IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE RELIABILITY SOCIETY SECTION

Received 23 July 2022, accepted 7 August 2022, date of publication 16 August 2022, date of current version 19 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3198694

RESEARCH ARTICLE

Automatic Test Case Generation Using Many-Objective Search and Principal Component Analysis

DONGCHENG LI[1], W. ERIC WONG[1], SEAN PAN[2], LIANG-SENG KOH[2], SHENGLONG LI[3], AND MATTHEW CHAU[1]

[1]Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080, USA
[2]RFCyber Corporation, Frisco, TX 75033, USA
[3]School of Computer Science, China University of Geosciences, Wuhan 430074, China

Corresponding author: W. Eric Wong (ewong@utdallas.edu)

This work was supported in part by the U.S. National Science Foundation under Grant 1757828, Grant 1822137, and Grant 2050869; and in part by RFCyber Corporation.

ABSTRACT Test case generation in essence is a many-objective optimization problem, with objectives such as program statements and branches. Currently, the DynaMOSA algorithm built on the EvoSuite framework simultaneously searches multiple objectives and eventually produces a test suite with high coverage for multiple testing criteria. However, its performance is undesirable in searching test cases when there are an excessive number of objectives. As the software structure becomes increasingly complex and the lines of program code increase, the high cost of testing limits the application of search-based automatic test case generation technology for software testing. To generate test cases with high coverage within a limited time, this paper proposes PCA-DynaMOSA to overcome the shortcomings of DynaMOSA, and the improvement in the proposed algorithm is due to dimensionality reduction. To carry out the experiments, 49 projects or 110 classes were selected from the SF110 benchmarking dataset according to the complexity and the number of objectives of the classes under test. The experimental results indicate that PCA-DynaMOSA outperforms DynaMOSA in generating test cases on most projects in terms of line, branch, mutation, and multi-criteria coverage. Moreover, it achieves higher or equivalent coverage and offers improved test case generation performance.

INDEX TERMS Test case generation, many-objective optimization, EvoSuite, DynaMOSA, PCA.

I. INTRODUCTION

Traditional software testing relies heavily on the manual work of testers [1], [2]. In particular, it requires a series of inputs from the tester to define software functions and examines the validity of the corresponding outputs, while the internal logic of the software remains unknown. Consequently, such testing is highly subjective and inaccurate, and unstable. This kind of software testing has a long cycle and incurs high costs, accounting for a large proportion of the investment in software research and development [3], [4]. Automated software testing is therefore of great significance for the development of software technology, as it has the potential to considerably shorten software research-and-development cycles, lower expenditures, increase user confidence, and improve software reliability [5]. A complete test suite should cover all scenario in codes to the greatest extent when the program is running. In this way, software functions can be sufficiently validated, reducing the flaws of manual testing [6].

In recent years, applications of bio-inspired heuristic search algorithms in NP-hard optimization problems have been widely investigated [7]. Compared with random approaches, heuristic search algorithms show strong abilities in searching for optimal solutions. Through continuous evolution and guidance by fitness functions, global and local searches can be conducted in a solution space to find

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Steven Li.

an optimal solution. In the field of testing automation, the automatic test case generation technique based on heuristic search algorithms has also attracted extensive attention. This technique evaluates the quality of test cases with two factors: (1) coverage and (2) the size of the test case. A complete test case should cover as many scenario existing in the program code as possible, and an excessively long test case may raise the subsequent workload of test cases. In essence, the search for test cases with optimal coverage while maintaining a reasonable size of those test cases is an optimization problem. The combination of the automatic test case generation with the heuristic search algorithm is apparent. In this study, test case generation is transformed into a many-objective optimization problem. The branches, statements, and mutations of code in the software program were selected as the coverage objectives. The aim is to generate test cases with high coverage.

In a many-objective optimization problem with no more than three objectives, the traditional Pareto-based many-objective sorting algorithm can be effective. Regarding the problem of test case generation, there are dozens or hundreds of coverage objectives in a single class, and so it is less likely that Pareto-based techniques would produce optimal solutions [3]. In this study, an improved many-objective optimization algorithm for test case generation based on dimensionality reduction is proposed. It is capable of decomposing objectives in great quantity and preserving the dominant objectives. In this way, the performance of the algorithm in generating high-coverage test cases is improved.

Principal component analysis (PCA) was used to perform dimensionality reduction for excessive objectives in a test case generation problem. PCA decompose a large numbers of objectives into a small number of objectives that contain the essential information. Thus, test cases with high coverage are generated efficiently.

The rest of the paper is organized as follows. In Section 2, the studies related to search-based test case generation are presented. Section 3 introduces the test case generation problem as a many-objective optimization problem. Section 4 describes the proposed many-objective search algorithm with dimensionality reduction for the automatic generation of test cases. In Section 5, the experiment setup is described and the results analyzed. Finally, the conclusions and recommendations for future research are given in Section 6.

II. RELATED STUDIES

As heuristic search algorithms are applied and validated in more fields, researchers have begun to apply them to automatic test case generation. The whole suite method was introduced and built on the EvoSuite framework [8]. Fraser *et al.* converted the problem of test case generation into a many-objective optimization problem with a weighted-sum method that obtains its total fitness value by summing the fitness of all coverage objectives. This method resolves many problems in single-objective search-based test case generation, including dependency among objective branches, different searching difficulties of different branches, and search resource waste caused by repetitive test cases covering the same objective. Zhang and Li [9] proposed an automatic test case generation technique using a many-objective longicorn beetle search algorithm. A total path similarity was designed from the perspective of multiple paths for the target path group of program code. On this basis, test data capable of simultaneously satisfying the requirements of multiple target paths were generated, thus reducing iterations and shortening the time required. Sahin *et al.* [10] proposed a full test suite generation method using the archive-based multi-criteria artificial bee colony algorithm. It unifies fitness functions of diverse coverage criteria to simultaneously optimize multi-criteria objectives. In addition, the efficiency of the test case generation algorithm was improved using the archive-based method. During many-objective optimization, a weighted-sum method, the same as used in the whole suite method, was adopted to simultaneously optimize the covering objectives.

Although weighted-sum methods such as whole suite can be more effective than the traditional ways to search a single objective, the convergence to the optimal solution was difficult for test case generation in non-convex search space [11]. Based on the whole suite method, Panichella *et al.* proposed a many-objective genetic algorithm with preference ordering, called MOSA [12]. For test case generation, the goal of the search algorithm was not to achieve population diversity but to obtain individuals with fitness equal to zero. Thus, the strategy of preferring individuals having fitness values close to zero was reasonable. Yao [13] combined an infeasible path detection method based on conditional statement correlations with the improved many-objective genetic algorithm to generate test cases. Such a method ensures testing adequacy and reduces redundant test cases.

The major issue of the search-based many-objective automatic test case generation techniques is low efficiency in generating test cases with high coverage caused by excessive search objectives. It is therefore important to improve the search capability of test case generation algorithms in the context of an excessive number of objectives. Panichella *et al.* [14] proposed DynaMOSA to improve the search performance of MOSA by dynamically selecting targets to be covered. Many covering objectives exist in the classes under test, and they are dependent on each other. To be specific, for two adjacent branches in the same method of the same class under test, one branch cannot be reached until the previous branch has been executed, which is decided by the execution sequence from top to bottom of the program code. Other objectives depend on the covering objectives (branches) in the dominant position. Thus, in the DynaMOSA algorithm, only covering objectives in the dominant position are searched in a control dependency graph of the class under test. The objectives can be removed from this graph only when they are covered. By dynamically selecting covering objectives, the number of objectives to be searched by the many-objective algorithm can be reduced to improve

the search performance of the algorithm. Subsequently, a multi-criteria strategy built on DynaMOSA was proposed by Panichella *et al.* [15]. Covering objectives of diverse coverage criteria can be optimized at the same time, and, in the optimization process, a set of objectives from different coverage criteria (branch coverage, line coverage, etc.) can be searched simultaneously. In this way, test cases with higher overall coverage can be generated. Recently, search-based test case generation has been improved for its non-functional attributes. For example, aDynaMOSA [16] was proposed by Grano *et al.*, with both execution time and memory consumption of test cases as additional objectives to be optimized by the algorithm.

Currently, automatic test case generation based on many-objective optimization is still a novel research direction. Relevant studies are scarce, and in most of these studies, multiple objectives are aggregated into a single objective for search. In this field, DynaMOSA proposed by Panichella is the most advanced search-based many-objective optimization algorithm for automatic test case generation. Although the dynamic selection of objectives leads to a reduction in the number of objectives for search, more objectives need to be searched as the program code becomes increasingly complex, resulting in a decline in the search performance of the algorithm. Further study is therefore required to determine how to improve the search performance of many-objective optimization algorithms for test case generation in the context of excessive coverage objectives.

III. TEST CASE GENERATION PROBLEM

The test case generation problem is to determine a test suite composed of a set of unit tests for the program under test. Each test case includes a series of different test statements. Running results produced by the unit tests invoking the program are passed to a test oracle to verify the correctness of the outputs. In automatic white-box test case generation, however, no test oracle is provided. Thus, it cannot be determined whether the running output is consistent with the actual output based on the function of the program unless manual adjustments by programmers are made [17]. The key to automatic test case generation is therefore to generate the fewest tests and produce test cases with the maximum testing coverage.

The generation of test cases needs to satisfy two conditions. First, the number of test statements in a test case should be small. During software testing, the test oracle needs to be manually created for test cases. If the test cases are lengthy, redundant, and hard to understand, the manual cost will be increased during the creation of the test oracle, and the testing cycle will also be extended. Thus, the fewer the test statements included in a test case, the better the generated test suite will be overall. Second, the maximum testing coverage of the objectives should be obtained. For a test case generation problem, the testing coverage is raised for the purpose of evaluating whether the current test case has been used to sufficiently test the class under test. A test suite with a higher testing coverage can be used to test the program more robustly. Some important concepts related to the test case generation problem are described below.

A. CONTROL FLOW GRAPH

The control flow graph (CFG) describes the execution procedures of a program. It depicts all paths that may be traversed during program execution. The calculations of various coverage objectives are formulated with such a graph.

B. TEST SUITE AND TEST CASES

A test suite is a set of test cases generated for the entire program, and each test case is formed by method call sequences with variable lengths. For test case generation with a single-objective search method, a test case is generated for each objective, and all generated test cases are combined to form a test suite for testing the entire program. With regard to test case generation with a many-objective search method, the test cases are no longer searched for a single objective; rather, all objectives that should be covered are simultaneously searched, and the test case eventually obtained may cover single or multiple objectives.

C. OBJECTIVES TO BE COVERED

The adequacy of the test cases for a given program can be described with various coverage criteria, such as line coverage, branch coverage, mutation coverage, method coverage, etc. The coverage criteria used in this study are line, branch, and mutation coverage.

D. BRANCH COVERAGE

"Branch" refers to the decision statements (e.g., *if* and *while*) in a program. In the CFG, they are nodes with at least two outgoing edges. The outgoing edges of branch nodes are branches. Therefore, a branch node may possess multiple branches. The coverage of a test case t to a branch can be represented by the following fitness formula:

$$f_B(t) = al(b_i, t) + d(b_i, t) \tag{1}$$

where $al(b_i, t)$ stands for the distance from the branch executed by the test case t to the objective branch to be covered. It is equal to the number of control dependencies or edges between branch nodes in the CFG. Moreover, $d(b_i, t)$ is the branch distance, which can be calculated by the following formula:

$$d(b,t) = \begin{cases} 0 & \text{if branch has been covered} \\ \frac{D_{\min}(t \in T, b)}{D_{\min}(t \in T, b)+1} & \text{if current branch has been excuted twice by the test case} \\ 1 & \text{other} \end{cases} \tag{2}$$

where $D_{\min}(t \in T, b)$ is the minimized non-normalized distance. It is executed twice, because a decision statement has two possible situations. The shortest branch distance is selected from the execution.

E. LINE COVERAGE

Each node in the CFG of a program is the bytecode of a particular line. The line coverage of test case t can be expressed in the following fitness formula:

$$f_{\mathrm{L}}(t) = al(s_i, t) + d(b(s_i), t) \tag{3}$$

where $al(s_i, t)$ is the distance between statements; $d(b(s_i), t)$ refers to the distance from the current branch executed by the test case t to the branch closest to the current statement.

F. MUTATION COVERAGE

Mutants are obtained by applying a set of mutation operators or introducing minor changes or human errors to the source code. In other words, they are mutated programs obtained by modifying a node via a selected mutation operator in the CFG of the source program. The mutation operators used in this study are *ReplaceVariable*, *InsertUnaryOperator*, *ReplaceConstant*, and *ReplaceArithmeticOperator*. By executing the test case in mutated and original programs, different execution results can be compared. Based on the results, one can determine whether the mutants are killed. The mutation coverage of test case t is represented by the following fitness formula:

$$f_{\mathrm{M}}(t) = al(m_i, t) + d(b(m_i), t) + d_i(m_i, t) + d_p(m_i, t) \tag{4}$$

where $al(m_i, t)$ is the distance from the statement node executed by the test case t to the mutated node; $d(b(m_i), t)$ represents the normalized branch distance of the mutated node; and $d_i(m_i, t)$ and $d_p(m_i, t)$ stand for the infection status distance and transmission distance, respectively.

G. MULTI-OBJECTIVE TEST CASE GENERATION

In test case generation, two conditions are taken into account: the fewest statements in a test case and the maximum coverage for all objectives. Moreover, various coverage criteria can be utilized to improve the overall testing adequacy. Each coverage criterion contains many concrete optimization objectives. Therefore, the test case generation problem can be transformed into a many-objective optimization problem.

If the program under test has a total m objectives to be covered, comprising s_B objectives for branch coverage, s_L objectives for line coverage, and s_M objectives for mutation coverage, then $m = s_B + s_L + s_M$. For a test suite $T = \{t_1, t_2, \cdots, t_n\}$, the included test cases are variable. The goal is to minimize the fitness value of all m objectives:

$$\begin{cases} \min f_{B,i}(\mathrm{t})(i = 1, 2, \cdots, s_B) \\ \min f_{L,j}(t)(j = 1, 2, \cdots, s_L) \\ \min f_{M,k}(t)(k = 1, 2, \cdots, s_M) \end{cases} \tag{5}$$

IV. MANY-OBJECTIVE TEST CASE GENERATION ALGORITHM USING PRINCIPAL COMPONENT ANALYSIS

For the multi-criteria many-objective test case generation problem, there exist dozens or even hundreds of coverage objectives in each class under test, which poses a significant challenge to the search performance of the algorithm. DynaMOSA is a many-objective search-based test case generation algorithm based on NSGA-II (non-dominated sorting genetic algorithm II) for generating test cases that achieve high coverage for multiple testing criteria while maintaining a good search performance in the context of excessive objectives. In DynaMOSA, new preference criteria are raised for test case generation. More weight is put into the test cases that are closer to the covering objectives, which greatly improves the efficiency of the test case generation. The main advantages of DynaMOSA are the novel preference ordering method and the dynamic selection of objectives to be optimized. However, the performance of the DynaMOSA algorithm rapidly declines as the number of optimization objectives increases. One reason is that the traditional non-dominated sorting method is used to acquire dominated solution sets. Although such an issue can be alleviated to some degree by preference ordering and dynamic selection of objectives, it fails to improve the performance of the Pareto-based sorting (including inadequate selection pressure and a low convergence rate) under the condition of excessive objectives.

A. DIMENSIONALITY REDUCTION FOR TEST CASE GENERATION

The aim of the dimensionality reduction algorithm is to reduce a large dataset into a small dataset based on certain dimensionality-reduction criteria through analytical calculations. For many-objective optimization, it is difficult to search using traditional Pareto-based algorithms such as NSGA-II due to the excessive number of objectives that need to be optimized. The comparison among individuals can be difficult. Even with many computing resources, no favorable result set can be obtained. A dimensionality-reduction algorithm can be adopted to reduce the dimensionality of the problem space and reduce the optimization objectives to a manageable number, which can improve the problem-solving and optimization ability of the search algorithm and thus the efficiency of the test case generation. The feasibility of applying a dimensionality-reduction method to the search-based automated test case generation was analyzed from the following two aspects:

1) MULTIPLE COVERAGE OBJECTIVES ARE COVERED BY THE SAME TEST CASE

In a test case generation problem, a test case contains multiple method call sequences. When these method call sequences are executed in the program under test, they may cover multiple coverage objectives (e.g., branches, lines) in various program paths of the tested program. Reducing or eliminating such objectives may not cause important optimization information to be lost.

2) CORRELATIONS EXIST AMONG DIFFERENT COVERAGE OBJECTIVES

The difference between a test case generation problem and a traditional optimization problem is that, for test case

generation, the objectives to be optimized are statements or branches in a program, and there exists a semantic relationship between these statements or branches, such as the inclusion or dependency relationships between two decision statements.

Thus, dimensionality reduction may be appropriate for these objectives. There are also some tradeoffs with the dimensionality-reduction method, such as the computing resources and time consumed by the calculation. In selecting and configuring a specific dimensionality-reduction algorithm, major consideration is given to the time consumption of the algorithm and the number of objectives after dimensionality reduction.

B. PRINCIPAL COMPONENT ANALYSIS ALGORITHM

PCA [18] is an unsupervised dimensionality reduction algorithm. It is mainly intended for analyzing the coverage objectives to be optimized based on the fitness of the existing objectives. On the premise of no information loss, it eliminates objectives in a large objective set to lower the dimensionality of the test case generation problem. Its advantages can be described as follows:

- Eliminates repetitive feature information in the objective space;
- Reduces complexity and improves search performance of the algorithm; and
- Preserve critical coverage objective.

In PCA, dimensionality reduction of objectives is realized by transforming data in the original objective space into a new objective space. The dimensions of the objective space represent the number of objectives. The selection of vector directions with high variances as the directions of the coordinate axis is preferred. The greater the variance, the more important the information contained in the data is. Thus, variances that satisfy relevant conditions are sorted before each selection. First, the vector direction of the maximum variance is selected to be the first coordinate axis direction. Next, the vector direction that is orthogonal to the first coordinate axis and that has the largest variance among the remaining vectors is selected to be the next coordinate axis direction. By repeating the step above, a new coordinate space can be constructed. In the course of selection, dimensions with variances equal to zero are neglected. As a result, dimensionality reduction of the objective space is achieved. The main steps of PCA are shown in Figure 1.

C. MAIN PROCEDURES OF PRINCIPAL COMPONENT ANALYSIS

To generate test cases, PCA was adopted to perform dimensionality reduction for a set of objectives, removing repetitive feature information and keeping the important components. Furthermore, the set of reduced objectives was used to carry out the many-objective sorting of the individuals in the population to guide the search process of the algorithm. In this way, the problem of performance decline in DynaMOSA in the context of excessive objectives can be alleviated.

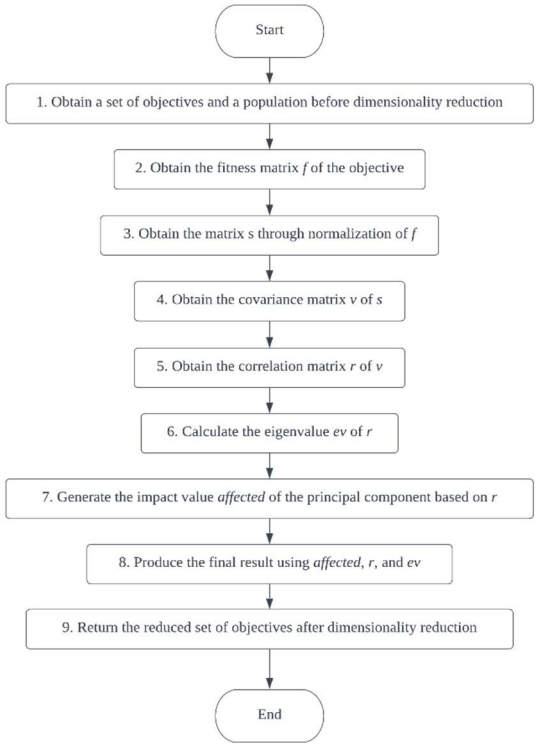

FIGURE 1. Flow Chart of PCA.

Considering that test case generation involves many objectives, dimensionality reduction of all objectives at once may cause convergence failures, large time consumption, or poor final results of the dimensionality-reduction algorithm. For this reason, in our proposed algorithm, objectives were grouped, with each group consisting of a certain number of objectives to dynamically reduce the corresponding dimensionality. The final set of objectives used to guide the many-objective search was obtained through the aggregation of all reduced sets of objectives. The specific number of groups used in our algorithm was determined by the experiments. The proposed many-objective test case generation algorithm using PCA, called PCA-DynaMOSA, is presented in Figure 2 and Algorithm 1.

V. EXPERIMENTAL DESIGN AND RESULTS ANALYSIS

To evaluate the performance of the proposed algorithm, different experiments were designed and conducted. The results of the experiments are analyzed and compared with the performance of DynaMOSA.

A. EXPERIMENTAL DESIGN

The experiments were primarily aimed at answering the following research questions (RQs) for multi-criteria many-objective test case generation:

RQ1. How does group size affect the search performance of the PCA-DynaMOSA algorithm? How can the specific number of groups be determined for the algorithm?

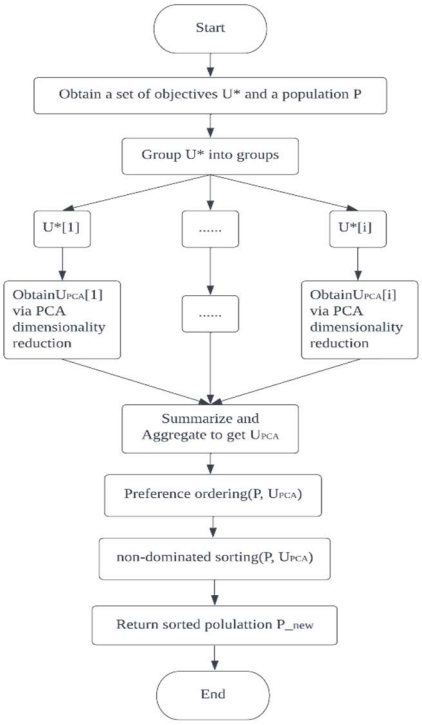

FIGURE 2. Flow chart of PCA for many-objective test case generation.

Algorithm 1 Many-Objective Test Case Generation Algorithm Based on PCA.

Input: Population P, Population size S, Target set M
Output: A test suite T
$U^* =$ target in M with not control dependencies
$t = 0$
$P_t =$ RANDOM-POPULATION(S)
archive = UPDATE-ARCHIVE(P_t)
U^* = UPDATE-TARGETS(U^*,G)
while not (Search_budget_consumed) do
  U∗[i] = group(U∗)
  U_{PCA}[i] = PCA.run(U∗)
  U_{PCA}= Summarize(U_{PCA}[i])
  PREFERENCE-SORT(U_{PCA})
  CROWDING-DISTANCE-ASSIGNMENT(F_d,U_{PCA})
end while
T <- archive

RQ2. How does the PCA-DynaMOSA algorithm perform in comparison with DynaMOSA in terms of line coverage, branch coverage, and mutation coverage of the generated test cases?
RQ3. How does the PCA-DynaMOSA algorithm perform in comparison with DynaMOSA in terms of multi-criteria total coverage of the generated test cases and search efficiency of the algorithm? What is the relationship between the algorithm's search efficiency and the multi-criteria total coverage of the generated test cases?

1) EVALUATION METRICS

The following evaluation metrics were selected to test and compare the test case search capabilities of both the original and the improved algorithms.

a: TESTING COVERAGE

This metric represents the adequacy of the final test suite in testing the program. It includes the sub-metrics line coverage, branch coverage, mutation coverage, and multi-criteria coverage. Moreover, the multi-criteria coverage is the coverage of all objectives (the aggregation of all branches, all lines, and all mutants) for test case generation. The metrics for calculating those coverages are as follows:

$$\begin{cases} statement\_coverage = \dfrac{\text{Number of covered statements}}{\text{Total number of statements to be covered}} \\ \times 100\% \\ branch\_coverage = \dfrac{\text{Number of covered branches}}{\text{Total number of branches to be covered}} \\ \times 100\% \\ mutation\_coverage = \dfrac{\text{Number of killed mutants}}{\text{Total number of mutants to be killed}} \\ \times 100\% \\ multicriteria\_coverage \\ = \dfrac{\text{Number of covered objectives}}{\text{Total number of objectives to be covered}} \times 100\% \end{cases} \tag{6}$$

b: SEARCH EFFICIENCY

Using the coverage percentage collected every second throughout the test case generation, the search efficiency of an algorithm can be calculated by the following equation:

$$AUC = \frac{\sum_{i=0}^{120} [cov_i + cov_{i+1}] * \Delta Time}{2 * TotalTime} \tag{7}$$

where cov_i refers to the percentage of coverage at time i, and cov_{i+1} refers to the percentage of coverage at time $i + 1$; $\Delta Time$ represents the time interval, which is 1 second; *TotalTime* is the total running time of an algorithm, which is 120 seconds.

The increase in testing coverage from the original algorithm to the improved algorithm can be calculated using the following equation:

$$Increase\_rate = \frac{\text{Value}_{\text{after}} - \text{Value}_{brfore}}{\text{Value}_{brfore}} \times 100\% \tag{8}$$

In this study, a proportional change of no more than 0.5% is considered an insignificant change or no change. Therefore, projects or classes with an increase rate below 0.5% are removed when comparing the coverage and search efficiency of the algorithms.

2) EXPERIMENT SETUP

The proposed algorithm was implemented in the EvoSuite framework and compared with DynaMOSA. The experiments were conducted on the Windows platform. The setup for the experimental environment is shown in Table 1.

TABLE 1. Experimental environment.

Hardware	Intel(R) Core(TM) i5-6500 CPU@ 3.20GHz
	RAM 12GB
Software	Windows10
	Java 1.8

3) DATASETS

First, CKJM was used to analyze the complexity of all projects in the SF110 dataset [19]. Then, projects with high complexity were selected to verify the performance of the proposed algorithm.

The SF110 dataset contain 110 statistically representative open-source Java projects collected from SourceForge. SourceForge is a popular open-source repository with over 300,000 projects and more than 2 million registered users. It includes all types of projects. Thus, SF110 is appropriate for testing the performance of the test case generation algorithms and frameworks. However, due to the large number of projects and classes in the datasets, experiments using all of the classes in the datasets may take a long time and incur high costs. More than 50% of the classes in the SF110 datasets have low program complexity, which means a simple method call is capable of covering most of the objectives. CKJM [20], [21], a program that calculates several object-oriented metrics (e.g., weighted methods per class, depth of inheritance tree, coupling between object classes, lack of cohesion in methods, response for a class, and the number of children) by processing the bytecode of compiled Java files, was adopted to analyze the complexity of all classes in the SF110 datasets.

The dataset used in the experiments were selected by the following procedure. First, the code complexity of all classes in the SF110 dataset was calculated using the CKJM tool, and the number of objectives covered in each class was obtained using the EvoSuite framework. Next, classes under test were sorted according to their complexity and number of objectives covered. Projects containing more complex classes with a greater number of objectives were selected and included in the final test dataset. Considering very few projects or classes behave abnormally during the experiment, such projects or classes were excluded from the experiment. The projects and the number of classes selected from each project are presented in Table 2.

Table 2 shows that 49 of the 110 projects in the SF110 datasets were selected. In total, 110 classes served as the fundamental test data for the experiments.

To compare the performance of PCA-DynaMOSA with DynaMOSA, the experiments were repeated 10 times for each test class in the selected datasets. The running results were averaged for comparison and analysis.

4) PARAMETER SETTINGS

a: PARAMETER SETTINGS FOR THE PROPOSED ALGORITHM

The default parameters for the DynaMOSA algorithm in EvoSuite yield a reasonably high performance, which has been empirically validated by other scholars. To preserve the quality of DynaMOSA and to ensure a controlled setting for the experiments, the same parameter settings (shown in Table 3) were adopted for our proposed algorithm.

TABLE 2. List of projects and classes under test.

Project name	Classes	Project name	Classes
gaj	1	biff	1
sfmis	1	lavalamp	2
imsmart	2	jhandballmoves	3
jdbacl	1	hft-bomberman	2
omjstate	1	dom4j	5
beanbin	1	openjms	2
inspirento	3	gae-app-manager	1
jsecurity	5	biblestudy	1
nekomud	1	lhamacaw	5
geo-google	3	ext4j	1
jni-inchi	1	fim1	2
gangup	4	fixsuite	1
apbsmem	7	twfbplayer	2
bpmail	1	wheelwebtool	7
xisemele	1	javathena	1
corina	5	xbus	2
schemaspy	3	ifx-framework	1
petsoar	1	classviewer	2
diffi	2	quickserver	1
glengineer	6	heal	3
follow	2	feudalismgame	1
lilith	3	liferay	2
lotus	1	pdfsam	1
resources4j	1	firebird	3
diebierse	1	-	-

TABLE 3. Parameter settings for the many-objective optimization algorithm.

Parameter	Value
Population size	100
Search time	120000 seconds (2 minutes)
Covering objectives	Total number of lines, branches, and mutants to be covered
Crossover rate	0.75

b: GROUP SETTINGS FOR PCA-DynaMOSA

To select an optimal number of groups in PCA-DynaMOSA, the number of groups for the experiments was established as follows (see Table 4).

B. EXPERIMENTAL RESULTS AND ANALYSIS

1) PERFORMANCE OF PCA-DYNAMOSA UNDER DIFFERENT GROUP SETTINGS (RQ 1)

To determine what group setting of PCA-DynaMOSA produces the optimal performance in terms of multi-criteria coverage, 18 classes were randomly selected from the test dataset. Their mean coverage was obtained for analysis and to produce the corresponding curves, as presented in Table 5 and Figure 3.

Table 5 and Figure 3 show that different numbers of objectives in a PCA group have a clear influence on the

TABLE 4. Group settings for PCA-DynaMOSA.

Group No.	Group size
PCA5	5
PCA10	10
PCA20	20
PCA30	30
PCA40	40
PCA50	50
PCA60	60

TABLE 5. Statistics of different numbers of PCA groups*.

Group No.	Group size	Mean multi-criteria coverage	Mean AUC
PCA5	5	82.37%	79.81%
PCA10	10	82.87%	80.47%
PCA20	20	83.61%	80.86%
PCA30	30	82.89%	80.24%
PCA40	40	82.34%	77.96%
PCA50	50	83.71%	80.15%
PCA60	60	81.38%	78.71%
DynaMOSA	-	**82.07%**	**79.44%**

*PCA5 represents a version of the PCA-DynaMOSA algorithm that groups every 5 objectives of the total objectives together for dimensionality reduction.

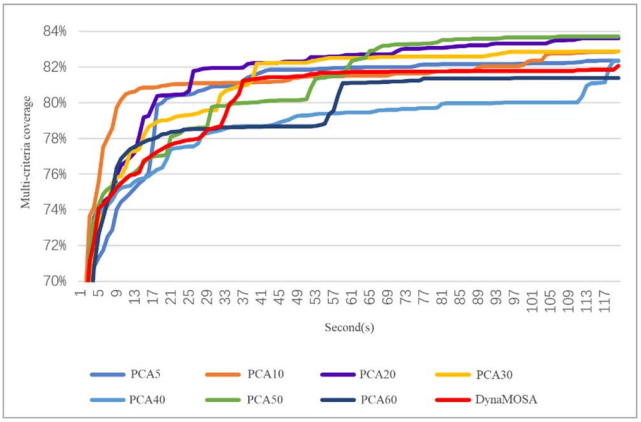

FIGURE 3. Search Performance Curves Generated by Different Numbers of PCA Groups.

performance of the test case generation. Too many or too few objectives in each group may lead to poor performance of the algorithm. For example, the search performance of PCA60 and PCA5 show no substantial improvements compared with DynaMOSA. If the number of objectives is too small in each group, the final set of objectives used to guide the search will barely change. Additionally, computing resources will have been spent on dimensionality reduction. In contrast, if the number of objectives is too large in each group, it becomes difficult for the dimensionality reduction algorithm to converge, which may lead to a significant increase in the time consumed by the algorithm. However, with appropriate parameter settings, PCA can be used to improve DynaMOSA and significantly raise its search efficiency and test coverage.

Among the data shown in Table 5, the mean coverage and the mean area under the curve (AUC) value, excluding those in PCA60 and PCA40, are improved compared with those of the DynaMOSA algorithm. Moreover, the optimal overall results are generated by PCA20 and PCA50. More specifically, PCA20 provides the most satisfactory search performance, and PCA50 achieves the highest final coverage ratio. The search performance of PCA20 is 0.71% higher than that of PCA50, but the final coverage ratio of PCA50 is just 0.1% greater than that of PCA20. According to the search performance curves, PCA20 (purple) and PCA50 (green) perform best among the different groupings. However, the search performance of PCA50 is no better than the original DynaMOSA in the first 60 seconds. On the other hand, the search performance of PCA20 is consistently better than DynaMOSA. Thus, 20 was selected as the number of objectives in each group for PCA-DynaMOSA.

2) COVERAGE ACHIEVED FOR EACH PROJECT (RQS 2 AND 3)

Tables 6 to 8 does not include projects that have the same coverage result achieved by the algorithms under analysis. Results that vary by no more than 0.5% are denoted by "-". In addition, the numbers in blue represent results with significant increases, while those in red represent results with significant decreases.

a: LINE COVERAGE ACHIEVED FOR EACH PROJECT

The line (statement) coverages of DynaMOSA and PCA-DynaMOSA are shown in Table 6:

TABLE 6. Mean line coverage achieved for each project.

Project	PCA-DYNAMOSA Statement coverage	DYNAMOSA Statement coverage	PCA-DYNAMOSA versus DYNAMOSA
imsmart	90.67%	84.14%	**7.75%**
inspirento	92.61%	91.18%	**1.57%**
jsecurity	92.58%	92.12%	**0.50%**
apbsmem	90.18%	89.60%	**0.65%**
corina	77.82%	78.04%	-
diffi	92.98%	92.07%	**0.99%**
follow	94.01%	81.11%	**15.91%**
resources4j	93.94%	75.76%	**24.00%**
diebierse	97.06%	100.00%	**-2.94%**
dom4j	85.62%	85.04%	**0.69%**
openjms	87.98%	85.78%	**2.56%**
lhamacaw	64.77%	63.02%	**2.78%**
twfbplayer	98.14%	97.33%	**0.83%**
wheelwebtool	59.30%	57.46%	**3.19%**
xbus	85.21%	83.04%	**2.62%**
quickserver	96.53%	95.83%	**0.72%**
heal	72.68%	72.49%	-
liferay	75.79%	74.91%	**1.18%**
firebird	94.05%	91.52%	**2.76%**
Mean over all projects	86.42%	83.71%	**3.87%**

b: BRANCH COVERAGE ACHIEVED FOR EACH PROJECT

The branch coverages of DynaMOSA and PCA-DynaMOSA are shown in Table 7.

TABLE 7. Mean branch coverage achieved for each project.

Project	PCA-DYNAMOSA Branch coverage	DYNAMOSA Branch coverage	PCA-DYNAMOSA versus DYNAMOSA
imsmart	86.11%	76.11%	13.14%
inspirento	96.45%	94.86%	1.67%
jsecurity	90.64%	89.72%	1.02%
apbsmem	90.89%	90.62%	-
xisemele	80.00%	77.14%	3.70%
corina	70.83%	69.79%	1.48%
diffi	92.86%	88.80%	4.57%
follow	91.98%	66.98%	37.32%
resources4j	91.89%	70.27%	30.77%
diebierse	80.00%	90.00%	-11.11%
hft-bomberman	98.53%	97.06%	1.52%
dom4j	82.59%	82.20%	-
openjms	77.42%	76.04%	1.81%
lhamacaw	79.04%	77.32%	2.22%
twfbplayer	91.41%	88.65%	3.11%
wheelwebtool	69.25%	67.75%	2.22%
xbus	83.89%	81.32%	3.15%
classviewer	86.37%	87.14%	-0.89%
quickserver	95.00%	93.33%	1.79%
heal	79.51%	79.32%	-
liferay	70.26%	68.84%	2.06%
firebird	85.88%	81.80%	4.98%
Mean over all projects	85.04%	81.59%	**5.50%**

c: MUTATION COVERAGE ACHIEVED FOR EACH PROJECT

The mutation coverages of DynaMOSA and PCA-DynaMOSA are shown in Table 8:

TABLE 8. Mean mutation coverage achieved for each project.

Project	PCA-DYNAMOSA Mutation coverage	DYNAMOSA Mutation coverage	PCA-DYNAMOSA versus DYNAMOSA
imsmart	86.60%	73.33%	18.09%
inspirento	93.42%	91.16%	2.48%
jsecurity	89.34%	88.93%	-
apbsmem	95.12%	95.04%	-
xisemele	83.82%	82.35%	1.79%
corina	77.71%	76.78%	1.21%
schemaspy	69.74%	68.62%	1.64%
diffi	89.65%	86.48%	3.67%
glengineer	89.42%	89.29%	-
follow	87.17%	71.14%	22.54%
lilith	76.77%	75.98%	1.05%
diebierse	98.42%	100.00%	-1.58%
jhandballmoves	99.10%	98.75%	-
dom4j	80.48%	79.73%	0.94%
openjms	83.21%	82.48%	0.88%
biblestudy	96.15%	96.15%	-
lhamacaw	63.08%	60.20%	4.78%
ext4j	100.00%	97.87%	2.17%
fim1	94.69%	95.58%	-0.93%
twfbplayer	92.48%	91.61%	0.95%
wheelwebtool	63.98%	61.49%	4.05%
xbus	82.61%	79.25%	4.24%
quickserver	88.30%	87.47%	0.96%
heal	81.44%	80.11%	1.65%
liferay	72.97%	69.93%	4.35%
Mean over all projects	85.43%	83.19%	**3.75%**

Tables 6 to 8 show that the coverage achieved by PCA-DynaMOSA improves on that of DynaMOSA. The proposed algorithm achieves equivalent or higher coverage on most of the projects; coverage is lower on only a few projects. In comparison with DynaMOSA, the mean coverage of PCA-DynaMOSA increased by 3.87% for line, 5.50% for branch, and 3.75% for mutation.

TABLE 9. Number of projects with better and worse results in multiple coverage criteria.

Algorithm	Coverage criteria	Better	Worse
PCA-DYNAMOSA versus DYNAMOSA	Statement coverage	16 (32.65%)	1 (2.04%)
	Branch coverage	17 (34.69%)	2 (4.08%)
	Mutation coverage	18 (36.73%)	2 (4.08%)

Table 9 shows that, compared with DynaMOSA, the coverage achieved by our approach is on average significantly higher in 32.65% of the classes under test for line, 34.69% for branch, and 36.73% for mutation. The improvement in the mutation coverage is the greatest among the three coverage criteria. In addition, a decrease in coverage is found in less than 4.08% of the projects. Among the projects for which our algorithm achieved superior coverage, *follow* presents the most significant increases, on average 25.26% for line, branch, and mutation coverage.

From the results above, it can be concluded that the PCA-DynaMOSA algorithm has the potential to improve line, branch, and mutation coverage for multi-criteria test case generation. Figure 4 shows a comparison of two algorithms in terms of projects under test with a significant increase or decrease in coverage achieved by the algorithm. From Figure 4, it is apparent that the proposed algorithm outperforms DynaMOSA.

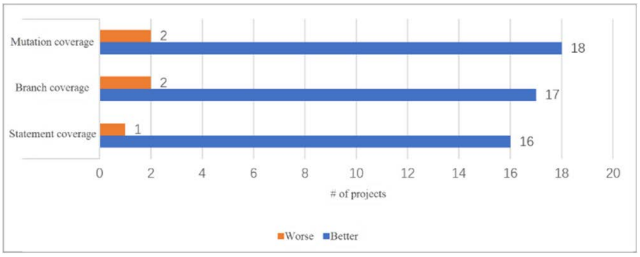

FIGURE 4. Comparison of Quantities of "Better" or "Worse"* Projects based on Line, Branch, and Mutation Coverage↗.

↗A project is a "better" project if the proposed algorithm shows more than 0.5% of performance increase than the original algorithm; a project is a "worse" project if the proposed algorithm shows worse performance than the original algorithm.

3) MULTI-CRITERIA COVERAGE AND SEARCH PERFORMANCE ACHIEVED BY PCA-DYNAMOSA (RQ 3)

a: MULTI-CRITERIA COVERAGE RESULT

Table 10 shows a comparison of the results of multi-criteria coverage between DynaMOSA and PCA-DynaMOSA.

According to Table 10, the multi-criteria coverage achieved by PCA-DynaMOSA significantly improves on that of DynaMOSA. The proposed algorithm achieves equivalent or higher coverage on all the tested projects, with no projects showing lower coverage. More specifically, compared with DynaMOSA, the multi-criteria coverage achieved by our approach is on average significantly higher in 32.65% of the classes under test. The multi-criteria coverages achieved for the remaining projects was the same as those achieved by DynaMOSA. The mean multi-criteria coverage of PCA-DynaMOSA increased by 4.40% in comparison with that of DynaMOSA.

TABLE 10. Comparison of multi-criteria coverage between DynaMOSA and PCA-DynaMOSA.

Project	PCA-DYNAMOSA Multi-criteria coverage	DYNAMOSA Multi-criteria coverage	PCA-DYNAMOSA versus DYNAMOSA
imsmart	75.25%	75.25%	-
inspirento	94.16%	91.48%	2.93%
jsecurity	90.81%	89.95%	0.96%
apbsmem	91.93%	91.62%	-
xisemele	86.99%	85.55%	1.69%
corina	75.53%	75.18%	-
schemaspy	69.50%	68.60%	1.31%
diffi	91.83%	89.12%	3.05%
follow	90.44%	72.46%	24.81%
resources4j	95.28%	82.01%	16.18%
hft-bomberman	97.56%	97.07%	0.50%
dom4j	82.73%	82.41%	-
openjms	82.40%	80.93%	1.81%
lhamacaw	68.66%	66.36%	3.47%
twfbplayer	93.94%	93.13%	0.87%
wheelwebtool	63.89%	62.13%	2.84%
xbus	83.90%	81.20%	3.32%
quickserver	93.28%	92.21%	1.16%
heal	78.19%	78.50%	-
liferay	74.48%	72.23%	3.12%
firebird	91.71%	89.59%	2.36%
Mean over all projects	84.40%	81.76%	**4.40%**

From the results above, it can be concluded that, for multi-criteria coverage, dimensionality reduction improves the efficiency of many-objective search-based test case generation without causing the search performance of the algorithm to decline.

b: SEARCH EFFICIENCY COMPARISON

Table 11 shows a comparison of the search performance of DynaMOSA and PCA-DynaMOSA. Although the coverage of testing criteria of the final test suite is important, the search efficiency of the corresponding algorithm is also crucial. The AUC [12], a statistical value representing the search efficiency of the test case generation algorithm, is also used for comparison. An algorithm with high search efficiency (or AUC value) is preferred, because it has the potential to produce a test suite with high coverage in a short period. Table 11 shows that PCA-DynaMOSA achieves a higher AUC value than DynaMOSA for most projects, with an average improvement of 1.72%.

TABLE 11. Search efficiency of DynaMOSA and PCA-DynaMOSA for each project.

Project	PCA-DYNAMOSA AUC	DYNAMOSA AUC	PCA-DYNAMOSA versus DYNAMOSA
imsmart	74.23%	74.33%	-
beanbin	87.36%	86.80%	0.64%
inspirento	89.57%	86.53%	3.52%
jsecurity	88.91%	88.38%	0.60%
nekomud	14.04%	14.05%	-
apbsmem	84.72%	87.53%	-3.21%
xisemele	85.36%	83.15%	2.65%
corina	73.61%	70.60%	4.26%
schemaspy	66.91%	65.12%	2.75%
diffi	88.89%	86.56%	2.69%
glengineer	83.24%	84.74%	-1.77%
follow	77.57%	69.44%	11.71%
resources4j	94.22%	81.29%	15.90%
diebierse	94.41%	92.90%	1.63%
jhandballmoves	95.91%	96.53%	-0.64%
hft-bomberman	95.60%	95.43%	-
dom4j	77.70%	78.11%	-0.53%
openjms	80.50%	78.96%	1.95%
biblestudy	97.81%	96.81%	1.03%
lhamacaw	63.50%	59.83%	6.12%
fim1	90.09%	87.01%	3.54%
fixsuite	34.89%	41.81%	-16.55%
twfbplayer	91.63%	90.15%	1.64%
wheelwebtool	57.10%	57.43%	-0.58%
javathena	83.03%	82.45%	0.70%
xbus	81.21%	79.72%	1.87%
classviewer	88.38%	88.52%	-
quickserver	91.05%	90.60%	0.50%
heal	74.48%	73.20%	1.76%
liferay	70.76%	69.17%	2.30%
pdfsam	84.85%	84.57%	-
firebird	87.22%	85.46%	2.05%
Mean over all projects	79.65%	78.35%	**1.72%**

TABLE 12. Number of projects with better and worse results in multi-criteria coverage and AUC value.

Algorithm	Coverage criteria	Better	Worse
PCA-DYNAMOSA vesus DYNAMOSA	Multi-criteria coverage	16 (32.65%)	0 (0.00%)
	AUC	21 (42.86%)	6 (12.24%)

Table 12 shows that, compared with DynaMOSA, the AUC value achieved by PCA-DynaMOSA is on average significantly higher in 42.86% of the classes under test. A decrease in coverage is found in less than 12.24% of the projects. This demonstrates that the proposed improvements to DynaMOSA are valid. As the efficiency of the test case generation algorithm is improved, test cases with high coverage are efficiently generated.

The correlation between AUC and multi-criteria coverage can be observed from Table 12. As the AUC value increases, the final overall coverage also increases, and vice versa. Thus, to enhance the coverage of the test cases, the search performance of the algorithm must be improved.

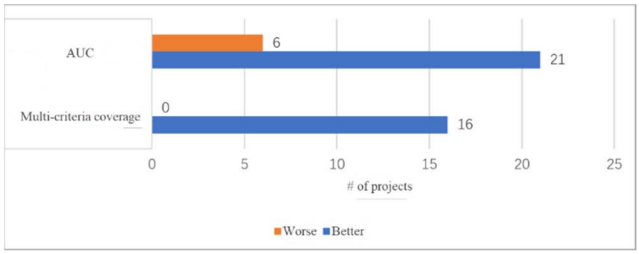

FIGURE 5. Comparison of Quantities of "Better" and "Worse" Projects based on AUC and Multi-Criteria Coverage.

Figure 5 shows the "better" and "worse" projects based on AUC and multi-criteria coverage from the perspective of their quantity.

C. THREATS TO RESEARCH VALIDITY

To reduce the randomness of the results generated by the proposed algorithm, the experiments were repeated 10 times. The results obtained from the experiments were averaged, and the mean values were selected for analysis and comparison. However, this repetition lowered but did not entirely remove the influence of randomness in the algorithm. Additionally, a proportional change of no more than 0.5% was considered an insignificant change or no change. Projects or classes with an increase rate less than 0.5% were therefore removed when comparing the coverage and search efficiency of the algorithms.

Moreover, many parameters may have affected the experimental results. To ensure the validity of the results, the same parameter settings were used in all the algorithms under comparison, and all the algorithms were run in the same framework for controlled experiments. Furthermore, the datasets selected for the experiments have been widely used in similar studies. However, they do not contain the most recent open-source projects. To enhance the generality of the research findings, both algorithms were evaluated according to the three most common coverage criteria (line, branch, and mutation). The effectiveness of the proposed algorithm on other coverage criteria, such as MC/DC, requires additional analysis.

VI. CONCLUSION

Traditional Pareto-based many-objective search algorithms have a limitation of inadequate selection pressure in the context of excessive objectives. For the problem of generating test cases with high coverage for multiple criteria, there are dozens or hundreds of coverage objectives in a single class. In this case, it is less likely that search-based test case generation algorithms can produce optimal solutions. To solve this problem, algorithms with stronger search performance when there is an excessive number of covering objectives should be selected for test case generation.

In this paper, the PCA-DynaMOSA algorithm is proposed for multi-criteria test case generation. PCA is adopted to perform dimensionality reduction for an excessive number of covering objectives while preserving important information. The proposed algorithm was experimentally validated on selected open-source datasets from SF110. The results show that, compared with DynaMOSA, the coverage achieved by our approach is on average significantly higher in 32.65% of the classes under test for line, 34.69% for branch, 36.73% for mutation, and 32.65% for multi-criteria. In comparison with DynaMOSA, the mean coverage of PCA-DynaMOSA increased by 3.87% for line, 5.50% for branch, 3.75% for mutation, and 4.40% for multi-criteria. In terms of the algorithms' search efficiency, PCA-DynaMOSA also outperforms DynaMOSA and produces a higher AUC value.

REFERENCES

[1] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, A. Bertolino, and J. J. Li, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, pp. 1978–2001, Aug. 2013.
[2] P. M. S. Bueno, M. Jino, and W. E. Wong, "Diversity oriented test data generation using Metaheuristic search techniques," *Inf. Sci.*, vol. 259, pp. 490–509, Feb. 2014.
[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, May 2015.
[4] C. Hsin-Yu, H. Chin-Yu, and F. Chih-Chiang, "Applying slicing-based testability transformation to improve test data generation with symbolic execution," *Int. J. Performability Eng.*, vol. 17, no. 7, pp. 589–599, Jul. 2021.
[5] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Reading, MA, USA: Addison-Wesley, 1999.
[6] B. W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Softw.*, vol. 1, no. 1, pp. 75–88, Jan. 1984.
[7] A. K. Kar, "Bio inspired computing—A review of algorithms and scope of applications," *Expert Syst. Appl.*, vol. 59, pp. 20–32, 2016.
[8] G. Fraser, A. Arcuri, and P. McMinn, "A memetic algorithm for whole test suite generation," *J. Syst. Softw.*, vol. 103, pp. 311–327, May 2015.
[9] M. Zhang and H. Li, "Research on automatic generation of test data for multi-path coverage," in *Proc. 4th Int. Conf. Adv. Electron. Mater., Comput. Softw. Eng. (AEMCSE)*, Mar. 2021, pp. 1185–1188.
[10] 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.
[11] K. Deb, "Multi-objective optimization," in *Search Methodologies*. Boston, MA, USA: Springer, 2014, pp. 403–449.
[12] A. Panichella, F. M. Kifetew, and P. Tonella, "Reformulating branch coverage as a many-objective optimization problem," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2015, pp. 1–10.
[13] T. Yao, "Research on automatic generation of multi-path coverage test cases based on genetic algorithms," M.S. thesis, Dept. Comput. Sci., Univ. South China, Hengyang, China, 2018.
[14] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 122–158, Feb. 2017.
[15] A. Panichella, F. M. Kifetew, and P. Tonella, "Incremental control dependency frontier exploration for many-criteria test case generation," in *Proc. Int. Symp. Search Based Softw. Eng.* Cham, Switzerland: Springer, 2018, pp. 309–324.
[16] G. Grano, C. Laaber, A. Panichella, and S. Panichella, "Testing with fewer resources: An adaptive approach to performance-aware test case generation," *IEEE Trans. Softw. Eng.*, vol. 47, no. 11, pp. 2332–2347, Nov. 2021.
[17] A. Shamsu, A. Zakari, H. Abdu, A. Nura, M. A. Zayyad, S. Suleiman, A. Adamu, and A. S. Mashasha, "Software testing: Review on tools, techniques and challenges," *Int. J. Adv. Res. Technol. Innov.*, vol. 2, no. 2, pp. 11–18, 2020.
[18] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdiscipl. Rev., Comput. Statist.*, vol. 2, no. 4, pp. 433–459, 2010.
[19] G. Fraser and A. Arcuri, "A large-scale evaluation of automated unit test generation using EvoSuite," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 2, pp. 1–42, Dec. 2014.

[20] *CKJM Extended—An Extended Version of Tool for Calculating Chidamber and Kemerer Java Metrics*. Accessed: Apr. 16, 2021. [Online]. Available: http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/
[21] B. Suri and S. Singhal, "Investigating the OO characteristics of software using CKJM metrics," in *Proc. 4th Int. Conf. Rel., Infocom Technol. Optim. (ICRITO) (Trends Future Directions)*, Sep. 2015, pp. 1–6.

DONGCHENG LI received the B.S. degree in computer science from the University of Illinois, Springfield, and the M.S. degree in software engineering from The University of Texas at Dallas, where he is currently pursuing the Ph.D. degree. His research interests include search-based software engineering and intelligent optimization algorithms.

W. ERIC WONG received the M.S. and Ph.D. degrees in computer science from Purdue University. He was at Telcordia Technologies (formerly, Bellcore) as a Senior Research Scientist and a Project Manager, where he was in charge of dependable telecom software development. He is currently a Full Professor and the Founding Director of the Advanced Research Center for Software Testing and Quality Assurance, Computer Science Department, The University of Texas at Dallas. He also has an appointment as a Guest Researcher with the National Institute of Standards and Technology (NIST), an agency of the U.S. Department of Commerce. He has very strong experience developing real-life industry applications of his research results. His research interest includes helping practitioners improve the quality of software while reducing the cost of production. In particular, he is working on software testing, debugging, risk analysis/metrics, safety, and reliability. In 2014, he was named as the IEEE Reliability Society Engineer of the Year. He is also the Founding Steering Committee Chair of the IEEE International Conference on Software Quality, Reliability, and Security (QRS). He is the Editor-in-Chief of the IEEE TRANSACTIONS ON RELIABILITY.

SEAN PAN received the bachelor's degree in computer science from the National Taiwan University, the master's degree in computer science from Michigan State University, and the Ph.D. degree in computer science from Purdue University. He has more than 20 working experience at IBM, Clarent, Verity, Symantec, and RFCyber, where he held various Senior Management and Software Engineer positions with specialties on business development, blockchain, smart contract, cybersecurity, contactless mobile payment, radio frequency identification (RFID), VoIP, search engine, database engine, and software engineering. He was an Associate Professor of computer science at the National Chiao Tung University, Taiwan. He has awarded 26 patents from the USA, China, and Taiwan in the area of contactless mobile payment, database engine, and distributed system. RFCyber Corporation is a Texas registered company, since 2020, and provides solutions on web 3.0, such as block chain, smart contract, cryptocurrency, and mobile payment. It has offices at Waco, TX, USA, and Plano, TX, to sponsor joint research projects with Baylor University and The University of Texas at Dallas, respectively.

LIANG-SENG KOH received the bachelor's degree in computer science from the National University of Singapore, and the master's and Ph.D. degrees in computer science from The Ohio State University. For the last 20 years, he worked and held various senior engineering positions in different computer software industries, including telecommunication, business workflow automation, cyber security, radio-frequency identification (RFID), and contactless mobile payment. He has been awarded ten patents from the USA and eight patents from China in the area of contactless mobile payment transaction security and secure element provisioning. His research interest includes web 3.0 topics, such as blockchain, smart contracts, and cryptocurrency.

SHENGLONG LI received the bachelor's degree in physics and the master's degree in computer technology from the China University of Geosciences, Wuhan. His main research interests include intelligent dispatching and information engineering.

MATTHEW CHAU is currently pursuing the bachelor's degree in electrical engineering with The University of Texas at Dallas. His research interests include software fault prevention, software reliability, and preventive engineering.