3287502.

[5] S. Sheoran, N. Mittal, and A. Gelbukh, "Artificial bee colony algorithm in data flow testing for optimal test suite generation," *Int. J. Syst. Assurance Eng. Manage.*, vol. 11, no. 2, pp. 340–349, Apr. 2020.

[6] A. K. Alazzawi, H. M. rais, and S. Basri, "ABCVS: An artificial bee colony for generating variable T-way test sets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 4, pp. 106–111, 2019.

[7] M. M. Shahabi, S. E. Beheshtian, S. P. Badiei, S. M. Moosavi, and R. Akbari, "EVOTLBO: A TLBO based method for automatic test data generation in EvoSuite," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 6, pp. 214–226, 2017, doi: 10.14569/IJACSA.2017.080627.

[8] K. Hrabovská, B. Rossi, and T. Pitner, "Software testing process model benefits and drawbacks: A systematic literature review," 2019, *arXiv:1901.01450*.

[9] N. Jatana and B. Suri, "An improved crow search algorithm for test data generation using search-based mutation testing," *Neural Process. Lett.*, vol. 52, pp. 767–784, Jun. 2020, doi: 10.1007/s11063-020-10288-7.

[10] R. B. Bahaweres, K. Zawawi, D. Khairani, and N. Hakiem, "Analysis of statement branch and loop coverage in software testing with genetic algorithm," in *Proc. 4th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, Sep. 2017, pp. 1–6, doi: 10.1109/EECSI.2017.8239088.

[11] M. Khari and P. Kumar, "An extensive evaluation of search-based software testing: A review," *Soft Comput.*, vol. 23, no. 6, pp. 1933–1946, Mar. 2019, doi: 10.1007/s00500-017-2906-y.

[12] Y. Shin, Y. Choi, and W. J. Lee, "Integration testing through reusing representative unit test cases for high-confidence medical software," *Comput. Biol. Med.*, vol. 43, no. 5, pp. 434–443, Jun. 2013, doi: 10.1016/j.compbiomed.2013.01.024.

[13] B. T. M. Anh, "Enhanced genetic algorithm for automatic generation of unit and integration test suite," in *Proc. RIVF Int. Conf. Comput. Commun. Technol. (RIVF)*, Oct. 2020, pp. 1–6, doi: 10.1109/RIVF48685.2020.9140778.

[14] B. Geetha and D. J. Mala, "A hybrid bat approach with Tabu search algorithm for test case selection in object-oriented testing," *ICTACT J. Soft Comput.*, vol. 11, no. 1, pp. 1–5, 2020, doi: 10.21917/ijsc.2020.0318.

[15] M. Khari and P. Kumar, "An effective meta-heuristic cuckoo search algorithm for test suite optimization," *Formatica*, vol. 41, no. 3, pp. 363–377, 2017.

[16] M. Panda, P. P. Sarangi, and S. Dash, "Automatic test data generation using meta-heuristic cuckoo search algorithm," *Int. J. Knowledge Discovery Bioinf.*, vol. 5, no. 2, pp. 71–79, 2017, doi: 10.4018/IJKDB.2015070102.

[17] R. Sharma and A. Saha, "Optimization of object-oriented testing using firefly algorithm," *J. Inf. Optim. Sci.*, vol. 38, no. 6, pp. 873–893, Aug. 2017, doi: 10.1080/02522667.2017.1372135.

[18] H. Peng, C. Deng, and Z. Wu, "Best neighbor-guided artificial bee colony algorithm for continuous optimization problems," *Soft Comput.*, vol. 23, no. 18, pp. 8723–8740, Sep. 2019, doi: 10.1007/s00500-018-3473-6.

[19] R. V. Rao, V. J. Savsani, and D. P. Vakharia, "Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems," *Comput.-Aided Des.*, vol. 43, no. 3, pp. 303–315, Mar. 2011, doi: 10.1016/j.cad.2010.12.015.

[20] M. Kumar and R. T. Rajeev, "Automated test case generation based on coverage using teaching learning based optimization," Tech. Rep., 2014.

[21] R. Malhotra and M. Khari, "Heuristic search-based approach for automated test data generation: A survey," *Int. J. Bio-Inspired Comput.*, vol. 5, no. 1, p. 1, 2013.

[22] L. S. D. Souza, P. B. C. D. Miranda, R. B. C. Prudencio, and F. D. A. Barros, "A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort," in *Proc. IEEE 23rd Int. Conf. Tools Artif. Intell.*, Nov. 2011, pp. 245–252, doi: 10.1109/ICTAI.2011.45.

[23] M. Khari, P. Kumar, D. Burgos, and R. G. Crespo, "Optimized test suites for automated testing using different optimization techniques," *Soft Comput.*, vol. 22, no. 24, pp. 8341–8352, Dec. 2018, doi: 10.1007/s00500-017-2780-7.

[24] F. Hamad, "Using an artificial bee colony algorithm for test data generation and path testing coverage," pp. 99–112, 2018, doi: 10.5539/mas.v12n7p99.

[25] T. Saber, F. Delavernhe, M. Papadakis, M. O'Neill, and A. Ventresque, "A hybrid algorithm for multi-objective test case selection," Tech. Rep., 2018.

[26] S. Rani, B. Suri, and R. Goyal, "On the effectiveness of using elitist genetic algorithm in mutation testing," *Symmetry*, vol. 11, no. 9, p. 1145, Sep. 2019.

[27] M. Khari, A. Sinha, E. Verd, and R. G. Crespo, "Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization," *Soft Comput.*, vol. 24, no. 12, pp. 9143–9160, Jun. 2020.

[28] A. Monemi Bidgoli and H. Haghighi, "Augmenting ant colony optimization with adaptive random testing to cover prime paths," *J. Syst. Softw.*, vol. 161, Mar. 2020, Art. no. 110495.

[29] S. Sharma, S. A. M. Rizvi, and V. Sharma, "A framework for optimization of software test cases generation using cuckoo search algorithm," in *Proc. 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2019, pp. 282–286, doi: 10.1109/CONFLUENCE.2019.8776898.

[30] M.-F. Leung, C. A. C. Coello, C.-C. Cheung, S.-C. Ng, and A. K.-F. Lui, "A hybrid leader selection strategy for many-objective particle swarm optimization," *IEEE Access*, vol. 8, 2020, pp. 189527–189545, doi: 10.1109/ACCESS.2020.3031002.

[31] A. H. Damia and M. M. Esnaashari, "Automated test data generation using a combination of firefly algorithm and asexual reproduction optimization algorithm," *Int. J. Web Res.*, vol. 3, no. 1, pp. 1–10, 2020.

[32] U. Jaiswal and A. Prajapati, "Optimized test case generation for basis path testing using improved fitness function with PSO," in *Proc. 13th Int. Conf. Contemp. Comput. (IC)*, Aug. 2021, pp. 475–483, doi: 10.1145/3474124.3474197.

[33] M. Esnaashari and A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning," *Exp. Syst. Appl.*, vol. 183, Nov. 2021, Art. no. 115446, doi: 10.1016/j.eswa.2021.115446.

[34] P. Lakshminarayana and T. V. S. Kumar, "Automatic generation and optimization of test case using hybrid cuckoo search and bee colony algorithm," *J. Intell. Syst.*, vol. 30, no. 1, pp. 59–72, Jul. 2020, doi: 10.1515/jisys-2019-0051.

[35] R. R. Sahoo and M. Ray, "Forest optimization-based test case generation for multiple paths with metamorphic relations," *Int. J. Appl. Metaheuristic Comput.*, vol. 13, no. 1, pp. 1–18, Jan. 2022, doi: 10.4018/IJAMC.292503.

[36] K. Gupta and P. Goyal, "Systematic study on test case generation: Thoughts and drifts with future perceptions," *J. Harbin Inst. Technol.*, vol. 54, no. 2, pp. 1–15, 2022.


**OHOOD AL-MASRI** received the B.Sc. degree in software engineering from the University of Science and Technology Sana'a (UST), Sana'a, Yemen, where she is currently pursuing the master's degree in software engineering. She works as an Instructor at UST Sana'a. Her current research interests include software engineering, artificial intelligence, and software testing.

**WEDAD AL-SORORI** received the B.Sc. degree in computer science from the Department of Computer Science, Faculty of Science, Sana'a University, Sana'a, Yemen, in 2008, the M.Sc. degree in computer information sciences from Arab Academy for Banking and Financial Sciences (AABFS), in 2014, and the Ph.D. degree in computing from the Department of Computer Science, Faculty of Computing and IT (CIT), University of Science and Technology Sana'a (UST), Sana'a, in 2020. She is currently an Assistant Professor with CIT, UST Sana'a. She has been the Academic Supervisor for the faculty in the girls' branch, since 2015, and a member of several academic committees at CIT. She has published several papers in journals and conference proceedings. Her research and teaching interests include the areas of computational and artificial intelligence, optimization, data science, and approximate algorithms (meta-heuristic algorithms).

• • •