IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 16, 2021, accepted June 2, 2021, date of publication June 14, 2021, date of current version June 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3089196

An Evolutionary Generation Method of Test Data for Multiple Paths Based on Coverage Balance

SHUPING FAN[1], NIANMIN YAO[2], LI WAN[3], BAOYING MA[4], AND YAN ZHANG[1]

[1]School of Computer and Information Technology, Mudanjiang Normal University, Mudanjiang 157011, China
[2]School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China
[3]Department of Intelligence and Computing, Tianjin University, Tianjin 300350, China
[4]School of Health Management, Mudanjiang Medical University, Mudanjiang 157011, China

Corresponding authors: Baoying Ma (mabaoying6688@163.com) and Nianmin Yao (lucos@dlut.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0100300; in part by the Innovation Foundation of Science and Technology of Dalian under Grant 2018J12GX045; in part by the Scientific and Technological Plan Project of Mudanjiang City under Grant HT2020JG049; in part by the Natural Science Foundation of Heilongjiang Province under Grant LH2020F038, and in part by the Science and Technology Research Project of Mudanjiang Normal University (Research on Optimization Method of Test Cases in Regression Testing).

ABSTRACT Test data generation is one of the main tasks of software testing. The goal of test data generation based on search algorithms is to automate the task and find test data that meet test criteria. In this study, an evolutionary generation method for test data that cover multiple paths is proposed. Firstly, the method obtains the coverage balance for each target path based on the number of individuals traversing the true and false branches of branch nodes, and calculates the individual's influence on coverage balance before and after an individual joining based on our previous work. Then, according to the number of branch nodes on each target path, the weights of different target paths are designed to obtain the individual fitness to adjust the evolution process and quickly generate test data covering multiple target paths. Finally, the proposed method is compared with existing techniques. Experimental results of benchmark programs and industrial use cases show that the proposed method can effectively improve the efficiency of test data generation for multiple paths.

INDEX TERMS Keywords software testing, test data generation, multi-path coverage, genetic algorithm, coverage balance.

I. INTRODUCTION

Software testing is an important verification method in software development. Its goal is to use the least test data to find as many faults as possible on the premise of satisfying the test criteria, thereby reducing the cost of software development [1], [2]. Software testing is divided into white box testing, black box testing, and gray box testing between the two according to the degree of visibility of software code [3], [4]. In white box testing, compared with statement coverage and branch coverage, the path coverage criterion is more effective, which means that each distinct path in the program is executed at least once [4]. It is very difficult to search the test data that traverse each path, because the paths in the program may be infinite in the case of loops, or infeasible in the case of nested branches. As a result, different techniques have been proposed to search for test data that can achieve high path coverage [5], [6].

As software becomes more and more complex, software testing becomes more and more challenging. Search-based technology is used to search for data automatically. For example, Particle Swarm Optimization (PSO) [7], [8], Difference Evolutionary algorithm (DE) [9]–[11], Artificial Bee Colony algorithm (ABC) [12], [13], Firework Algorithm (FA) [14] are applied by scholars. They convert test data generation problems into combinatorial optimization problems and evolve to generate test data that meet the requirements. In recent years, genetic algorithm (GA) has also been proven to be effective to generate test data. Up to now, GA is not only used for target-oriented software test data generation [15] but also object-oriented test data generation [16]. It can also be applied to embedded systems [17]. Besides, GA is also the preferred algorithm for solving new problems in software engineering [18].

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

In the generation of test data for path coverage, it is often concerned with data generation covering multiple target paths, that is, to achieve a higher path coverage. However, the times the true and false branches of branch statements traversed by data are not considered. As a result, some branches are traversed many times, while other ones are not traversed by data or traversed by few data, which will lead to unbalanced coverage of program branches, and if these branches containing many faults are traversed by less data, generated data is difficult to detect the faults.

Therefore, we have proposed an evolutionary method of generating test data for path coverage based on balance optimization theory in the previous work [19]. Employing the method in [19], test data can be generated effectively. However, this method is only suitable for the case of one target path. Besides, the algorithm needs to be run multiple times to generate test data covering multiple target paths, so the cost is large. In this paper, we focus on the problem of generating test data for covering multiple paths. The goal of the research presented here is to generate test data traversing multiple target paths to reduce the cost. To achieve this goal, first, we present a test data generation model for multi-target paths, and the constraint requires that the traversed path is just one of the target paths. Then, we give the process to generate test data by using GA. Finally, we apply our method to some typical programs, and the experimental results confirm that our method can efficiently generate test data that cover target paths. The main contributions of this study are as follows: 1) we establish a mathematical model for multi-path test data generation problem; 2) we introduce an evolutionary optimization algorithm to generate test data for multiple paths effectively and present a function to get the fitness of each individual; 3) we apply the proposed method in different programs to confirm its effectiveness.

The rest of this paper is organized as follows: Section 2 reviews related work; Section 3 introduces a mathematical model for the problem of generating test data for multiple paths; Section 4 describes how to calculate the fitness of each individual to generate test data using GA; The applications of our method in benchmark programs and some programs of SIR are given in Section 5; Section 6 concludes our work and give the future research.

II. RELATED WORK

A. GENERATING TEST DATA FOR ONE TARGET PATH

With the rapid growth of software, a large amount of data needs to be generated during testing. The effectiveness of testing methods depends on the standards considered in the generation of test data. In fact, many software testing problems can be attributed to path coverage test data generation problems [20], [21].

In recent years, there have been many research results for test data generation. Liao *et al.* combined the automatic path segmentation method and artificial fish school algorithm, and proposed a test data generation method for path coverage [22]. Arcuri presented a novel search algorithm for generating a test suite, in particular for system testing. They carried out an empirical study to compare the proposed method with other ones. The results show that the proposed method has the best overall performance, but not the best on all problems [18]. Mohammad *et al.* applied the competitive algorithm to generate test data and evaluated the effectiveness of the proposed method according to path coverage, discovered failures as well as cost function [21]. Nikravan *et al.* introduced a new standard for white box testing-domain coverage. The method uses an incremental method to detect subregions, which cover the longest sub-path from the beginning of a given path. The detected sub-domains are further subdivided, and this process is repeated until the generated test data completely cover the path [23]. Dang *et al.* investigated the correlation between the true branches of the variant sentence and proposed a method for executable path generation for weak mutation testing. It enables test data covering these paths to kill all variants [24]. Mao *et al.* applied dynamic program slicing in test data generation and calculated dynamic slicing of points of interest variables in the program to obtain the current values for them. Besides, the minimization method is used to adjust the input of the program in the branch function [25]. Ding *et al.* proposed a complete model for rapid generation of test data from the perspective of software testing engineering practice, which gave the representation of key point paths and improved the efficiency for test data generation [26]. Bidgoli *et al.* presented a new method based on swarm intelligence to cover the program path. In the method, ant colony algorithm and particle swarm algorithm are applied, and the test data covering the prime path are generated by designing a normalized fitness function [27].

Currently, there are also some works on the application of heuristic optimization algorithms to automatically generate test data, among which GA is the most popular [28]. Cao *et al.* aimed to solve the premature and easy to fall into the local optimal solution problem of GA in the automatic generation of test data, and introduced intelligent mutation in GA to optimize the execution strategy of the traditional genetic operator [29]. Xue also improved the GA, and analyzed the fuzzy test data generation method from three aspects: fuzzy test input variable adjustment, test data initial population formation, and test data mutation rate position control. Experimental results show that the improved GA has higher efficiency [30]. Gao *et al.* improved the traditional genetic algorithm. In the automatic generation of test data, adaptive crossover operator, mutation operator and simulated annealing mechanism are applied to accelerate the optimization process of test data [31]. The calculation of the individuals' fitness is the key to GA. However, most of the fitness functions in the methods above are designed for one single target path, which will cause test data that only traverse one path are generated by GA for running one time. For many paths to be covered, if the methods for one target path are followed, the GA must be run one by one to generate test data covering each path in turn. As a result, the efficiency of these methods will be greatly reduced. It is necessary to further improve the efficiency of test data generation.

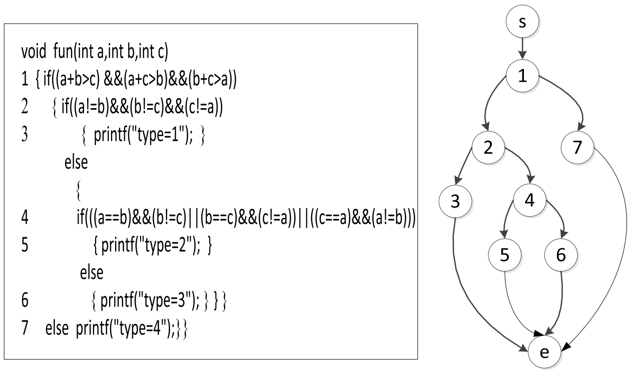
B. GENERATING TEST DATA FOR MULTIPLE PATHS

To solve the problem of test data generation for multi-target paths, Ahmed *et al.* proposed to convert the test data generation problem into a multi-objective optimization one and verified that the proposed method is superior to the methods of test data generation for the one path by experiments [32]. However, the fitness function used to evaluate individuals includes two parts: branch distance and layer proximity. When faced with complex programs, the calculation of branch distance will be too complicated to accept. Chen *et al.* proposed a multi-group GA called MPGA for path testing. In the method, the fitness function is the sum of branch predicate distance between the target path and the actual path. The triangle classification program is used to verify the efficiency of the method. Experimental results show that MPGA is more effective than the ordinary single population method [33]. Zhang *et al.* introduced the concept of multiple groups based on the low search efficiency of single population GA, and a parallel evolutionary algorithm named IPEA based on GA is presented. Besides, the search time, coverage and test case scale of the algorithm are all verified by experiments [34]. Considering the fault detection ability by generated test data, Zhang *et al.* proposed a fault-oriented method for multi-path coverage [28]. In the method, the path that an individual traverses must be one of the target paths is taken as a constraint. The number of faults and the risk degree of the faults found by an individual are considered as two evolutionary goals. So the problem of test data generation is transformed into a multi-objective optimization one with constraints, and the effectiveness of the proposed method is verified by experiments. Tian *et al.* studied the problem of multi-path coverage for uncertain execution of parallel programs [35]. Based on each given path and its equivalent path, the test data generation problem for multi-path coverage was modeled as an optimization one containing multiple goals. Zhang *et al.* proposed a method to generate test data covering multiple paths using GA combined with a method of reducing the input domain of a program [36]. Using the method, independent sub-paths of target paths are first extracted. Then the input variables corresponding to these traversed sub-paths are kept fixed, and the scope of crossover and mutation operations is reduced to ensure that these sub-populations are searched in a reduced input field. As a result, the efficiency of test data generation is improved. Gong *et al.* used the Huffman coding to encode the target paths and studied a method of test data generation based on it [37]. The fitness function considers the matching degree and the number of consecutive identical nodes between the path that an individual traverses and each target path. The method effectively guides the evolution process of GA and improves the efficiency of test data generation. However, when the target path is complex, generating test data will require a large amount of calculation and it is time-consuming. Besides, the efficiency of the algorithm needs to be further improved.

C. STATEMENT OF THE PROBLEM

Generating some test data that satisfied the required adequacy criterion is one of the main tasks of software testing. When generating test data evolutionarily, existing methods often design individual fitness by comparing the similarity between the test data traversal path and the target path [32], [37], without considering the fact that the true and false branches of the program branch nodes are traversed by the data. As a result, the number of generated test data that traverses some branch nodes is too much, and the number traversing other branch nodes is small or even no data traverses. Although test data that cover the target paths can be generated, it may cause the test data to cover the program branches unevenly. If the branch with less data traverse contains a large number of faults, it may fail to detect some of the faults. Software testing aims to find and modify faults in the program under test [28]. So it is very important to generate test data that can expose the faults of a program.

The following example illustrates the necessity of introducing coverage balance in the evolutionary generation of multi-target test data. According to the control flow diagram in Figure 1(b), assuming $p_1 = \{s, 1, 7, e\}$ and $p_2 = \{s, 1, 2, 3, e\}$ are not target paths, $p_3 = \{s, 1, 2, 4, 5, e\}$ and $p_4 = \{s, 1, 2, 4, 6, e\}$ are, and the initial population size of the GA is $m = 10$, where the i *th* individual is $x_i (1 \leq i \leq m)$.


```
void  fun(int a,int b,int c)
1 { if((a+b>c) &&(a+c>b)&&(b+c>a))
2    { if((a!=b)&&(b!=c)&&(c!=a))
3         { printf("type=1"); }
       else
         {
4        if(((a==b)&&(b!=c)||(b==c)&&(c!=a))||((c==a)&&(a!=b)))
5           {printf("type=2"); }
          else
6           { printf("type=3"); } } }
7   else  printf("type=4");}}
```

(a) The Source Code (b) The Control Flow diagram

FIGURE 1. Sample program.

Using the methods in [32] and [37], in the generation of test data, an individual will be retained according to the matching degree between the individual's traversal path and the target path. The more similar these two paths are, the greater the probability that the individual will be retained. Therefore, in the generation of test data, a large amount of data will cover the nodes on the target path, and a small amount of data or no data will cover the nodes on the non-target path. That is to say, in Figure 1(b), there will be a large amount of data covering n_2 and n_4, and a small number of data covering n_3 and n_7. Therefore, suppose we may get the number of individuals traversing different nodes in Table 1.

TABLE 1. Test data traversing branch nodes.

Nodes	Individuals	Number
n_1	$x_1 \sim x_{10}$	10
n_2	$x_1 \sim x_8$	8
n_3	—	0
n_4	$x_1 \sim x_8$	8
n_7	x_9, x_{10}	2

It can be seen from Table 1 that the true and false branches of n_2 are covered unevenly by 8 individuals of the population, the result is that 8 individuals are traversing the branch where n_4 is located, that is, test data may cover the target paths including n_4. The number of data covering n_3 is zero, that is, no data is traversing the node, which will result in the inability to generate test data covering p_2. In addition, there are only two data traverse p_1. It can be seen that if these two paths contain multiple faults, the imbalance of test data covering the program is more prominent, which will greatly reduce the efficiency of fault detection.

If the balance of individuals covering different paths is considered when generating test data for multiple paths, it will undoubtedly accelerate the evolutionary generation of test data. So we have proposed an evolutionary method of generating test data for path coverage based on balance optimization theory in the previous work [19]. Employing the method in [19], test data covering one target path can be generated effectively. However, this method is only suitable for the case of one target path. As various real-world programs often contain lots of target paths, we have to run the method many times to generate data for them, resulting in a waste of time and resources. So it is necessary to propose an effective method to generate data for multiple paths.

III. THE PROBLEM MODEL OF TEST DATA GENERATION FOR MULTI-PATH COVERAGE

When GA is used to generate test data that traverse multiple target paths, existing works usually convert this problem into one of minimizing function, thereby converting a complex multi-objective optimization problem into multiple relatively simple sub-problems. Reference [28] considered the number of faults detected and the severity of these faults and presented a model of test data generating for multi-target paths. Reference [36] grouped target paths according to the same independent sub-paths in the target path. Reference [38] formulated the model of test case generation and its goal is to minimize the number of generated test cases after considering complete path coverage. In our method, we convert the problem into a maximum problem. Assuming that the target paths of the program under test are $p_1, p_2, \cdots p_N$, where N is the number of target paths. The input of the program is selected as the decision variable, that is, $x = x_1, x_2, \cdots x_m$, where m is the number of input data required when running the program, that is, the initial population size of GA. $x_i (i = 1, 2, \cdots, m)$ is the component of the input data, and these variables are all set to integers. Supposing that the path traversed by the decision variable x is $p(x)$, $|p_j|$ represents the number of branch nodes on p_j. Starting from the first node of $p(x)$, compare whether the nodes of $p(x)$ and p_j are the same, and calculate the number of identical nodes, denoted as $Sam(x)$. The mathematical model of test data generation problem for multi-path coverage can be expressed as:

$$max\, f(x,t) = \left(\left(\sum_{j=1}^{N} CB'_j(x,t) - \sum_{j=1}^{N} CB_j(t)\right) * Wet_j\right) \Big/ N$$
$$s.t.\ \vee_{j=1}^{N} \frac{Sam(x)}{|p_j|} = 1 \tag{1}$$

where $\sum_{j=1}^{N} CB'_j(x,t)$ and $\sum_{j=1}^{N} CB_j(t)$ represent the coverage balance before and after adding the decision variable x respectively. Wet_j indicates the ratio of the number of branch nodes on the target path p_j to all target paths, as is shown in (6) - (8), and $\vee$ means the maximum value. $\vee_{j=1}^{N} \frac{Sam(x)}{|p_j|} = 1$ is the constraint condition, which means that the path that x traverses is close to the target path p_j. As the goal of our method is to find test data that traverse target paths, when the constraint condition takes its maximum value of 1. That is, when the individual x_i traverses p_j, $\frac{Sam(x_i)}{|p_j|} = 1$, and $\vee_{j=1}^{N} \frac{Sam(x)}{|p_j|} = 1$.

It can be observed from (1) that the constraint of (1) requires that the traversed path by x is equal to one of the target paths. Previous studies have not considered the coverage balance. Therefore, test data generation efficiency is not high. On the contrary, our mathematical model given here takes the coverage balance and the number of branch nodes on the target path into account to generate test data rapidly. We say from this point that a balanced coverage of our model is more evident.

IV. EVOLUTIONARY GENERATION PROCESS OF TEST DATA FOR MULTIPLE PATHS

We use GA to solve (1). The fitness of an individual is determined to compare the quality of different individuals. So we will first give a method of calculating the fitness of an individual when solving (1).

A. DIFFERENCE BETWEEN THE PROPOSED METHOD AND THE PREVIOUS WORK

To generate test data that cover the target path rapidly, the method in [19] makes good use of the balance by individuals traversing the program to adjust the evolutionary process of GA. In the method, the program balance is proposed and designed to realize coverage balance. Besides, each individual's fitness is calculated according to the influence on the balance of the test cases covering the program before and after the individual joining, and an individual with high influence has a bigger fitness to have a greater chance to participate in subsequent evolution. The method of test data generation in [19] is shown in Algorithm 1. Although it improves the efficiency of test data generation, there are still some limitations. Our proposed method further studies the limitations of the previous work in [19]. Different from the method in [19], the innovation of the proposed method is as follows.

(1) We proposed a method of test data generation based on GA suitable for coverage of multiple target paths.

(2) The weight of different target paths is considered to accelerate the evolutionary generation of test data.

(3) The results verified by experiments show that the proposed method not only improves the efficiency of test data generation for multi-target paths but also effectively reduces the test cost.

The proposed method is shown in algorithm 2. First, all values of control parameters are assigned, target paths are selected randomly and initial data corresponding to different individuals are generated randomly.

By comparison of algorithm 1 and algorithm 2, it can be seen that test data covering N target paths can both be generated by using the two methods. We can see from algorithm 1 that GA needs to be run N times to generate test data for covering these paths since only test data covering one path can be obtained by running one time.

B. THE FITNESS OF INDIVIDUALS

1) CALCULATION OF BRANCH BALANCE FOR EACH TARGET PATH

After running the program by all individuals corresponding to the data, the branch balance is used to indicate the difference between the number of individuals traversing the true and false branches of a branch node [19]. Then the coverage balance is calculated before and after an individual joins basing on branch balance.

To calculate branch balance, the coverage of different program branches is obtained after all test data run the program. Since the loop structure and the switch statement can both be expressed as a double-branch selection structure [39], we study the situation where each branch node has two branches in the paper. When GA runs to the t th generation, denote x_i and p_j as the i th individual and j th target path respectively, and $Cov_{jkT}(x_i, t)$ represents whether x_i covers the true branch of the k th branch node on p_j, and it can be described as follows:

$$Cov_{jkT}(x_i, t) = \begin{cases} 1, & x_i \text{ covers the true branch of} \\ & \text{the kth branch node on } p_j \\ 0, & else \end{cases} \tag{2}$$

The same, assuming $Cov_{jkF}(x_i, t)$ indicates whether x_i covers the false branch of the k th branch node on p_j, and it can be expressed as follows:

$$Cov_{jkF}(x_i, t) = \begin{cases} 1, & x_i \text{ covers the false branch of} \\ & \text{the kth branch node on } p_j \\ 0, & else \end{cases} \tag{3}$$

Assuming currently that the number of data in initial population is m, after all the data run the program, we can express the branch balance as follows:

$$BL_{jk}(t) = \begin{cases} 0, & \sum_{i=1}^{m} Cov_{jkT}(x_i, t) = \sum_{i=1}^{m} Cov_{jkF}(x_i, t) = 0 \\ \dfrac{\left|\sum_{i=1}^{m} Cov_{jkT}(x_i, t) - \sum_{i=1}^{m} Cov_{jkF}(x_i, t)\right|}{\left(\sum_{i=1}^{m} Cov_{jkT}(x_i, t) + \sum_{i=1}^{m} Cov_{jkF}(x_i, t)\right)}, & else \end{cases} \tag{4}$$

The same, after all the data except $x_w (1 \leq w \leq m)$ run the program, we can express the branch balance as follows:

$$BL'_{jk}(x_w, t) = \begin{cases} 0, & \sum_{i=1, i\neq w}^{m} Cov_{jkT}(x_i, t) = \sum_{i=1, i\neq w}^{m} Cov_{jkF}(x_i, t) \\ \dfrac{\left|\sum_{i=1, i\neq w}^{m} Cov_{jkT}(x_i, t) - \sum_{i=1, i\neq w}^{m} Cov_{jkF}(x_i, t)\right|}{\left(\sum_{i=1, i\neq w}^{m} Cov_{jkT}(x_i, t) + \sum_{i=1, i\neq w}^{m} Cov_{jkF}(x_i, t)\right)}, & else \end{cases} \tag{5}$$

2) CALCULATION OF COVERAGE BALANCE FOR EACH TARGET PATH

Denote $|p_j|$ as the number of branch nodes on p_j. Calculate the branch balance for each branch node on p_j, and use the sum of the branch balance of all branch nodes on p_j as the coverage balance of t th generation population of GA traversing p_j. When all data run the program, the coverage balance is denoted as $CB_j(t)$, as is shown in (6):

$$CB_j(t) = \sum_{k=1}^{|p_j|} BL_{jk}(t) \tag{6}$$

We can see that $CB_j(t)$ reflects the overall balance of different branches on p_j when all the data join. According to the definition of branch balance in (4), it is not difficult to conclude that the smaller the coverage balance, the better the balance of the individual covering different branches of the program. The same, after all the data except $x_w (1 \leq w \leq m)$ run the program, the coverage balance can be described as follows:

$$CB'_j(x_w, t) = \sum_{k=1}^{|p_j|} BL'_{jk}(x_w, t) \tag{7}$$

3) CALCULATION OF THE WEIGHT FOR EACH TARGET PATH

Considering that the more branch nodes on p_j, the more difficult it is to generate test data for the path. The reason is that the input data at this time have a greater chance of traversing other program branches that deviate from p_j. Therefore, we set that the more branch nodes on p_j, the greater the weight of the individual's fitness to the path, to generate data covering p_j rapidly. The weight of p_j is described as follows:

$$Wet_j = |p_j| / |br| \tag{8}$$

where $|br|$ is the total number of branch nodes on all target paths.

4) CALCULATION OF THE FITNESS OF INDIVIDUALS

According to (6)~(8), the fitness of the w th individual for path p_j can be expressed as:

$$f_j(x_w, t) = \begin{cases} CB'_j(x_w, t) - CB_j(t), & CB'_j(x_w, t) \geq CB_j(t) \\ 0, & else \end{cases} \tag{9}$$

when the number of target paths N is not zero, the fitness of the w th individual can be expressed as:

$$f(x_w, t) = \sum_{j=1}^{N} \left(f_j(x_w, t)^* Wet_j\right) \Big/ N \tag{10}$$

where Wet_j indicates the weight of the fitness obtained by x_w to p_j. It can be seen that $f(x_w, t)$ is used to represent the weighted average of the fitness of x_w for all target paths. We can also see that when $\sum_{j=1}^{N} CB'_j(x_w, t)$ is greater than $\sum_{j=1}^{N} CB_j(t)$, positive data will be obtained. In other words, when x_w is deleted, the coverage balance will increase, and such individuals as x_w should be retained in the evolutionary process.

After all test data in individuals execute the program under test, each individual's fitness is calculated according to (10). Then, selection, crossover and mutation operations will be executed until one of the terminal conditions is obtained. Finally, output the desired test data, their traversed paths.

C. CASE STUDY

To describe the proposed method in detail, the sample program in Fig. 1(a) is taken as the program under test. According to its control flow diagram shown in Fig. 1(b), four target paths are selected randomly: $p_1 = \{s, 1, 7, e\}, p_2 = \{s, 1, 2, 3, e\}, p_3 = \{s, 1, 2, 4, 5, e\}, p_4 = \{s, 1, 2, 4, 6, e\}$.

Assuming that the population size is 10, and GA runs to the t th generation. After the data corresponding to individuals run the program, we can get that whether x_i covers the true branch and false branch of the k th branch node on p_3, denoted as $Cov_{3kT}(x_i, t)$ and $Cov_{3kF}(x_i, t)$ respectively. So the number of data covering the true and false branches of each branch node can be obtained according to it. Assuming the number of individuals traversing the true branch and false branch of n_1 are $\sum_{i=1}^{10} Cov_{31T}(x_i, t) = 6$ and $\sum_{i=1}^{10} Cov_{31F}(x_i, t) = 4$ respectively, and the values of n_2 and n_4 are shown in Table 2. Then the branch balance of p_3 represented as $BL_{3k}(t)$ can

TABLE 2. The case of branch nodes traversed by individuals.

nodes	$\sum_{i=1}^{10} Cov_{3kT}(x_i, t)$	$\sum_{i=1}^{10} Cov_{3kT}(x_i, t)$	$BL_{3k}(t)$
n_1	6	4	0.20
n_2	5	1	0.67
n_4	1	0	1.00

be calculated. $BL_{31}(t) = (6 - 4)/6 + 4 = 0.2$. The same, we can get: $BL_{32}(t) \approx 0.67, BL_{32}(t) \approx 1$, as is shown in Table 2.

Algorithm 1 Test Data Generation Method in [19]

```
Input: Algorithm parameters.
Individuals: $x_1, x_2, \cdots x_m$, maximum generation $G$,
the target path $p$
Output: Test data covering $p$
BEGIN
    $Settings()$;
        // Set each parameter value
    $Initialize(x_w)$;
        // Generate the data of GA
    $generation \leftarrow 0$
Do WHILE$(generation \leq G || path(x_w) \neq p)$
//If the maximum number of iterations is not exceeded or
test data covering $p$ is not generated, then execute the loop
body
  $generation = generation + 1$;
        $Fitness(x_w)$; // Calculate individuals' fitness
        $Select(x_w)$;
        $Cross(x_w)$;
        $Mutate(x_w)$;
    $IF(path(x_w) == p))$ THEN
     $x_w \rightarrow DataSet()$;
    END IF
  END WHILE
END
```

Finally, the coverage balance after all individuals joining can be obtained according to (6). Since there are three branch nodes on p_3, so we can get:

$$CB_3(t) = \sum_{k=1}^{|p_j|} BL_{jk}(t) = \sum_{k=1}^{3} BL_{3k}(t) = (0.2 + 0.67 + 1) = 1.87$$

In the same way, we can get:

$$CB_1(t) = BL_{31}(t) = 0.2, \quad CB_2(t) = 0.87, \quad CB_4(t) = 1.87.$$

After all the data except $x_w(1 \leq w \leq m)$ run the program, we can calculate the $BL'_{3k}(x_w, t)\,(1 \leq k \leq 3)$, as is shown in Table 3. So the coverage balance can be calculated as

Algorithm 2 Test Data Generation of the Proposed Method

```
Input: Individuals: $x_1, x_2, \cdots x_m$, the number
of target paths $N$, maximum generation $G$
Output: Test data covering multiple target paths:
$p = \{p_1, p_2, \cdots, p_N\}$
BEGIN
 DataSet = $\emptyset$
Initialize($x_1, x_2, \cdots x_m$); //Initialize the population
Execute the program, get
 $p(x_k)$ $(k = 1, 2, \cdots, m)$
Do WHILE($generation \leq G || p \neq \emptyset$)
    $generation \leftarrow generation + 1$;
    Do WHILE($k \leq m$)
        IF($p(x_k) \in p$)
            DataSet = DataSet $\cup$ $x_k$
            $p = p - p(x_k)$;
            $k = k + 1$;
        END IF
    Evaluate($x_w$);
    Select($x_w$);
    Cross($x_w$);
    Mutate($x_w$);
   END WHILE
  END WHILE
END
```

TABLE 3. The case of branch nodes traversed by individuals.

nodes	$\sum_{i=1,i\neq w}^{10} Cov_{3kT}(x_i,t)$	$\sum_{i=1,i\neq w}^{10} Cov_{3kT}(x_i,t)$	$BL'_{3k}(x_w,t)$
n_1	6	3	0.33
n_2	5	1	0.67
n_4	1	0	1.00

follows:

$$CB'_3(x_w, t) = \sum_{k=1}^{|p_j|} BL'_{jk}(x_w, t) = \sum_{k=1}^{3} BL'_{3k}(x_w, t)$$
$$= (0.33 + 0.67 + 1) = 2.$$

In the same way, we can get:

$$CB'_1(x_w, t) = BL'_{31}(x_w, t) = 0.33, \quad CB'_2(x_w, t) = 1,$$
$$CB'_4(x_w, t) = 2.$$

According to the number of branch nodes on different paths, the weight of p_j is calculated as follows.

$Wet_3 = |p_j| / |br| = 3/3 = 1$. The same, we can get: $Wet_1 = 1/3$, $Wet_2 = 2/3$, $Wet_4 = 1$. Then we can calculate the fitness of x_w:

$$f(x_w, t) = \sum_{j=1}^{N} \left(f_j(x_w, t) * Wet_j\right)/N$$
$$= \left((0.33 - 0.2) * 1/3 + (1 - 0.87) * 2/3 + (2 - 1.87) * 1 * 2\right)/4$$
$$= 0.0975$$

Since $f(x_w, t) > 0$, this shows that the addition of x_w can improve the coverage balance, and it will be selected first during the evolution process of GA.

V. EXPERIMENTS

To validate the effectiveness of our method, three benchmark programs and three industrial test cases are selected, all of which are C language programs, and the simulation environment is VC++ 6.0.

A. COMPARISON METHODS SELECTED

Given the proposed method is an improvement of [19] (mentioned in Section IV), the method of single-target path (referred to as SIPA) is first selected as a comparison to confirm that the proposed method can improve the efficiency of test data generation for covering multiple paths and faults detection. In the method, genetic algorithm is used to generate test data covering one target path at a time. Therefore, it is necessary to run the algorithm N times to generate the test data covering N target paths. Since the proposed method (referred to as MTPA) is a test data generation method for multi-target paths. The other two comparison methods for multi-target paths are selected: the method described in [32] (referred to as AMED), using branch distance and layer proximity as the individuals' fitness. It is a typical method and is often used as a comparison method in experiments [28], [37]. Therefore, the proposed method compares with it in the same way as other methods. The method in [37] (referred to as HUFF) designs the fitness function based on the method of AMED. It calculates the fitness of test data by comparing the matching degree of the path code traversed by the test data with each target path code. Besides, test data are all generated by setting individual' fitness of GA when using these four methods.

The termination conditions of these four algorithms are to generate test data covering all target paths or reach the maximum generation. To compare the four methods under the same experimental conditions, the common experimental parameters and population size are the same, and the initial population data are randomly generated in the experiment.

B. PROBLEMS TO BE VERIFIED

By experiments, the running time, evaluation times, success rate and coverage balance finding test data covering multiple target paths of the four methods are compared. Besides, the branch coverage and fault detection are also compared. Problems to be verified and the evaluation criteria are described as follows:

RQ1: Can the proposed method effectively generate test data covering multiple target paths?

$$Suc = \left(\sum_{i=1}^{Num} \frac{Total'_i}{Total} \Big/ Num\right) * 100\% \tag{11}$$

It is verified by the success rate in (11), where *Num* indicates the number of experiments, *Total* represents the number of target paths of the program under test, while $Total'_i$ indicates the number of target paths covered by the generated data in the i th $(1 \leq i \leq Num)$ experiment. It is not difficult to conclude that the higher the success rate Suc, the higher the efficiency of the proposed method to generate test data.

RQ2: What is the time validity of the proposed method for generating test data?

$$TT = \sum_{i=1}^{Num} T_i \Big/ Num \tag{12}$$

It is measured by the average running time TT, as shown in (12), where $T_i \ (1 \leq i \leq Num)$ represents the running time of the algorithm in i th experiment. It can be seen that the smaller the average running time, the better the time effectiveness of the algorithm.

RQ3: How efficient is the GA when generating test data by the proposed method?

$$EN = \sum_{i=1}^{Num} m^* G_i \Big/ Num \tag{13}$$

It is measured by the average number of individuals evaluation EN (in (13)), where m and G_i respectively represent population size and the running generation of GA in the i th experiment. It can be seen from (13) that when the population size remains the same, the more generations GA runs, the more evaluation times, which will cause the GA to run inefficiently. Therefore, the smaller the value of EN, the better.

RQ4: Can test data generated by the proposed method evenly cover the program under test?

After running the program with data corresponding to all individuals, the balance of data covering program branches can be obtained [19]. We record the balance obtained in the last generation of GA, as is shown in (14).

$$PB(t) = \sum_{k=1}^{|br|} BB_k(t) \tag{14}$$

where $|br|$ represents the total number of branch nodes on the multi-target paths, and $BB_k(t)$ indicates the branch balance of the k th branch node. We can see that the smaller the $PB(t)$, the more balanced the generated data covering program branches.

RQ5: Can the proposed method effectively generate test data for code coverage and fault detection?

We use the branch coverage to measure the effectiveness of code coverage, and use the fault detection rate FD to verify the fault detection capability of different methods.

$$FD = \left(\sum_{i=1}^{Num} \frac{Fa'_i}{Fa} \Big/ Num \right) {}^* 100\% \tag{15}$$

where Fa represents the number of faults of the program under test, and Fa'_i indicates the number of faults detected in the i th $(1 \leq i \leq Num)$ experiment.

C. EXPERIMENTAL PROGRAMS AND PARAMETER SETTINGS

To answer the five questions raised and verify the effectiveness of the proposed method, we select a set of programs that are often used in the field of software testing. We compare experimental results to verify the performance of different methods. The benchmark programs include Triangle, Bubble and Maxmin. In terms of program structure, the three programs include sequence statements, selection statements, and loop statements. In particular, the triangle classification program and bubble sorting are more typical benchmark programs for experiments in software testing and are often used in various verification experiments. Besides, three widely used industrial programs Replace, Space and Grep are selected for experiments. Sub-functions of each industrial program are selected for experiments.

Experimental settings of these programs are shown in Table 4 [19], [37]. Target paths for each program are randomly selected. The lines of code (#LOC), the selected functions, the population size and the maximum generations (referred to as Gen) of GA are all given in the table. Besides, the number of target paths and faults are also given.

TABLE 4. Description of the programs.

ID	Programs	#LOC	Functions	Size	Gen	Paths	Faults
T1	Triangle	36	Triangle	200	200	18	52
T2	Bubble	18	Bubble	15	200	6	34
T3	Maxmin	25	Maxmin	20	800	8	26
T4	Space	6199	fixgramp	100	2000	34	144
T5	Space	6199	fixsgrid	100	1500	51	197
T6	Replace	550	putsub, subline, change	200	30000	103	254
T7	Grep	13226	common_op_match_null_string_p	200	35000	86	398

As programs with real faults are difficult to find, one common solution is to seed faults by mutation testing. We use the method in [40] to transform the mutant killing problem into a branch coverage problem based on weak mutation testing. Considering the characteristics of the selected programs, we apply the types of Method-Level mutation operators in MuClipse to construct mutation branches [40].

In order to better illustrate the effectiveness of the proposed method in the experiments, the common parameter settings of the proposed method are the same as those of the other three methods. Since the proposed method and the three comparison methods both use genetic algorithm to generate test data. To ensure that the initial experimental conditions of the four methods are the same, Table 5 lists the parameter settings of genetic algorithm in the experiments [37].

Under the same experimental conditions, each algorithm in experiments was run many times to get more accurate experimental results, and the average values are listed, as are shown in Table 6 to Table 9 and Fig. 2 to Fig. 7.

TABLE 5. Parameter settings of genetic algorithm.

Parameters	Selection	Crossover	Crossover probability	Mutation	Mutation probability	Gene coding
Values	Roulette	Single point	0.9	Single point	0.3	Binary

TABLE 6. Experimental results of programs.

	MTPA		SIPA		HUFF		AMED	
ID	time(ms)	rate(%)	time(ms)	rate(%)	time(ms)	rate(%)	time(ms)	rate(%)
T1	16.54	100	18.08	100	19.02	100	24.66	100
T2	0.31	100	0.31	99.83	1.24	99.83	0.63	99.5
T3	6.11	100	6.13	99.88	6.41	99.75	14.84	95.25
T4	32.68	100	40.16	100	39.48	100	42.17	100
T5	39.46	100	50.38	100	51.07	100	62.14	97.64
T6	42.78	100	48.17	100	53.29	96.15	59.83	95.23
T7	159.42	100	142.78	98.17	201.53	95.05	224.29	92.17

TABLE 7. Experimental results of programs.

ID	MTPA		SIPA		HUFF		AMED	
	CB(%)	ET	CB(%)	ET	CB(%)	ET	CB(%)	ET
T1	56.48	4392	67.01	8392	78.19	6384	71.46	14480
T2	41.44	44.1	38.98	64.35	44.11	91.05	44.31	225.6
T3	47.7	2830.6	53.88	3200.8	65.48	3276.2	57.75	10640
T4	58.67	3487	53.48	5078	62.17	5217	61.26	4932
T5	58.37	2864	62.34	4452	59.07	6721	63.15	18421
T6	59.76	3372	64.28	4876	73.18	5819	69.24	13246
T7	56.03	6482	69.82	8457	70.05	6791	75.18	25884

D. EXPERIMENTAL RESULTS

Answers to RQ1: Can the proposed method effectively generate test data covering multiple target paths?

It can be seen from Table 6 that for the proposed method, the success rates of T1-T7 are all 100%, that is to say, the seven programs have successfully generated data covering the target paths by employing it. Using the single-target path method SIPA, the success rate of T1 and T4-T6 is 100%, but the success rate of T2, T3 and T7 is 99.83%, 99.88% and 98.17% respectively, which is less than the proposed method. Similarly, it can be seen from the table that the success rate of HUFF for T1 and T4-T5, the success rate of AMED for T1 and T4 is all 100%, and the success rate of these two methods for other programs is all less than 100%. This is because that fitness function calculated by our method is more reasonable and it can generate test data covering multiple target paths quickly, which also shows our method in this paper is more effective.

By comparison, it can be seen that the proposed method has a higher success rate than the other three ones on the whole, that is, under the same initial parameters, test data covering multiple target paths can be generated more effectively by employing the proposed method. This is because the balance of test data covering different target paths and the weight of different target paths are considered, and individuals which promote the generation of test data covering target paths will be retained in the evolutionary process. So the RQ1 is verified. It can also be seen from Table 6 that the success rate of the SIPA method is better than the other two methods. This also illustrates the feasibility of applying the idea of balance in the generation of test data.

Answers to RQ2: What is the time validity of the proposed method for generating test data?

For the average running time of different programs, except for T7 (Grep), the time of our method is 159.42ms, which is slightly more than 142.78ms of SIPA. For other conditions, the time used by our method is less than the other three methods.

The reason why the proposed method has less running time than the other three methods is: compared with SIPA, the number of branch nodes on different target paths is taken into account in the calculation for individuals' fitness, which ensures that the more branch nodes on the target path, the greater the fitness is. In addition, when the GA is run once, test data covering multiple target paths can be generated by employing MTPA. While test data only covering one target path can be generated by SIPA, so it is necessary to run GA multiple times to generate data covering multiple target paths for the method. As a result, less time will be taken for the proposed method. When calculating the individual fitness, we calculate the coverage balance and the weights of different targets, which is easy to get after running the program with test data. When calculating the fitness for AMED and HUFF, AMED needs to calculate the branch distance and layer proximity according to the branch predicate of individuals traversing each branch node, while the HUFF method needs to compare the execution path of an individual with each target path and calculate the fitness according to the path matching degree and the number of consecutive same nodes on the two paths. When the number of individuals is large, the calculation amount of these two methods is large. Therefore, the running time of these two methods is also longer than the

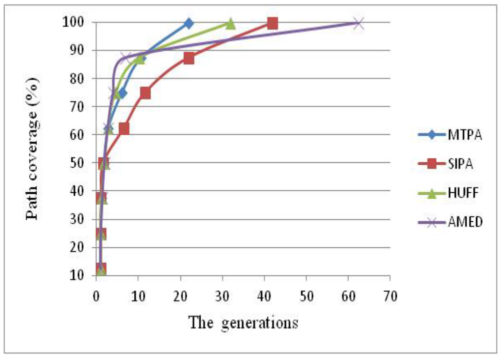


FIGURE 2. Generations to generate target paths (T1).

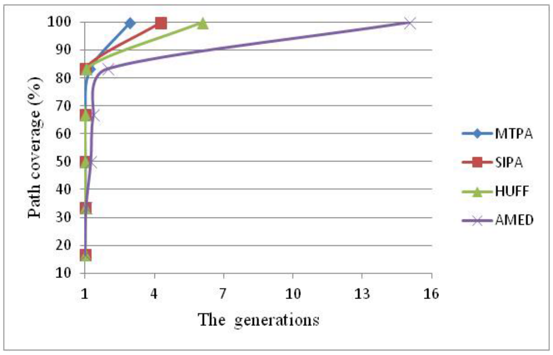


FIGURE 3. Generations to generate target paths (T2).

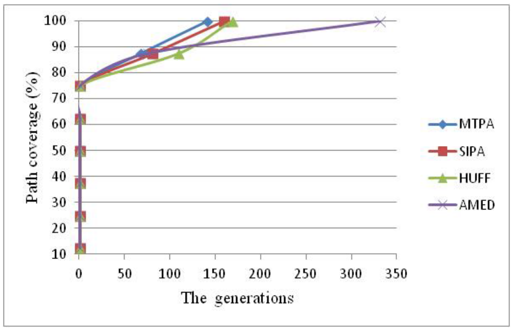


FIGURE 4. Generations to generate target paths (T3).

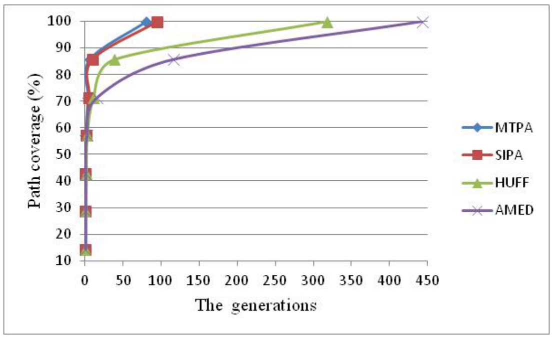


FIGURE 5. Generations to generate target paths (T4).

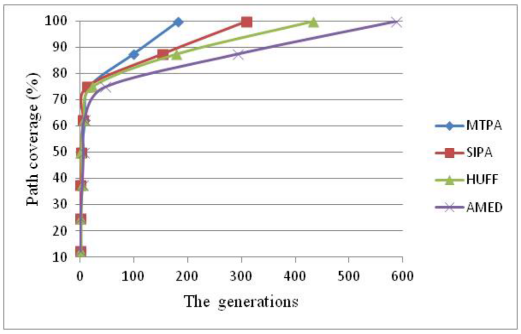


FIGURE 6. Generations to generate target paths (T5).

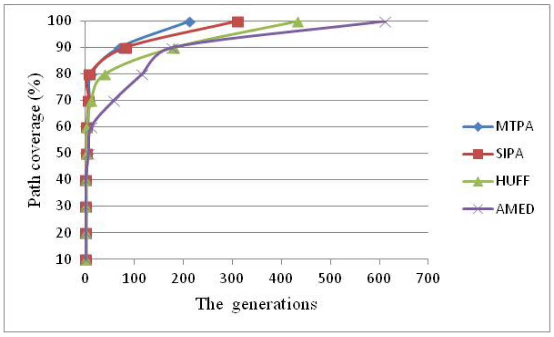


FIGURE 7. Generations to generate target paths (T6).

proposed method. This also answers RQ2: compared with the other three methods, the time effectiveness of the proposed method to generate test data is better.

Although for T4-T6, except for the AMED method, the running time of the other three methods has little difference. Our method has a slight advantage. Combining the success rate, it is not difficult to conclude that compared with the other three methods, our method can improve the success rate of test data generation while ensuring time validity.

Answers to RQ3: How efficient is the GA when generating test data by the proposed method?

Denote the coverage balance and evaluation times as *CB* and *ET* respectively in Table 7. In the table, the average evaluation times and coverage balance of the four methods are compared.

1) COMPARISON OF EVALUATION TIMES

It can be seen from Table 7 that from T1 to T7, the average evaluation times of the proposed method are significantly less than those of other methods, which means that the GA of our method is more efficient. As under the same conditions, our fitness is more reasonable. Individuals that are conducive to generate test data covering the target paths are more likely to be retained. Therefore, fewer evolution generations are needed for GA to achieve one of the termination conditions. It can also be seen from Table 7 that AMED needs more evaluation times than the other three methods. This is because it requires more generations to generate test data covering the target paths.

2) COMPARISON OF RUNNING GENERATIONS OF GA

To further verify the cost of the four methods, the experiment recorded the running generations of GA required for different paths for the programs. Experimental results are shown in Fig. 2 to Fig. 4. The horizontal axis of each figure represents the evolutionary generations of GA for different paths, and the vertical axis represents the path coverage.

It can be seen from Fig. 2 that for the easy-to-find target paths, compared with other methods, the evolutionary generations of MTPA are significantly less than that of SIPA, but it is more than that of HUFF and AMED. This is because the fitness is computed according to the comparison between each path traversing by an individual and the target path for HUFF. The branch distance and layer proximity are calculated according to the predicate of each branch node that an individual traverses for AMED. For the path that is easy to generate test data, it often contains fewer branch nodes, and the calculation of the fitness of these two methods is small. Therefore, these two methods can generate test data required faster. For paths that are difficult to generate test data, more branch nodes are often included, and these two methods require a lot of calculation. The method in this paper calculates the individuals' fitness according to the changes in the coverage balance before and after the individual's corresponding test data traversing the program branches. That is, the goal of the proposed method is to make test data evenly cover the branches of the program. Therefore, it is possible to generate test data that cover complex target paths, and the more branch nodes the target path contains, the better the effectiveness of the method in generating test data. Therefore, the advantages of our method are more obvious for target paths that are difficult to generate test data.

TABLE 8. Generations to generate target paths (T7).

Generations / PC (%)	MTPA	SIPA	HUFF	AMED
37.5	1	1	1	1
50	2.24	1.93	2.01	2.55
62.5	5.18	3.34	2.89	4.35
75	16.18	12.15	13.79	18.23
87.5	18.14	17.56	16.18	25.34
100	298.67	384.52	443.18	517.84

The evolutionary generations required for each method to generate test data covering each target path are compared in Fig. 2 to Fig. 7. For T7, the evolutionary generations of various methods for target paths are quite different. Therefore, the experimental results are shown in Table 8, where PC means the path coverage. It is not difficult to see that for T5-T7, the advantages of our method are more obvious. As the target paths increases, test data of the target paths that are easy to cover are generated at the beginning with fewer generations. Therefore, the difference of generations among these methods is small, but for complex target paths, compared with the other three methods, evolutionary generations of the proposed method are less. This also answers RQ3, the proposed method is more efficient when generating test data covering the target paths.

Answers to RQ4: Can test data generated by the proposed method evenly cover the program under test?

Except for program T2 in Table 7, the coverage balance of SIPA is less than the proposed method. In other cases, the coverage balance of our method is more obvious. In other words, test data generated by the proposed method can cover the programs more evenly. The reason is that coverage balance of test data covering the program in the calculation of fitness function is considered, and the weights of different target paths are set according to the number of branch nodes contained in target paths. This also answers question 4, test data generated by MTPA can more evenly cover the programs under test.

From the coverage balance of industrial programs, except for T4, the coverage balance of our method is slightly higher than that of SIPA method. For other situations, the coverage balance of our method is less than other methods. It is not difficult to see from Table 7 that both the proposed method and SIPA are both better than the other two methods on the whole. Since these two methods both consider the problem of balanced coverage in the generation of test data. So the RQ4 is verified.

Answers to RQ5: Can the proposed method effectively generate test data for code coverage and fault detection?

Experimental results are shown in Table 9. In the table, denote BC and FD as branch coverage and fault detection rate.

Answers to RQ5: Can the proposed method effectively generate test data for code coverage and fault detection?

We can see from Table 9 that for the proposed method, the branch coverage for T1 to T5 are all 100%, while the values are different for other programs. We can also see that the methods of MTPA and SIPA can both achieve higher branch coverage for all programs than the other two methods, the reason is that they consider branch balance and test data cover the programs evenly in the process. In addition, as in the proposed method, we set that the more branch nodes on target path p_j, the greater the weight of the individual's fitness to the path. As a result, test data are generated to cover p_j with more branch nodes rapidly. Compared with the other three methods, MTPA not only guarantees balanced coverage of program branches, but also ensures more data traverse branches that are difficult to cover. Therefore, the fault detection rate of the proposed method is higher than that of other methods. So the RQ5 is verified.

Experimental results of benchmark and industrial programs show that comparing with the other three methods, the method proposed shows good performance in terms of evaluation times, running time, success rate and coverage balance. That is, it is verified that the proposed method can improve the efficiency of test data generation for multi-target paths.

E. STATISTICAL ANALYSIS OF EXPERIMENTAL RESULTS

To verify the reliability of experimental results, we use statistical analysis to test the significance of the results. Hypothesis testing is an important branch of statistical inference. In this

TABLE 9. Experimental results of branch coverage and fault detection rate.

ID	MIPA		SIPA		HUFF		AMED	
	BC(%)	FD(%)	BC (%)	FD(%)	BC (%)	FD(%)	BC (%)	FD(%)
T1	100	100	100	96.88	100	98.96	99.86	82.69
T2	100	100	100	100	100	100	100	100
T3	100	100	100	99.23	100	98.46	100	99.23
T4	100	100	95.12	92.34	93.12	66.71	88.63	71.97
T5	100	97.37	91.67	85.61	84.37	79.36	89.34	69.94
T6	93.23	91.17	84.76	82.44	78.56	73.15	75.63	72.95
T7	92.16	89.74	82.43	78.29	76.33	72.18	73.79	71.36

paper, the Z test method of hypothesis testing is used to predict the actual performance of the algorithm by experimental results [41], [42].

Assuming that the significance level α is set to 0.01, then $-Z_a = -2.325$. The test hypothesizes is that there is no significant difference in the number of average evaluation times compared with different methods. Let the random variable X denote the average number of evaluations. In practical problems, these random variables are formed by the comprehensive influence of a large number of mutually independent random factors. Therefore, X approximately follows the normal distribution $X_i \sim N(\mu_i, \delta_i^2)$, the mean μ of random variables corresponding to each method is compared. The smaller the value, the lower the expected value of evaluation times required for the method to generate test data, and the higher the efficiency of the method for generating test data. Assuming that the evaluation times for each run of the proposed method MIPA, SIPA, HUFF and AMED are X_1, X_2, X_3 and X_4 respectively.

Taking the triangle classification program as an example, the comparison process of the four methods is given. Considering that the sample variance is an unbiased estimate of the population variance, therefore, the value of the sample variance is used as an estimate of the population variance. According to experimental results of the triangle classification program, the following data are obtained: the sample size: $n_1 = n_2 = 50$, the sample mean: $\overline{X_1} = 4400$, $\overline{X_2} = 8392$, the sample standard deviation: $\delta_1 = 3049.7$, $\delta_2 = 5875$.

Step 1. Establish the original hypothesis $H_0 : \mu_1 \geq \mu_2$ and the opposite hypothesis $H_1 : \mu_1 < \mu_2$;

Step 2. Construct the statistic $U_1 = \frac{\overline{X_1}-\overline{X_2}}{\sqrt{\frac{\delta_1^2}{n_1}+\frac{\delta_2^2}{n_2}}}$.

Step 3. Give the exclusion domain $U_1 = \frac{\overline{X_1}-\overline{X_2}}{\sqrt{\frac{\delta_1^2}{n_1}+\frac{\delta_2^2}{n_2}}} \leq -Z_a$.

Step 4. Calculate the value of the statistic:

$$U_1 = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{\delta_1^2}{n_1} + \frac{\delta_2^2}{n_2}}} = \frac{4400 - 8392}{\sqrt{\frac{3049.7^2}{50} + \frac{5875^2}{50}}} \approx -4.26.$$

Step 5. Give the conclusion:

Because $U_1 = -4.26 \leq -Z_a = -2.325$, it falls within the rejection domain. Therefore, reject $H_0 : \mu_1 \geq \mu_2$ and accept $H_1 : \mu_1 < \mu_2$, which means that the expected value of the evaluation times of our method is significantly smaller than that of the single-path method SIPA. The results show that compared with SIPA, our method requires significantly less evaluation times to generate test data.

At the same significance level $\alpha = 0.01$, $\delta_1 = 3049.7$, $\delta_3 = 4338.3$, the sample size $n_1 = n_3 = 50$, and $\overline{X_3} = 6452$. Establish the original hypothesis and the opposite hypothesis respectively: $H_0 : \mu_1 \geq \mu_3$; $H_1 : \mu_1 < \mu_3$. Construct the statistic and calculate:

$$U_2 = \frac{\overline{X_1} - \overline{X_3}}{\sqrt{\frac{\delta_1^2}{n_1} + \frac{\delta_3^2}{n_3}}} = \frac{4400 - 6452}{\sqrt{\frac{3049.7^2}{50} + \frac{4338.3^2}{50}}} \approx -2.74 \leq Z_a$$
$$= -2.325$$

TABLE 10. Hypothesis test results.

ID	Methods	Statistic value	Test result
T2	SIPA	$U_1 \approx -1.48 > -2.325$	Accept $H_0 : \mu_1 \geq \mu_2$
	HUFF	$U_2 \approx -1.63 > -2.325$	Accept $H_0 : \mu_1 \geq \mu_3$
	AMED	$U_3 \approx -9.71 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$
T3	SIPA	$U_1 \approx -3.07 < -2.325$	Accept $H_1 : \mu_1 < \mu_2$
	HUFF	$U_2 \approx -3.39 < -2.325$	Accept $H_1 : \mu_1 < \mu_3$
	AMED	$U_3 \approx -9.91 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$
T4	SIPA	$U_1 \approx -6.24 < -2.325$	Accept $H_1 : \mu_1 < \mu_2$
	HUFF	$U_2 \approx -9.84 < -2.325$	Accept $H_1 : \mu_1 < \mu_3$
	AMED	$U_3 \approx -13.24 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$
T5	SIPA	$U_1 \approx -14.71 < -2.325$	Accept $H_1 : \mu_1 < \mu_2$
	HUFF	$U_2 \approx -12.39 < -2.325$	Accept $H_1 : \mu_1 < \mu_3$
	AMED	$U_3 \approx -21.45 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$
T6	SIPA	$U_1 \approx -14.32 < -2.325$	Accept $H_1 : \mu_1 < \mu_2$
	HUFF	$U_2 \approx -19.78 < -2.325$	Accept $H_1 : \mu_1 < \mu_3$
	AMED	$U_3 \approx -24.89 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$
T7	SIPA	$U_1 \approx -26.34 < -2.325$	Accept $H_1 : \mu_1 < \mu_2$
	HUFF	$U_2 \approx -38.71 < -2.325$	Accept $H_1 : \mu_1 < \mu_3$
	AMED	$U_3 \approx -42.24 < -2.325$	Accept $H_1 : \mu_1 < \mu_4$

Therefore, reject $H_0 : \mu_1 \geq \mu_3$ and accept $H_1 : \mu_1 < \mu_3$.

In the same way, when $\alpha = 0.01$, $\delta_1 = 3049.7$, $\delta_4 = 9238.7$, the sample size $n_1 = n_3 = 50$, and $\overline{X_4} = 13176$. Establish the original hypothesis and the opposite hypothesis respectively: $H_0 : \mu_1 \geq \mu_4$; $H_1 : \mu_1 < \mu_4$. Construct the statistic and calculate:

$$U_3 = \frac{\overline{X_1} - \overline{X_4}}{\sqrt{\frac{\delta_1^2}{n_1} + \frac{\delta_4^2}{n_4}}} = \frac{4400 - 13176}{\sqrt{\frac{3049.7^2}{50} + \frac{9238.7^2}{50}}} \approx -6.38 \leq Z_a$$
$$= -2.325$$

Therefore, reject $H_0 : \mu_1 \geq \mu_4$ and accept $H_1 : \mu_1 < \mu_4$. The results show that the evaluation times required to generate test data by our method is significantly less than that of HUFF or that of AMED.

Using the comparison method of the above experimental results to test the experimental results of other programs, the test results are shown in Table 10. For program T2, the scale of sorting data selected is small, so our method has little difference from the result of SIPA and HUFF. It can be seen from Table 10 that in addition to T2, for other programs, the results of MIPA are significantly different from those of other comparison methods. Experimental results show that the evaluation times required by the proposed method is significantly less than those of the other three methods.

VI. CONCLUSION

Although there are various studies related to generating test data for multiple path coverage, the proposed method is different from other ones. The purpose of this study is to generate test data that traverse target paths as rapidly as possible. We apply the coverage balance as well as the weight of different target paths, and the constraint is that the traversed path is one of the target paths.

We apply the proposed method in some benchmark and industrial programs, and compare it with the single path method (SIPA) [19], the Huffman method (HUFF) [37], the Ahmed's method [32] in several aspects.

The experimental results show that our method can generate test data that traverse target paths rapidly. However, there are some limitations in our work. The programs under test considered here are limited. Therefore, our future work will focus on investigating and verifying the performance of our method in more complex industrial programs.

REFERENCES

[1] S. Jiang, J. Shi, Y. Zhang, and H. Han, "Automatic test data generation based on reduced adaptive particle swarm optimization algorithm," *Neurocomputing*, vol. 158, pp. 109–116, Jun. 2015.

[2] W. Zheng, R. M. Hierons, M. Li, X. Liu, and V. Vinciotti, "Multi-objective optimization for regression testing," *Inf. Sci.*, vols. 334–335, pp. 1–16, Mar. 2016.

[3] A. Shahbazi and J. Miller, "Black-box string test case generation through a multi-objective optimization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 4, pp. 361–378, Apr. 2016.

[4] I. Hermadi, C. Lokan, and R. Sarker, "Dynamic stopping criteria for search-based test data generation for path testing," *Inf. Softw. Technol.*, vol. 56, no. 4, pp. 395–407, Apr. 2014.

[5] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm," *Inf. Process. Lett.*, vol. 114, no. 6, pp. 304–316, Jun. 2014.

[6] S. M. Mohi-Aldeen, R. Mohamad, and S. Deris, "Application of negative selection algorithm (NSA) for test data generation of path testing," *Appl. Soft Comput.*, vol. 49, pp. 1118–1128, Dec. 2016.

[7] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "PSO based pseudo dynamic method for automated test case generation using interpreter," in *Proc. 2nd Int. Conf. Adv. Swarm Intell.*, 2011, pp. 147–156.

[8] P. Chawla, I. Chana, and A. Rana, "A novel strategy for automatic test data generation using soft computing technique," *Frontiers Comput. Sci.*, vol. 9, no. 3, pp. 346–363, Jun. 2015.

[9] R. L. Becerra, R. Sagarna, and X. Yao, "An evaluation of differential evolution in software test data generation," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 2850–2857.

[10] W. Jianfeng, W. Changan, and J. Shouda, "Test data generation algorithm of combinatorial testing based on differential evolution," in *Proc. 3rd Int. Conf. Instrum., Meas., Comput., Commun. Control*, Sep. 2013, pp. 544–548.

[11] X. Liang, S. Guo, M. Huang, and X. Jiao, "Combinatorial test case suite generation based on differential evolution algorithm," *J. Softw.*, vol. 9, no. 6, pp. 1479–1484, Jun. 2014.

[12] N. J. Kulkarni, K. V. Naveen, P. Singh, and P. R. Srivastava, "Test case optimization using artificial bee colony algorithm," in *Advances in Computing and Communications*. Kochi, India: Springer, 2011, pp. 570–579.

[13] D. Suri and P. Kaur, "Path based test suite augmentation using artificial bee colony algorithm," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 2, no. 9, pp. 156–164, 2014.

[14] P. R. Srivatsava, B. Mallikarjun, and X.-S. Yang, "Optimal test sequence generation using firefly algorithm," *Swarm Evol. Comput.*, vol. 8, pp. 44–53, Feb. 2013.

[15] M. Xiao, M. El-Attar, M. Reformat, and J. Miller, "Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques," *Empirical Softw. Eng.*, vol. 12, no. 2, pp. 183–239, Mar. 2007.

[16] A. Arcuri and X. Yao, "Search based software testing of object-oriented containers," *Inf. Sci.*, vol. 178, no. 15, pp. 3075–3095, Aug. 2008.

[17] O. Bühler and J. Wegener, "Evolutionary functional testing," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3144–3160, Oct. 2008.

[18] A. Arcuri, "Test suite generation with the many independent objective (MIO) algorithm," *Inf. Softw. Technol.*, vol. 104, pp. 195–206, Dec. 2018.

[19] S. Fan, Y. Zhang, and B. Ma, "Evolutionary generation of path coverage test data based on equilibrium optimization theory," *Acta Electronica Sinic*, vol. 48, no. 7, pp. 1303–1310, 2020.

[20] S. Jinhui, J. Ying, and S. Ping, "Research progress in software testing," *Acta Scientiarum Naturalium Universitatis Pekinensis*, vol. 41, no. 1, pp. 134–145, 2005.

[21] M. A. Saadatjoo and S. M. Babamir, "Test-data generation directed by program path coverage through imperialist competitive algorithm," *Sci. Comput. Program.*, vol. 184, pp. 1–19, Dec. 2019.

[22] L. Weizhi, "Test data generation based on automatic division of path," *Acta Electronica Sinica*, vol. 44, no. 9, pp. 2254–2261, 2016.

[23] E. Nikravan and S. Parsa, "A reasoning-based approach to dynamic domain reduction in test data generation," *Int. J. Softw. Tools Technol. Transf.*, vol. 21, no. 3, pp. 351–364, Jun. 2019.

[24] D. Xiang-Ying, G. Dun-Wei, and Y. Xiang-Juan, "Feasible path generation of weak mutation testing based on statistical analysis," *Chin. J. Comput.*, vol. 39, no. 11, pp. 197–213, 2016.

[25] M. Y. Hong and L. R. Qin, "Application of dynamic program slicing technique in test data generation," *Procedia Comput. Sci.*, vol. 111, pp. 355–360, Jan. 2017.

[26] D. Rui, D. Hongbin, and Z. Yan, "Fast automatic generation method for software testing data based on key-point path," *J. Softw.*,vol. 27, no. 4, pp. 814–827, 2016.

[27] A. M. Bidgoli, H. Haghighi, T. Z. Nasab, and H. Sabouri, "Using swarm intelligence to generate test data for covering prime paths," in *Proc. Int. Conf. Fundam. Softw. Eng.* Cham, Switzerland: Springer, 2017, pp. 132–147.

[28] Y. Zhang and D. Gong, "Generating test data for both paths coverage and faults detection using genetic algorithms: Multi-path case," *Frontiers Comput. Sci.*, vol. 8, no. 5, pp. 726–740, Oct. 2014.

[29] C. Xun and F. Huixing, "Research on automatic generation of test data based on improved GA," *Inf. Recording Mater.*, vol. 21, no. 8, pp. 164–165, 2020.

[30] X. Feng, "Research on fuzzy test data generation method based on improved genetic algorithm," *Inf. Comput. Theor. Ed.*, vol. 12, no. 5, pp. 52–54, 2020.
[31] G. Xuedi, Z. Lijuan, Z. Shudong, and L. Haoming, "Research on test data automatic generation based on improved genetic algorithm," *Comput. Sci.*, vol. 44, no. 3, pp. 209–213, 2017.
[32] M. A. Ahmed and I. Hermadi, "GA-based multiple paths test data generator," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3107–3124, 2008.
[33] Y. Chen and Y. Zhong, "Automatic path-oriented test data generation using a multi-population genetic algorithm," in *Proc. 4th Int. Conf. Natural Comput.*, 2008, pp. 566–570.
[34] N. Zhang, B. Wu, and X. Bao, "Automatic generation of test cases based on multi-population genetic algorithm," *Int. J. Multimedia Ubiquitous Eng.*, vol. 10, no. 6, pp. 113–122, Jun. 2015.
[35] T. Tian and D. Gong, "Evolutionary generation approach of test data for multiple paths coverage of message-passing parallel programs," *Chin. J. Electron.*, vol. 23, no. 2, pp. 291–296, 2014.
[36] Y. Zhang, D. Gong, X. Yao, and Q. Lu, "Generating test data covering multiple paths using genetic algorithm incorporating with reducing input domain," in *Proc. Chin. Intell. Syst. Conf.*, 2018, pp. 739–747.
[37] D. W. Gong and Y. Zhang, "Novel evolutionary generation approach to test data for multiple paths coverage," *Acta Electronica Sinica*, vol. 38, no. 6, pp. 1299–1304, 2010.
[38] H. Huang, F. Liu, X. Zhuo, and Z. Hao, "Differential evolution based on self-adaptive fitness function for automated test case generation," *IEEE Comput. Intell. Mag.*, vol. 12, no. 2, pp. 46–55, May 2017.
[39] H. Xia, X. Song, and L. Wang, "Research of test case auto-generating based on Z path coverage," *Modem Electron. Techn.*, vol. 6, pp. 92–94, Mar. 2006.
[40] G. J. Zhang, D. W. Gong, and X. J. Yao, "Test case generation based on mutation analysis and set evolution," *Chin. J. Comput.*, vol. 38, no. 11, pp. 2318–2331, 2015.
[41] C. Y. Xia, Y. Zhang, L. Wan, Y. Song, N. Xiao, and B. Guo, "Test data generation of path coverage based on negative selection genetic algorithm," *Acta Electronica Sinica*, vol. 47, no. 12, pp. 2630–2638, 2019.
[42] Y. Zhang and D.-W. Gong, "Evolutionary generation of test data for paths coverage based on scarce data capturing," *Chin. J. Comput.*, vol. 36, no. 12, pp. 2429–2440, Mar. 2014.

SHUPING FAN received the B.S. degree in mathematics education from Mudanjiang Normal University, China. He is an Associate Professor with the School of Computer and Information Technology, Mudanjiang Normal University. He is a Teacher of computer science. His research interests include test data generation for complex software and evolutionary computation.

NIANMIN YAO received the B.S., M.S., and Ph.D. degrees from Jilin University, China, in 1997, 2000, and 2003, respectively. From 2003 to 2005, he did his postdoctoral research with the Department of Computer Science, Tsinghua University. From 2010 to 2011, he visited the University of Connecticut as a Visiting Scholar. He is a Professor with the School of Computer Science and Technology, Dalian University of Technology. His research interests include search-based software engineering and network security.

LI WAN received the B.S. degree in computer science and technology from Mudanjiang Normal University and the M.S. degree from the Department of Intelligence and Computing, Tianjin University, China, in 2020. His interests include software testing and voice signal processing.

BAOYING MA received the B.S. degree in computer application technology from Mudanjiang Normal University, China, in 2007, and the M.S. degree in computer application technology from Harbin Engineering University, China, in 2010. She is a Lecturer with Mudanjiang Medical University. Her interest includes test data generation for complex software.

YAN ZHANG received the B.S. degree in mathematical coefficients education (computer software) from the China University of Mining and Technology, China, in 1994, the M.S. degree in teaching theory (computer) from Liaoning Normal University, China, in 2000, and the Ph.D. degree in control theory and control engineering from the University of Mining and Technology. She is a Professor and the Director of the School of Computer and Information Technology, Mudanjiang Normal University, China. Her research interests include search-based software engineering and evolutionary computation.