IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 31, 2020, accepted March 11, 2020, date of publication March 27, 2020, date of current version April 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2983762

Testing Method for Software With Randomness Using Genetic Algorithm

XIANGJUAN YAO[1], DUNWEI GONG[2], (Member, IEEE), BIN LI[1], XIANGYING DANG[2], AND GONGJIE ZHANG[3]
[1]School of Mathematics, China University of Mining and Technology, Xuzhou 221116, China
[2]School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China
[3]School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China

Corresponding author: Xiangjuan Yao (yaoxj@cumt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61573362, Grant 61773384, and Grant 61203304, and in part by the National Key Research and Development Program of China under Grant 2018YFB1003802-01.

ABSTRACT The main task of software testing is the generation of test data with high quality in restricted time. But for software with uncertainties, such as randomness, existing methods of generating test data often lose their effectiveness. For software with randomness, a test datum might cover the test target in one run, but obtains different results in the next run. Therefore, special testing methods must be exploited for this kind of software. In order to test software with randomness, we first propose a novel test adequacy criterion, and then build a mathematical model for generating test data according to the new criterion. Finally, we present a method of solving the optimization model using a set-based genetic algorithm. We apply the proposed method to test 12 programs, and compare with traditional genetic algorithm and the random method. From the experimental results we can see that, the proposed adequacy criterion is available for the software with randomness and the proposed algorithm can effectively generate test data.

INDEX TERMS Software testing, genetic algorithm, adequacy criterion, test data generation, software with randomness.

I. INTRODUCTION

In recent years, the importance of software quality continuously grows, for software is playing a vital role in national economy and social development. Poor software may result in not only a high maintenance cost, but also a huge amount of property loss and even serious national security or environmental issues [1], [2].

Software testing plays an important role in ensuring the quality of software, and thus has a critical position in software engineering. It has been observed that software testing may occupy more than 50% of the development cost in the software life cycle [3]. Moreover, according to Boehm's results, the later a defect is discovered, the more expensive it will be to modify it [4]. Therefore, it is very important to improve the effectiveness and efficiency of software testing.

Existing methods of software testing are mainly for deterministic programs. In face, there are various forms of uncertainty in many practical programs, such as randomness or fuzziness, suggesting that the behavior of these programs is uncertain. Specifically, for the program with uncertainty, when repeatedly running the program with the same test datum, it may traverse different paths, cover different statements or even have different outputs. On this circumstance, previous test adequacy criteria are not applicable.

At present, there have been some research results oriented to test a program with nondeterminism [5], but few of these researches aim at the programs containing randomness. On the other hand, programs with randomness widely exist in real-world applications, such as game software, Windows operating system, and network software [7].

For example, when a human player challenges the Chinese chess game by a computer program. In general, the program determines its operation according to some strategies. But the strategy may include some random choices. The computer decisions will be left unspecified as nondeterministic choices. So research on testing a program with randomness is quiet necessary and meaningful. If we can detect the impact of random behavior on the program through some test cases generated by random numbers, the stability of the program will be guaranteed.

The associate editor coordinating the review of this manuscript and approving it for publication was Geng-Ming Jiang.

Given the fact that the key step of software testing is to produce effective test data [8], this paper intends to study the problem of generating test data for programs with randomness. First, we propose a novel adequacy criterion for testing programs with randomness, and then establish an optimization model for the problem of generating test data for this kind of programs. Finally, we present a method of solving the optimization problem using a set-based genetic algorithm. Our experimental results illustrate that the proposed adequacy criterion is available for the software with randomness; In addition, the proposed algorithm can effectively generate test data.

The main contributions of this paper are as follows:

1) A novel adequacy criterion is proposed for testing a program with randomness.
2) An optimization method is presented to solve the problem of generating test data for the program with randomness.

The structure behind this section is as follows. Related works are put forward in Section 2. Section 3 proposes a novel adequacy criterion for testing a program with randomness. The mathematical model of the problem to produce test data according to the new criterion is built in Section 4. Section 5 presents a method of solving the optimization model based on a genetic algorithm. The applications of the proposed method in 12 programs and the comparisons with traditional genetic algorithm and the random method are demonstrated in Section 6. Finally, Section 7 concludes the whole paper, and comes out several topics to be further studied.

II. RELATED WORKS

Automatically generating test data will effectively reduce the labor intensity of the tester, improve testing efficiency and software quality, and therefore save the cost of software development [9].

In general, there are four kinds of methods of test data generation, i.e. random method, static method and dynamic method, as well as heuristic method [10].

Random method generates test data by randomly sampling the input space of the program under test. This kind of methods is simple and able to quickly generate a large amount of test data, but has great blindness [11]. In order to improve the efficiency of random testing, Chan *et al.* proposed the method of restricted random testing that offers a significant improvement over random testing [12].

Static method only carries out static analysis and transformation, and does not involve practical operation of the program [13]. This kind of methods usually need to perform many algebra and (or) interval operations, and the needed storage space is also very huge [14].

Dynamic method generates test data based on actual operation of the program, and the process is determinate [15]. This kind of methods need quite long time to generate test data, and is very sensitive to the initial test data [16]–[18].

Different from dynamic method, the process of generating test data for heuristic method is not completely determinate [19], although heuristic method is also based on the actual operation of the program. These methods include tabu search algorithm [20], particle swarm optimization algorithm [21], evolutionary strategy [22], and genetic algorithm [23], etc.

For complex software, using genetic algorithm to speed up the efficiency of test data generation is a potential research direction, and has achieved many inspiring results [24].

Baars *et al.* [25] presented an search-based testing method to generate test data satisfying branch adequate criterion. Pachauri and Srivastava [26] proposed an approach to test data generation for branch coverage with a structured genetic algorithm. Bueno and Jino [27], Lin and Yeh [28], and Watkins and Hufnagel [29] separately generated test data based on GAs for path coverage testing. Mei and Wang [30] automatically generated test cases to cover the selected paths using a special GA. Yao *et al.* [31] proposed a test data generation method for multi-path coverage based on a genetic algorithm with local evolution.

Although there are many test data generation methods, most of them are for traditional software. How to fully test the special software with random numbers has not been studied. Therefore, this paper focuses on the generation of test data for software with random numbers, which will no doubt bring new research topics to the field of software testing.

III. ADEQUACY CRITERION FOR TESTING PROGRAM WITH RANDOM NUMBER

To test a software product, we need first to generate a test suite according to some given criterion. Then, the program will be run to find out if there are errors or faults. To ensure the adequacy of the test, people put forward different test criteria to evaluate the testing results. This paper mainly orients coverage-based software testing. For this kind of testing, traditional adequacy criterion can be described as follows:

[Test Adequacy Criterion 1] [31] For a given set of targets (statements, branches, or paths and so on) of a program, find a test suite in the input domain of the program under test, such that for each target, there is at least one test datum in the test suite that can cover it.

Different test targets correspond to different test criteria. For example, statement coverage criterion requirements that the test data can cover all executable statements. Generally speaking, path coverage criterion is more difficult than branch coverage criterion, and branch coverage criteria is more difficult than statement coverage criterion.

For a program without uncertainties, the above criterion is valid. However, there are some limitations when applying it to a program with randomness. For a program with uncertainties, whether a test datum can cover a target or not is not deterministic. That is to say, the program may have different execution results with the same test datum. To clearly illustrate this, let us investigate the program shown in Figure 1.

```
%input a;
r=rand;   % r is a random parameter in the program
if a > 0
{     if r>0.5
            Result= 1;     %target statement 1
      else
            Result = -1;     % target statement 2
}
else
{     if r>0.7
            Result = 2;     % target statement 3
      else
            Result = -2;     % target statement 4
}
```

FIGURE 1. Example program.

The program in Figure 1 contains 4 target statements. The input of this program is a. r is a random number that obeys the uniform distribution from 0 to 1. When a is larger than 0, it can cover target statement 1 when $r > 0.5$ or statement 2 when $r \leq 0.5$. When a is smaller than or equal to 0, it can cover target statement 3 when $r > 0.7$ and target statement 4 when $r \leq 0.7$. Since the value of r is random, for any input a, which target statement will be covered is uncertain.

For a program without uncertainties, whether a test datum can cover a target is determined, suggesting that if we repeatedly run the program with the same test datum, the results of these runs are all the same. However, for a program with randomness, different executions may cause different results. Therefore, test adequacy criteria 1 does not applicable, and new test adequacy criteria for this kind of programs should be presented.

For a program with randomness, whether a test datum can cover a target is uncertain, therefore, we can regard it as a random event. Although the occurrence of a random event is uncertain, the probability of a random event occurring is determined under a specific condition. Therefore, we can present a test adequacy criterion for a program with randomness based on probability theory.

For the above example, suppose that the input of the program, a, is greater than 0. We can see that a can cover target statement 1 if and only if $r > 0.5$, where r is a random number. Assume that the probability density function of r is $f(x)$, then the probability of a covering target statement 1 is:

$$p = p\{r > 0.5\} = \int_{0.5}^{1} f(x)dx.$$

Because r obeys the uniform distribution from 0 to 1, $f(x) = 1$ when $x \in [0.1]$. So $p = 0.5$.

Because of the existence of random number, a test datum larger than 0 cannot guarantee to cover target statement 1. In this case, we can increase the probability of test target being covered by increasing the number of test data. Suppose that there are n test data, $a_1, a_2, \cdots, a_n$, where $a_i > 0, i = 1, 2, \cdots, n$, the probability that the test suite, $\{a_1, a_2, \cdots, a_n\}$, covers statement 1 is:

$$p_1 = 1 - (1 - 0.5)^n \tag{1}$$

We can see from formula (1) that, the value of p_1 goes up as the value of n increases. For some test target, although we cannot guarantee that it must be covered, we can guarantee the probability of this test target being covered is larger than a given positive between 0 and 1. Based on this observation, we present a novel adequacy criterion for testing a program with randomness as follows:

[Test Adequacy Criterion 2] For a given set of targets (statements, branches, paths and so on), seeking for a test suite in the input domain of the program, such that for each target, the probability of the test suite covering the target is not less than a given positive value.

For a program without uncertainties, the probability of a test datum covering a target is either 1 or 0. Then, the condition that the probability is larger than a given positive value is equivalent to the test datum covering the target. So, test adequacy criterion 2 is equivalent to test adequacy criterion 1 for a program without uncertainties. In this sense, test adequacy criterion 2 is a generalization of test adequacy criterion 1.

In addition, different test targets correspond to different test methods. For example, if the test targets are statement, the corresponding method is called statement coverage testing.

IV. OPTIMIZATION MODEL FOR TEST DATA GENERATION

After giving the novel adequacy criterion for testing a program with uncertainties, the next step is to generate test data that satisfy the criterion. Next, we will take the branch coverage criterion as an example to build the optimization model for the problem of generating test data to cover a series of branches of a program with randomness. Of course, this method can also be applied to other test criteria.

A. DECISION VARIABLE

Let the program under test be G, and the input of G be $x_1, x_2, \cdots, x_m$. Then, the input of G forms the following vector, $X = (x_1, x_2, \cdots, x_m)$. If the domain of x_i is D_i, $i = 1, 2, \cdots, m$, the domain of the input of G can be represented as $D(G) = D_1 \times D_2 \times \cdots \times D_m$, where "$\times$" refers to the cartesian product.

Suppose that program G has b target branches, denoted as $\mathscr{B}_1, \mathscr{B}_2, \cdots, \mathscr{B}_b$. According to test adequacy criterion 2, the problem of generating test data for covering $\mathscr{B}_1, \mathscr{B}_2, \cdots, \mathscr{B}_b$ can be described as: seeking for a test suite $\{X_1, X_2, \cdots, X_n\}$ in the domain of the input of G, such that the probability of each target branch being executed exceeds a given positive value. Since the task of generating test data is to determine the value of $X_1, X_2, \cdots, X_n$, they are regarded as the decision variable of the optimization problem.

B. OPTIMIZATION MODEL

Let $\mathscr{X} = \{X_1, X_2, \cdots, X_n\}$ be a test suite that contains n test data. If we take $\mathscr{X}$ as the input of a program to run it, and obtain the minimal probability of all targets being covered, denoted as q, we call q the credibility of the test suite, $\mathscr{X}$. For a given positive value, α, $\mathscr{X}$ satisfies test adequacy criterion 2 if and only if $q \geq a$. In the following, we will discuss how to obtain the credibility of $\mathscr{X}$.

Let A_{ij} be a random event that X_i covers branch $\mathscr{B}_j$. If $p_{ij} = p(A_{ij})$ is 0 or 1, A_{ij} is a deterministic event; otherwise, $0 < p_{ij} < 1$, and A_{ij} is random. Generally, the execution of a program is related only to its current input, so these test data can be regarded as independent from each other. Consequently, the events of $A_{1j}, A_{2j}, \cdots, A_{nj}$ are also independent from each other.

When all test data in $\mathscr{X}$ have been executed, the probability of $\mathscr{B}_j$ being covered is

$$\begin{aligned} p_{\mathscr{B}_j} &= p(A_{1j} \cup A_{2j} \cup \cdots \cup A_{nj}) \\ &= 1 - p(\overline{A_{1j} \cup A_{2j} \cup \cdots \cup A_{nj}}) \\ &= 1 - p(\overline{A_{1j}} \cap \overline{A_{2j}} \cap \cdots \cap \overline{A_{nj}}) \\ &= 1 - p(\overline{A_{1j}}\}p\{\overline{A_{2j}}\} \cdots p\{\overline{A_{nj}}) \\ &= 1 - (1 - p(A_{1j}))(1 - p(A_{2j})) \cdots (1 - p(A_{nj})) \\ &= 1 - \prod_{i=1}^{n} (1 - p_{ij}) \end{aligned} \tag{2}$$

This means that the credibility of $\mathscr{X} = \{X_1, X_2, \cdots, X_n\}$ is

$$q = \min_{1 \leq j \leq b} \{p_{\mathscr{B}_j}\} = \min_{1 \leq j \leq b} \{1 - \prod_{i=1}^{n} (1 - p_{ij})\} \tag{3}$$

According to test adequacy criterion 2, q should be larger than a given positive value, α, i.e.

$$f(X_1, X_2, \cdots, X_n) = q = \min_{1 \leq j \leq b} \{1 - \prod_{i=1}^{n} (1 - p_{ij})\} \geq \alpha. \tag{4}$$

In addition, we wish the number of test data is as small as possible so as to reduce test cost. Therefore, we take n as the optimization objective.

Based on the above discussion, the optimization model of generating test data for covering branches $\mathscr{B}_1, \mathscr{B}_2, \cdots, \mathscr{B}_b$ can be formulated as:

$$\begin{aligned} &min\ n \\ &s.t. \begin{cases} f(X_1, X_2, \cdots, X_n) \geq \alpha \\ X_i \in D(G) \end{cases} \end{aligned} \tag{5}$$

C. FURTHER DISCUSSION

Function $f(X_1, X_2, \cdots, X_n)$ are given in formula 4. However, the value of p_{ij} is hard to be determined. In the following, we will present a method of approximately calculating $f(X_1, X_2, \cdots, X_n)$.

Lemma 1: $\prod\limits_{i=1}^{n} (1 - a_i) \leq (1 - \frac{1}{n} \sum\limits_{i=1}^{n} a_i)^n$, where $0 \leq a_i \leq 1$, $i = 1, \cdots, n$.

Proof: Let $h(a_1, a_2, \cdots, a_n) = \prod\limits_{i=1}^{n} (1 - a_i) - (1 - \frac{1}{n} \sum\limits_{i=1}^{n} a_i)^n$,

then

$$\begin{aligned} \frac{\partial h}{\partial a_k} &= -\prod_{\substack{i=1 \\ i \neq k}}^{n} (1 - a_i) - n(1 - \frac{1}{n} \sum_{i=1}^{n} a_i)^{n-1}(-\frac{1}{n}) \\ &= -\prod_{\substack{i=1 \\ i \neq k}}^{n} (1 - a_i) + (1 - \frac{1}{n} \sum_{i=1}^{n} a_i)^{n-1} \end{aligned}$$

Let

$$\begin{cases} \dfrac{\partial h}{\partial a_1} = 0 \\ \dfrac{\partial h}{\partial a_2} = 0 \\ \cdots \\ \dfrac{\partial h}{\partial a_n} = 0 \end{cases}$$

We obtain the stationary points $a_1 = a_2 = \cdots = a_n$. The maximum value of function $h(a_1, a_2, \cdots, a_n)$ may be obtained at stationary points or boundary points. When $a_1 = a_2 = \cdots = a_n$, $h(a_1, a_2, \cdots, a_n) = 0$. Now consider the boundary points. Suppose that $a_{i1} = a_{i2} = \cdots = a_{ik} = 1 (k \geq 1)$ and for any $i \in \{1, 2, \cdots, n\} \backslash \{i1, i2, \cdots, ik\}, a_i = 0$, we obtain $h(a_1, a_2, \cdots, a_n) = -(1 - \frac{k}{n})^n \leq 0$. So the maximal value of $h(a_1, a_2, \cdots, a_n)$ is 0, thus $h(a_1, a_2, \cdots, a_n) \leq 0$, i.e., $\prod\limits_{i=1}^{n} (1 - a_i) \leq (1 - \frac{1}{n} \sum\limits_{i=1}^{n} a_i)^n$. ■

Let $\overline{p}_j$ be the probability of an arbitrary test datum in $\{X_1, X_2, \cdots, X_n\}$ covering target $\mathscr{B}_j$, then $\overline{p}_j = \frac{1}{n} \sum\limits_{i=1}^{n} p_{ij}$.

Theorem 2: $1 - \prod\limits_{i=1}^{n} (1 - p_{ij}) \geq 1 - (1 - \overline{p}_j)^n$, where $\overline{p}_j = \frac{1}{n} \sum\limits_{i=1}^{n} p_{ij}$.

Proof: In Lemma 1, let $a_i = p_{ij}$, we obtain

$$\prod_{i=1}^{n} (1 - p_{ij}) \leq (1 - \overline{p}_j)^n$$

■

By Theorem 2, the credibility of $\{X_1, X_2, \cdots, X_n\}$ is

$$q = \min_{1 \leq j \leq b} \{1 - \prod_{i=1}^{n} (1 - p_{ij})\} \geq \min_{1 \leq j \leq b} \{1 - (1 - \overline{p}_j)^n\} = q'$$

If $q' \geq \alpha, q \geq \alpha$. So we can substitute q with q' to evaluate the credibility of $\{X_1, X_2, \cdots, X_n\}$. However, the value of q' is also not easy to be calculated. In the following, we will employ a statistical method to estimate it.

Let Y_j be the number of test data that cover $\mathscr{B}_j$ when we run the program with $\{X_1, X_2, \cdots, X_n\}$ as the input, respectively, then $Y_j (j = 1, 2, \cdots, b)$ is a random variable. Let

$$Z_{ij} = \begin{cases} 1, & X_i \ covers \ \mathscr{B}_j \\ 0, & otherwise \end{cases}$$

Then $Y_j = Z_{1j} + Z_{2j} + \cdots + Z_{nj}$. So

$$E(\frac{Y_j}{n}) = \frac{1}{n}\sum_{i=1}^{n} E(Z_{ij}) = \frac{1}{n}\sum_{i=1}^{n} p_{ij} = \overline{p}_j$$

Thus, $\frac{Y_j}{n}$ is an unbiased estimator of $\overline{p}_j$, and $f(X_1, X_2, \cdots, X_n)$ can be estimated as

$$f(X_1, X_2, \cdots, X_n) \approx \min_{1 \le j \le b} \{1 - (1 - \frac{Y_j}{n})^n\} \tag{6}$$

V. TEST DATA GENERATION USING SET-BASED GENETIC ALGORITHM

In order to solve the optimization model formulated in Section 4.2, we propose a set-based genetic algorithm. In the following, we will first give the representation of the individual. Following that, we will design the fitness function, and present the genetic operators. Finally, the steps of the proposed algorithm are listed.

A. INDIVIDUAL REPRESENTATION

The decision variables of optimization model (5) are $X_1, X_2, \cdots, X_n$, so when we solve the model using evolutionary methods, an individual in the population is a test suite $\{X_1, X_2, \cdots, X_n\}$, where X_i is an input of program G. If $X_i = (x_{i1}, x_{i2}, \cdots, x_{im})$, individual $\{X_1, X_2, \cdots, X_n\}$ can be represented by the following matrix with n rows and m columns:

$$M_{n\times m} = \begin{array}{c} \left[\begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{array}\right] \end{array} \begin{array}{c} X_1 \\ X_2 \\ \vdots \\ X_n \end{array} \tag{7}$$

where each input is encoded by the binary coding.

B. FITNESS FUNCTION

In order to solve the optimization model formulated in Section 4.2 using genetic algorithms, we need to design a fitness function to evaluate the individuals. For the two objectives in this model, it is more important to maximum the required coverage rate than minimum the number of test data. So, we design the fitness function of individual $\{X_1, X_2, \cdots, X_n\}$ as follows:

$$fitness(X_1, X_2, \cdots, X_n) = \frac{f(X_1, X_2, \cdots, X_n)}{n + \theta} \tag{8}$$

where θ is a positive constant and determined according to the size of the program.

C. GENETIC OPERATORS

Due to the particularity of the individual representation, it is necessary to design appropriate genetic operators, including: selection operator, crossover operator and mutation operator.

1) SELECTION OPERATOR

A part of the existing individuals is selected to breed a new generation according to the fitness function. We adopt formula (8) as the fitness of an individual. In this paper, we use the roulette wheel method to select individuals.

2) MUTATION OPERATOR

We employ two kinds of mutation operators: one is called compression mutation that realizes the decrease of the test data in a test suite; while the other one called expansion mutation realizes the increase of the test data in a test suite.

a: COMPRESSION MUTATION

Suppose that $M_{n\times m} = \{x_{ij}\}$ is an individual. We randomly select a point (a test datum), denoted as $k (1 \le k < n)$. Then we can obtain the new individual by the following way:

$$x'_{i,j} = \begin{cases} x_{i,j}, & if \ i \le k \\ x_{i+1,j}, & if \ k < i \le n-1 \end{cases}$$

The resulting individual $M'_{(n-1)\times m} = [x'_{i,j}]$ is the children.

b: EXPANSION MUTATION

Suppose that $M_{n\times m} = \{x_{ij}\}$ is an individual. A single point is selected randomly, denoted as $l (1 \le l \le n)$. Then we randomly generate a test datum X^*, denoted as $X^* = (x_1^*, x_2^*, \cdots, x_m^*)$. Then let

$$x'_{i,j} = \begin{cases} x_{i,j}, & if \ i \le l \\ x_j^*, & if \ i = l \\ x_{i-1,j}, & if \ l+1 \le i \le n+1 \end{cases}$$

The resulting individual $M'_{(n+1)\times m} = [x'_{i,j}]$ is the children.

The aim of these two types of operators is to add or reduce a test suite in the individual. Thus, in the process of evolution, the number of test data in an individual may change.

3) CROSSOVER OPERATOR

Let $M_{n_1\times m} = [x_{i,j}]$ and $N_{n_2\times m} = [y_{i,j}]$ be two different individuals that are represented with two matrixes (The numbers of their rows may be different.). A crossover point is randomly selected, denoted as $(r, c) (1 \le r < min\{n_1, n_2\}, 1 \le c < m)$. Let

$$x'_{i,j} = \begin{cases} y_{i,j}, & if \ i \le r \ and \ j \le c \\ x_{i,j}, & if \ r < i \le n_1 \ or \ c < j \le m \end{cases}$$

$$y'_{i,j} = \begin{cases} x_{i,j}, & if \ i \le r \ and \ j \le c \\ y_{i,j}, & if \ r < i \le n_2 \ or \ c < j \le m \end{cases}$$

The resulting individuals $M'_{n_1\times m} = [x'_{i,j}]$ and $N'_{n_2\times m} = [y'_{i,j}]$ are the children.

D. TERMINATION CONDITION

By the discussion in Section 4.2 we know that, for the given threshold α, the reliability of $q \ge 1 - \alpha$, if and only if

TABLE 1. Basic information of test subjects.

No.	program	loc	function
1	Checkout	51	A simulation system of supermarket checkout line problem
2	Guessing	105	Output the result according to the drawn number
3	Fishing	112	Random Simulation Fishing
4	Distribution	180	Distribution Function Simulation
5	Poker	305	Poker Random Number Test
6	Tickets	450	Lottery Tickets Numbers Simulation
7	Random Number	610	Random Number Simulated Multi-Threaded
8	Voltage Signal	1650	Voltage Signal Stochastic Simulation
9	Dark Throne	16500	3D Action Role Playing Game
10	Angry Robots	18000	3D Action Role Playing Game
11	SDK	21500	SDK access information validation program
12	Miracle Come	23000	3D Action Role Playing Game

$q' = \min_{1\le j\le m} (1-(1-\bar{p}_j)^n) \ge 1-\alpha$. Let $p = \min_{1\le j\le m} \bar{p}_j$, we obtain

$$(1-(1-p)^n) \ge 1-\alpha$$

So

$$p \ge 1-\alpha^{\frac{1}{n}}$$

That is to say, for the given n, in order to make the reliability $q \ge 1-\alpha$, we only need to assure that $p \ge 1-\alpha^{\frac{1}{n}}$, where $p = \min_{1\le j\le m} \bar{p}_j \approx \min_{1\le j\le m} \frac{Y_j}{n}$. So the termination condition of the algorithm is that $\min_{1\le j\le m} \frac{Y_j}{n} \ge 1-\alpha^{\frac{1}{n}}$ or the algorithm have reached the maximal number of generations.

E. STEPS OF THE ALGORITHM

The steps of the proposed algorithm are as follows:

Step 1: Parameter settings.

Determine the values of the control parameters, such as the population size, the number of test data in each individual, the maximal number of generations, crossover probability, mutation probability, the threshold α, etc.

Step 2: Population initialization.

Randomly generate an initial population containing κ individuals, denoted as $Pop^{(0)} = \{M_1, M_2, \cdots, M_\kappa\}$, where each individual M_i is a test suit.

Step 3: Fitness value calculation.

Calculate the fitness value of individual M_i according to formula (8). In the algorithm, the greater the fitness value of an individual is, the better it is, and thus has a greater chance to be selected to the next generation.

Step 4: Termination condition determination.

Determine if the termination conditions of the algorithm are satisfied. If yes, go to step 6; otherwise, go to step 5.

Step 5: Genetic operator implementation.

Perform genetic operators, including selection, crossover and mutation operators, and go to step 3.

Step 6: Algorithm termination.

Stop the evolution and output the final results.

VI. EXPERIMENTS

This section conducts the experiments to evaluate the effectiveness and the efficiency of the proposed method.

A. TEST SUBJECTS

The experiments are based on 10 C programs with random numbers. Some basic information of these programs can be seen in Table 1. The first 8 programs are open source programs on the Internet, and the last 2 programs are action role playing games developed by a local company. All of these programs contain different number of random numbers.

B. RESEARCH QUESTIONS

We have two goals to conduct the experiment, one is to validate the effectiveness of the optimization model for test data generation based on the proposed adequacy criterion, the other is to evaluate the efficiency of the set-based evolutional algorithm in generating test data. So we consider the following research questions:

RQ1: Does the test data generated by the proposed method have higher quality?

RQ2: Could the set-based genetic algorithm generate test data more effectively?

C. EXPERIMENTAL DESIGN

1) EXPERIMENTAL DESIGN FOR FIRST GROUP OF EXPERIMENTS

This set of experiments attends to test the effectiveness of the proposed test data generation model. We choose three categories of test targets for coverage, i.e. statement, branch and path. Because this paper mainly studies the problem of generating test data for programs with random number, we selected part of targets influenced by random numbers in the program for test. The number of test targets for each programs are listed in Table 2.

In the circumstance of not causing confusion, we also call them statement coverage, branch coverage and path coverage testing, respectively. Unlike traditional test methods, we ensure that each test target can be covered with a certain probability. For each program under test and each testing method, we try to generate a test suite with a higher fitness value.

For the sake of verifying the performance of the proposed method, we compare it with traditional genetic algorithm and the random method [32]. For traditional genetic algorithm,

TABLE 2. Basic information of test subjects.

No.	No. of target statements	No. of target branches	No. of target paths
1	2	2	2
2	3	5	5
3	10	16	8
4	15	11	6
5	15	10	10
6	20	14	12
7	22	10	21
8	32	12	9
9	165	46	28
10	320	68	43
11	280	105	80
12	210	74	51

an individual is a test datum. Therefore, in order to generate test data covering all test targets, the algorithm needs to be run many times. In addition, the random method is a benchmark test data generation method. We also use it as a comparison algorithm. First, we generate a test suite according to the proposed method. Then, we generate another test suite containing the same number of test data as the first test suite using the random method and traditional genetic algorithm. We evaluate the performance of different methods by comparing the coverage rates of the test suites for the test targets.

2) EXPERIMENTAL DESIGN FOR SECOND GROUP OF EXPERIMENT

The second group of experiments aims at evaluating the efficiency of the proposed genetic algorithm. For this purpose, the traditional genetic algorithm and random method are also taken to compare with our method. In traditional genetic algorithm, the test data are generated one by one, and the fitness function is the coverage rate for the uncovered test targets. While in our method, the individual is a test suite. For the random method, the test suites are generated randomly. We will evaluate the performance of each test suite using formula (5).

We try to generate test data to cover all target statements, target branches and target paths using the proposed algorithm, traditional genetic algorithm and the random method. In order to evaluate the performance of our algorithm, and the other two methods, we compare the performance of different methods using the number of evaluations and the coverage rate. For our method and traditional genetic algorithm, the number of evaluations is equal to the total number of individuals generated during the evolution of the algorithm, while

TABLE 3. Results for statement coverage testing.

program	No. of test data	Proposed method(%)	Traditional genetic algorithm(%)	Random method(%)
1	32	97.3	92.3	83.4
2	38	95.5	78.2	76.6
3	18	92.6	90.7	83.7
4	30	96.2	89.3	80.3
5	73	93.7	88.2	73.4
6	132	95.6	91.4	81.3
7	42	100.0	97.2	82.6
8	156	93.7	86.8	78.3
9	1045	94.2	87.6	75.4
10	1065	94.2	88.3	75.2
11	640	95.3	89.2	73.3
12	1360	92.1	86.1	71.6
Average	385.92	95.03	88.78	77.93
Standard	deviation	2.1584	4.4648	4.2617

that of the random method is the total number of test data generated.

D. EXPERIMENTAL RESULTS AND ANALYSIS

In the following, we will give the experimental results of each group of experiments.

1) EXPERIMENTAL RESULTS OF FIRST GROUP OF EXPERIMENTS

a: RESULTS OF STATEMENT COVERAGE TESTING

We try to generate test data that can cover all statements of the program under test. We record the number of generated test data for each program. As a comparison, we also generate the same number of test data for each program with traditional genetic algorithm and the random method. Then, we run the program with the generated test data, and record the coverage rate of statements for each program. Due to the nondeterminism of the experimental results, we repeatedly run the program 10 times with the generated test data under the same conditions, and then calculate the average values. The final results are listed in Table 3.

As can be seen from Table 3 that, (1) for each program under test, the proposed method can achieve the highest coverage rate out of these three methods. The average coverage rate of the proposed method is 95.03%, while those of traditional genetic algorithm and the random method are only 88.78% and 77.93%, respectively; (2) the highest coverage rate of the proposed method is 100%, and the lowest coverage rate of the proposed method is 92.1%, whereas those of traditional genetic algorithm are 92.3% and 78.2%, and those of the random method are 83.4% and 71.6%, respectively. In addition, the sample standard deviation of our method is

TABLE 4. *T*-test results of statement coverage testing.

Null hypothesis	Statistic	Rejection domain	The value of statistic	Result
$X \leq Y$	$T_1 = \frac{\bar{X}-\bar{Y}}{S_\omega^1/\sqrt{n_1+n_2-2}}$	$T_1 > t_\alpha(22) = 2.5083$	$t_1 = 5.677$	$X > Y$
$X \leq Z$	$T_2 = \frac{\bar{X}-\bar{Z}}{S_\omega^2/\sqrt{n_1+n_3-2}}$	$T_2 > t_\alpha(22) = 2.5083$	$t_2 = 16.307$	$X > Y$

TABLE 5. Results for branch coverage testing.

program	No. of test data	Proposed method(%)	Traditional genetic algorithm(%)	Random method(%)
1	8	96.5	91.7	85.8
2	20	90.2	80.3	76.3
3	64	95.1	87.6	75.4
4	24	95.6	87.3	73.2
5	86	96.3	90.2	81.7
6	94	94.8	91.2	75.3
7	55	96.8	90.8	75.6
8	60	93.4	91.2	85.4
9	310	92.3	84.9	68.4
10	465	91.6	87.2	72.3
11	265	94.3	86.3	65.2
12	570	95.1	87.2	70.3
Average	168.42	94.33	87.99	75.41
Standard	deviation	2.0716	3.3107	6.3404

also the smallest, which shows that our method has highest stability.

In order to analyze the experimental results more scientifically, we use the statistical tool, Spss, to conduct $T-$test for the comparison of different methods.

Suppose that the result of the our method is X, that of traditional genetic algorithm is Y, and that of the random method is Z. We regard X, Y and Z as three random variables. Now, we're going to compare the means of X, Y and Z using statistical methods. Because there are 12 programs under test, the results of these programs can form a sample of size 12. Suppose that the results of the proposed method for the 12 programs under test are $X_1, \cdots, X_{12}$, those of traditional genetic algorithm are $Y_1, \cdots, Y_{12}$, and those of the random method are $Z_1, \cdots, Z_{12}$. Then $(X_1, \cdots, X_{12})$, $(Y_1, \cdots, Y_{12})$ and $(Z_1, \cdots, Z_{12})$ can be regarded as three samples of size 12.

Because the experimental results are affected by many factors, according to the statistical principle, X, Y and Z should obey or approximately obey the normal distribution. Therefore, we use the $T-$test method to compare their mean values. Let the significance level α be 0.01. The final results are listed in Table 4.

As can be seen from Table 4 that, the value of t_1 is 5.677, and that of t_2 is 16.307. The values of both statistics are much higher than the bound, $t_{0.01} = 2.5083$. So we can conclude that for statement coverage testing, the coverage rate of the proposed method is significantly greater than traditional genetic algorithm and the random method.

b: RESULTS OF BRANCH COVERAGE TESTING

We do the same experiments for branch testing as statement one, and the final results are listed in Table 5.

As can be seen from Table 5 that, (1) for each program under test, the proposed method can achieve the highest coverage rate out of these three methods. The average coverage rate of the proposed method is 94.33%, while those of traditional genetic algorithm and the random method are only 87.99% and 75.41%, respectively; (2) the highest coverage rate of the proposed method is 96.8%, and the lowest coverage rate of the proposed method is 90.2%, whereas those of traditional genetic algorithm are 91.7% and 80.3%, and those of the random method are 85.8% and 70.3%, respectively.

We also analyze the experimental results with the Spss tool, and the final results of the $T-$test are listed in Table 6.

As can be seen from Table 6 that, the value of t_1 is 9.813, and that of t_2 is 10.711. The values of both statistics are much higher than the bound, $t_{0.01} = 2.5083$. So we can conclude that for branch coverage testing, the coverage rate of the proposed method is significantly greater than traditional genetic algorithm and the random method.

c: RESULTS FOR PATH COVERAGE TESTING

The third group of experiments are conducted for path coverage testing. We try to generate test data that can cover all feasible paths of each program under test. The final results are listed in Table 7.

As can be seen from Table 7 that, (1) for each program under test, the proposed method can achieve the highest coverage rate out of these three methods. The average coverage rate of the proposed method is 93.28%, while those of traditional genetic algorithm and the random method are only 87.64% and 70.55%, respectively; (2) the highest coverage rate of the proposed method is 100.0%, and the lowest coverage rate of the proposed method is 90.2%, whereas those of traditional genetic algorithm are 90.8% and 80.6%, and those of the random method are 84.7% and 58.4%, respectively.

We also use the statistical tool, Spss, to conduct $T-$test for the comparison of different methods, and The experimental results of $T-$test for path coverage testing are listed in Table 8.

As can be seen from Table 8 that, the value of t_1 is 9.521, and that of t_2 is 7.860. The values of both statistics are much higher than the bound, $t_{0.01} = 2.5083$. So we can conclude that for path coverage testing, the coverage rate of the proposed method is significantly greater than traditional genetic algorithm and the random method.

From the above experimental results we can conclude that, for all testing methods, the test data generated according to test adequacy criterion 2 can generate test data with a higher

TABLE 6. *T*-test results of branch coverage testing.

Null hypothesis	Statistic	Rejection domain	The value of statistic	Result
$X \leq Y$	$T_1 = \frac{X-Y}{S_\omega^1/\sqrt{n_1+n_2-2}}$	$T_1 > t_\alpha(22) = 2.5083$	$t_1 = 9.813$	$X > Y$
$X \leq Z$	$T_2 = \frac{X-Z}{S_\omega^2/\sqrt{n_1+n_3-2}}$	$T_2 > t_\alpha(22) = 2.5083$	$t_2 = 10.711$	$X > Y$

TABLE 7. Results for path coverage testing.

program	No. of test data	Proposed method(%)	Traditional genetic algorithm(%)	Random method(%)
1	8	94.6	90.2	84.7
2	72	90.2	80.6	82.3
3	56	92.2	85.3	81.2
4	21	90.9	83.2	78.4
5	97	91.6	87.1	63.6
6	80	95.4	90.4	58.4
7	64	91.2	86.6	69.8
8	53	92.3	90.8	67.5
9	190	91.9	86.2	64.8
10	508	96.2	91.7	62.4
11	265	100.0	93.2	71.3
12	430	92.8	86.4	62.2
Average	153.67	93.28	87.64	70.55
Standard	deviation	2.8039	3.7086	8.9893

coverage rate than traditional genetic method and the random method. So we think the proposed method is of effectiveness.

2) EXPERIMENTAL RESULTS OF SECOND GROUP OF EXPERIMENTS

a: RESULTS FOR STATEMENT COVERAGE TESTING

We generate test data that can cover all target statements of each program under test by our method, traditional genetic algorithm and the random method, respectively. The final results are listed in Table 9.

As can be seen from Table 9 that, (1) On average, the random method needs more evaluations to generate test data satisfying test adequacy criterion 2 than our method and the traditional genetic algorithm, and the traditional genetic algorithm needs more evaluations than our our method. (2) The least number of evaluations of our method is 2574. For the same program, the evaluations of traditional genetic method is about 1.25 times of that of our method, and the random method is about 1.45 times of our method. The greatest number of evaluations of our method is 78546. For the same program, the evaluation of traditional genetic method is about 1.04 times of that of our method, and the random method is about 1.27 times of our method. (3) the coverage rates of the two methods are not quite different. The greatest coverage rate of our method is 99.8%, while that of traditional genetic algorithm is 98.1%, and that of the random method is 97.2% for the same program; the least coverage rate of our method is 90.7%, whereas that of traditional genetic algorithm is the same, and that of the random method is 89.9% for the same program.

b: RESULTS FOR BRANCH COVERAGE TESTING

The experimental results for branch coverage testing are listed in Table 10.

As can be seen from Table 10 that, (1) For branch coverage testing, the experimental results are similar with that of statement coverage testing. For most programs, our method needs less evaluations than the other two methods. In addition, there is no obvious difference in the coverage rate for the three methods. (2) The least number of evaluations of our method is 2352. For the same program, the evaluations of traditional genetic method is about 1.13 times of that of our method, and the random method is about 1.52 times of our method. The greatest number of evaluations of our method is 46952. For the same program, the evaluation of traditional genetic method is about 1.06 times of that of our method, and the random method is about 1.24 times of our method. (3) the coverage rates of the two methods are not quite different. The greatest coverage rate of our method is 98.3%, while that of traditional genetic algorithm is 97.8%, and that of the random

TABLE 8. *T*-test results of path coverage testing.

Null hypothesis	Statistic	Rejection domain	The value of statistic	Result
$X \leq Y$	$T_1 = \frac{X-Y}{S_\omega^1/\sqrt{n_1+n_2-2}}$	$T_1 > t_\alpha(22) = 2.5083$	$t_1 = 9.521$	$X > Y$
$X \leq Z$	$T_2 = \frac{X-Z}{S_\omega^2/\sqrt{n_1+n_3-2}}$	$T_2 > t_\alpha(22) = 2.5083$	$t_2 = 7.860$	$X > Y$

TABLE 9. Results for statement coverage testing.

program	our method		traditional genetic algorithm		random method	
	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate
1	2574	0.967	3218 0.964	3730	0.961	
2	3562	0.962	4257	0.961	5693	0.968
3	3923	0.922	4216	0.912	4663	0.907
4	4245	0.938	5064	0.931	6234	0.931
5	3245	0.943	3647	0.945	4435	0.937
6	5245	0.964	6059	0.952	7345	0.935
7	7835	0.998	9578	0.981	12652	0.972
8	9345	0.907	9547	0.907	12056	0.899
9	41054	0.931	49513	0.912	61284	0.857
10	78546	0.947	81456	0.934	100125	0.925
11	59454	0.954	61547	0.923	78954	0.893
12	56241	0.917	65481	0.914	85462	0.908
Average	22939.1	0.946	25298.6	0.936	31886.1	0.924

TABLE 10. Results for branch coverage testing.

program	our method		traditional genetic algorithm		random method	
	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate
1	934	0.967	1058	0.956	1423	0.952
2	2352	0.903	2567	0.907	3274	0.886
3	4724	0.946	5367	0.932	6273	0.945
4	3943	0.968	4005	0.955	4276	0.942
5	3425	0.956	4567	0.947	5263	0.951
6	5355	0.949	6249	0.942	7344	0.926
7	2522	0.983	2875	0.978	3163	0.973
8	4252	0.938	5345	0.925	6257	0.903
9	25463	0.941	31579	0.916	42643	0.892
10	46952	0.916	49567	0.904	58438	0.913
11	39547	0.943	42594	0.927	49534	0.923
12	34575	0.951	38562	0.919	43524	0.887
Average	14503.67	0.947	16194.6	0.934	19284.3	0.924

TABLE 11. Results for path coverage testing.

program	our method		traditional genetic algorithm		random method	
	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate	# of individual evaluation	coverage rate
1	845	0.942	1265	0.932	1853	0.923
2	2173	0.901	2397	0.899	2724	0.902
3	2374	0.936	3162	0.928	3724	0.924
4	3022	0.905	3648	0.911	4264	0.901
5	4583	0.912	5419	0.908	7356	0.901
6	4033	0.952	4591	0.937	5382	0.936
7	9445	0.909	10197	0.904	12765	0.904
8	62343	0.928	75344	0.917	93035	0.912
9	36874	0.919	46978	0.924	53676	0.935
10	56737	0.932	64257	0.929	89464	0.897
11	48244	0.954	55641	0.947	67975	0.936
12	37448	0.915	45367	0.927	57457	0.925
Average	22343.4	0.925	26522.2	0.922	33306.3	0.916

method is 97.3% for the same program; the least coverage rate of our method is 90.3%, whereas that of traditional genetic algorithm is 90.7%, and that of the random method is 88.6% for the same program.

c: RESULTS FOR PATH COVERAGE TESTING

The last experiments are conducted for path coverage testing. We try to generate test data to cover all target paths by the evolutionary method and the random method, and the experimental results are listed in Table 11.

As can be seen from Table 11 that, (1) For path coverage testing, the experimental results are also similar with that of statement coverage testing and branch coverage testing. (2) The least number of evaluations of our method is 2173. For the same program, the evaluations of traditional genetic method is about 1.50 times of that of our method, and the random method is about 2.19 times of our method. The greatest number of evaluations of our method is 56737. For the same program, the evaluation of traditional genetic method is about 1.13 times of that of our method, and the random method is about 1.58 times of our method. (3) The greatest coverage rate of our method is 95.4%, while that of traditional genetic algorithm is 94.7%, and that of the random method is 93.6% for the same program; the least coverage rate of our method is 90.1%, whereas that of traditional genetic algorithm is 89.9%, and that of the random method is 90.2% for the same program.

From the above experimental results we can conclude that, the genetic algorithm can greatly improve the efficiency of generating test data for programs with random numbers comparing with the random method. In addition, the propose method proves the efficiency of traditional genetic algorithm further.

VII. CONCLUSION

Test is an important means to improving the quality of software before delivery and usage, in which a crucial problem is the generation of effective test data using suitable theories and methods. Existing test methods are mainly designed for conventional software. In fact, there are programs containing random or other uncertain parameters in practice. Traditional test adequacy criteria cannot be applicable to those programs.

This paper intends to study the problem of test data generation for software with random numbers. First, we propose a novel test adequacy criterion for software with random numbers. Second, an optimal model of test data generation for software with random numbers is established. Finally, we present an evolutionary algorithm to solve the optimization model. The experimental results illustrate that the proposed method can solve the problem of test data for software with random numbers.

The test adequacy criteria proposed in this paper can enrich the theory of software test and improve the test effect and efficiency of software with random numbers. The set-based genetic algorithm is used to generate test data, which can further expand the scope of application of evolutionary optimization. So this paper has important theoretical significance and practical application value. The research of this paper can provide useful help and inspiration to the software testing with random numbers.

Of course, there are still some shortcomings in this paper. First of all, the proposed test criteria need to be further improved in practice in order to better improve the quality of software. Whether our proposed criteria can find more defects than the traditional criteria needs to be further verified. In addition, our proposed test data optimization model also needs to be improved to make the generated test data have better quality. In addition, we use genetic algorithm to solve the established model, although the algorithm is obviously superior to the traditional genetic algorithm and random method. However, can we find a better algorithm to solve this problem. The above aspects are also the topics we want to further study in the future.

ACKNOWLEDGEMENT
This work has been helped by Mark Harman of University College London, which has great effect on improving the quality of this article greatly.

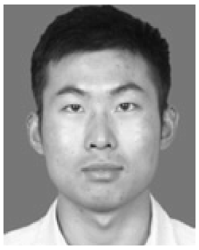
REFERENCES
[1] L. M. Surhone, M. T. Tennoe, and S. F. Henssonow, *Software Quality Assurance*. Whitefish, MT, USA: Betascript Publishing, Sep. 2010.
[2] M. Böhme, "STADS: Software testing as species discovery," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 2, pp. 1–52, Jun. 2018.
[3] B. Beizer, *Software Testing Techniques*. New York, NY, USA: Van Nostrand Rheinhold, Jan. 1990.
[4] B. W. Boehm, *Characteristics of Software Quality*. Amsterdam, The Netherlands: North Holland, , 1978.
[5] P. Arcaini, A. Gargantini, and E. Riccobene, "Combining model-based testing and runtime monitoring for program testing in the presence of nondeterminism," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2013, pp. 1–11.
[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *J. Syst. Softw.*, vol. 101, pp. 193–220, Mar. 2015.
[7] R. Prado, P. Souza, and S. Souza, "Valipar service: Structural testing of concurrent programs as a Web service composition," in *Advances in Intelligent Systems and Computing*, vol. 448. Apr. 2016, pp. 581–590.
[8] M.-Z. Zhang, Y.-Z. Gong, Y.-W. Wang, and D.-H. Jin, "Unit test data generation for c using rule-directed symbolic execution," *J. Comput. Sci. Technol.*, vol. 34, no. 3, pp. 670–689, May 2019.
[9] G. J. Myers, T. Badgett, and T. M. Thomas, *The Art of Software Testing*. Hoboken, NJ, USA: Wiley, 2004.
[10] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, A. Bertolino, J. Jenny Li, and H. Zhu, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, Aug. 2013.
[11] A. Shahbazi, A. F. Tappenden, and J. Miller, "Centroidal Voronoi tessellations-a new approach to random testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 163–183, Feb. 2013.
[12] K. P. Chan, T. Y. Chen, and D. Towey, *Restricted Random Testing*, vol. 2349. Springer-Verlag, 2002, pp. 321–330.
[13] D. Baca, B. Carlsson, K. Petersen, and L. Lundberg, "Improving software security with static automated code analysis in an industry setting," *Softw., Pract. Exper.*, vol. 43, no. 3, pp. 259–279, Mar. 2013.
[14] Z. Zhu, L. Jiao, and X. Xu, "Combining search-based testing and dynamic symbolic execution by evolvability metric," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2018, pp. 58–68.
[15] X. Wang, R. Zhao, and L. Li, "Program slice algorithm for test data generation," *J. Comput. Appl.*, vol. 25, no. 6, pp. 1445–1447, Jun. 2005.
[16] W. Miller and D. L. Spooner, "Automatic generation of floating-point test data," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 3, pp. 223–226, Sep. 1976.
[17] B. Korel, "Automated software test data generation," *IEEE Trans. Softw. Eng.*, vol. 16, no. 8, pp. 870–879, 1990.
[18] A. Salahirad, H. Almulla, and G. Gay, "Choosing the fitness function for the job: Automated generation of test suites that detect real faults," *Softw. Test., Verification Rel.*, vol. 29, nos. 4–5, pp. 4–5, Jun. 2019.
[19] M. Harman, "Search based software testing for Android," in *Proc. IEEE/ACM 10th Int. Workshop Search-Based Softw. Test. (SBST)*, May 2017, pp. 1–2.
[20] P. R. Srivastava, R. Khandelwal, S. Khandelwal, S. Kumar, and S. Santebennur Ranganatha, "Automated test data generation using cuckoo search and tabu search (CSTS) algorithm," *J. Intell. Syst.*, vol. 21, no. 2, pp. 195–224, Jan. 2012.
[21] W. Zhang, Y. Qi, and D. Li, "Test case prioritization based on discrete particle swarm optimization algorithm," *J. Comput. Appl.*, vol. 37, no. 1, pp. 108–113, Jan. 2017.
[22] E. Alba and F. Chicano, "Observations in using parallel and sequential evolutionary algorithms for automatic software testing," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3161–3183, Oct. 2008.
[23] J. H. Andrews, T. Menzies, and F. C. H. Li, "Genetic algorithms for randomized unit testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 1, pp. 80–94, Jan. 2011.
[24] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–61, 2012.
[25] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. Mcminn, P. Tonella, and T. Vos, "Symbolic search-based testing," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 53–62.
[26] A. Pachauri and G. Srivastava, "Towards a parallel approach for test data generation for branch coverage with genetic algorithm using the extended path prefix strategy," in *Proc. 2nd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2015, pp. 110–122.
[27] P. M. S. Bueno and M. Jino, "Automatic test data generation for program paths using genetic algorithms," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 12, no. 06, pp. 691–709, Dec. 2002.
[28] J.-C. Lin and P.-L. Yeh, "Automatic test data generation for path testing using GAs," *Inf. Sci.*, vol. 131, nos. 1–4, pp. 47–64, Jan. 2001.
[29] A. Watkins and E. M. Hufnagel, "Evolutionary test data generation: A comparison of fitness functions," *Softw., Pract. Exper.*, vol. 36, no. 1, pp. 95–116, Jan. 2006.
[30] J. Mei and S. Y. Wang, "An improved genetic algorithm for test cases generation oriented paths," *Chin. J. Electron.*, vol. 23, no. 3, pp. 494–498, Jul. 2014.
[31] X. Yao, D. Gong, and W. Wang, "Test data generation for multiple paths based on local evolution," *Chin. J. Electron.*, vol. 24, no. 1, pp. 46–51, Jan. 2015.
[32] P. J. Boland, H. Singh, and B. Cukic, "Comparing partition and random testing via majorization and Schur functions," *IEEE Trans. Softw. Eng.*, vol. 29, no. 1, pp. 88–94, Jan. 2003.

XIANGJUAN YAO received the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, in 2011.

She is currently a Professor and a Ph.D. Advisor with the School of Mathematics, China University of Mining and Technology. Her main research interests include search-based software testing and evolutionary computation.

DUNWEI GONG (Member, IEEE) received the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, in 1999.

He is currently a Professor and a Ph.D. Advisor with the School of Information and Electrical Engineering, China University of Mining and Technology. His main research interests include evolutionary computation, intelligence optimization, and data mining.



BIN LI received the M.S. degree in computer science from the School of Mathematics, China University of Mining and Technology, in 2016.

He is currently an Engineer. His research interests include software engineering and scheduling optimization.

XIANGYING DANG received the M.S. degree in computer science from the School of Information Engineering, Jiangnan University, in 2008. She is currently pursuing the Ph.D. degree with the School of Information and Electronic Engineering, China University of Mining and Technology. Her research interests include software engineering-based search, and mutation testing and analysis.

GONGJIE ZHANG received the Ph.D. degree in computer application technology from the China University of Mining and Technology, in 2017.

He is currently a Lecturer with the School of Computer Science and Technology, Jiangsu Normal University. His main research interests include software testing and semantic Web.

• • •