ET-based Test Data Generation for Multiple-path Testing

Qingjie Wei*
College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China

Yijin Li & Ying Qian
College of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China

ABSTRACT: Automatic test data generation for path testing is one of the difficult problems of software testing. And the existing methods have low efficiency and stability. We have conducted studies on the research field about ET-based test data generation for multiple-path testing, and we have improved the efficiency and the stability of the method by improved the fitness function of currently existing method. By the comparison experiments, it showed that the method in this paper have high efficiency and stability. Meanwhile, we gave the conclusion how the basic parameters impacted on the efficiency, and suggested the setting range of parameters.

Keywords: software testing; evolutionary testing; path coverage; test data; genetic algorithms

1 INTRODUCTION

Software testing is involved in the whole life cycle of software development, not only as a verification and validation process for the software quality improvement, but also as an essential method to assure the completion and to enhance the quality of a software project[1,2]. Meanwhile, it costs a lot to design the high quality test cases and check the results manually[3]. Finding the input data for these test cases manually is very time-consuming, especially when the logical structure of the program under test is very complicated[4]. And automated software testing can significantly reduce the cost of developing software. Therefore, Ould believes that the research of automatic test data generation will be the most important field for automatic testing[5].

Path coverage is the strongest coverage criterion in software testing, and automatic test data generation for path testing has become one of the difficult problems of automatic test data generation.

By the intensive study of automatic test data generation, this research field attracted more and more attention. Through the survey, we found the achievements in this field have been massive increased in the past few decades, especially in the last decade. The survey result is shown in Figure 1 and Figure 2.

Figure 1. The growth trend of the research of automatic test data generation.

Figure 2. The growth trend of the research of automatic test data generation for path testing.
As one of the research fields of automatic test data generation, evolutionary testing (ET) was first proposed by Xanthakis et al. in 1992 [6]. It is based on the basic framework of genetic algorithm (GA), which can find the test data in the input domain by heuristic searching. The method has high convergence and automation level, and can significantly reduce the test cost and improve the quality of testing [7-10].

Although the study of automatic test data generation for multiple-path testing received a lot of attention, most of them only consider the single path coverage (one search only aimed at single target path). At the same time, a few methods for multiple target paths have low efficiency and stability. Therefore, this paper adjusts and improves existing method to optimize the efficiency and stability. In addition, we first propose the impact factors on convergence rate of parameters and the recommended setting range of parameters.

2 BASIC CONCEPTS

2.1 ET

The basic idea of ET is: at first, in order to solve the problem of automatic test data generation, it is transformed into a function optimization, and the fitness function is defined; on the second, to encode the test data into the chromosomes and perform genetic manipulation; at last, to put decoded data to the program under test and calculate the performances of current test data. After certain generations of GA iteration, we can get the test data sets which are satisfied with the particular test adequacy criteria. The flow graph of ET is shown in Figure 3.

![Flow graph of ET](image)

It can be seen that the outer loop is the basic framework, which implements the population evolution in the input domain. The inner loop is the calculation for the individuals, which monitors the execution result and evaluates the fitness of the individuals. In other words, the inner loop is responsible for the evaluation of the individuals, and the outer loop is responsible for the population evolution.

2.2 Representation of the path

The representation of the path is the foundation of automatic test data generation for path testing, because the design of the fitness function depends on it. Gong et al. [11] have proposed that the path can be expressed as a kind of binary string by Huffman coding.

When Huffman coding was used to represent the path, programs under test should be instrumented via making bit ‘1’ to represent the true branch and making bit ‘0’ to represent the false branch. By this way, there is a one-to-one correspondence between the path of the program and the binary string. Then, we use a regular array A to store the execution path, and set its initial value to ‘0’. The specific rules are shown in Formula (1). |A| is the number of elements in the array, viz., the length of the path string. Among them, the different indexes of the array mean different branches. The assignment of the same index elements depend on the executions of the branches. If executed the true branch, the element will be assigned to 1, otherwise it will be assigned to 0.

\[
A[i] = \begin{cases} 
  1, & \text{executed true branch} \\
  0, & \text{executed false branch} 
\end{cases}, \quad i = 0, 1, \ldots, |A| - 1 \tag{1}
\]

Triangle Classifier [11-17] has been widely used in the field of software testing because of its clear and complicated logical structure. Meanwhile, only a tiny part of the test data in the whole input domain can cover the certain paths, so the Triangle Classifier also became the most common benchmark function of automatic test data generation for path testing. The sample of the Triangle Classifier is shown in Figure 4.

As shown in the figure, by instrumentation, the 4 paths of the Triangle Classifier can be expressed as: 000, 001, 01, 1. And after the execution of the program, the assignment result of the array A shows the path that the current test data can cover.

2.3 Fitness function design

As the core to transform automatic generate test data to the function optimization, the fitness function determines the quality of this method to some extents. In the methods based on Huffman coding, their design idea is to compare the execution path string with the target path string bit by bit. Through calculating the matching degree between them, let the path with higher matching degree have the higher fitness value. In the definition of fitness function of Gong [11], \( X \) is the test data, \( P(X) \) is the execution path string corresponds to \( X \), \( P_j \) is the j-th target path string, \( |P(X)| \) and \( |P_j| \) correspond to the length of \( P(X) \) and the length of \( P_j \). And \( d_B(P(X)) \) is the bit matching, it reflects
whether the k-th bit of \( P(X) \) and \( P_j \) are equal. If the k-th bit of \( P(X) \) and \( P_j \) are equal, then we let \( d_{jk}(P(X)) \) equal 1; otherwise, we let \( d_{jk}(P(X)) \) equal 0. The expression of \( d_{jk}(P(X)) \) is shown in Formula (2).

\[
\min(|P(X)|, |P_j|) \times \begin{cases} 
1, & \text{if the k-th bit of } P(X) \text{ and } P_j \text{ are equal} \\
0, & \text{otherwise}
\end{cases} \tag{2}
\]

The path strings of \( P(X) \) and \( P_j \) are compared bit-by-bit \( \min(|P(X)|, |P_j|) \) times. If the number of 1 in \((X)\) equals \(|P_j|\), it means that \( P(X) \) entirely matches with \( P_j \). In other word, the test data \( X \) covers the target path \( P_j \). Otherwise it means that \( P(X) \) partially matches with \( P_j \). If there are more numbers of 1 in \((X)\), the matching of \( P(X) \) and \( P_j \) will be higher.

The different bits of the path string correspond to the different matching degrees, and the later bit has the higher matching degree. So the different bits of the path strings should be given different weights. So, we introduce a variable \( m_{jk} \), which indicates the number of same bits when comparing \((X)\) with \( P_j \) to the k-th bit. And, \( f_j(X) \) is the matching degree of \( P(X) \) and \( P_j \), its expression is shown in Formula (3).

\[
f_j(X) = \sum_{k=1}^{\min(|P(X)|, |P_j|)} m_{jk}d_{jk}(P(X)) \tag{3}
\]

Thus, if there are more numbers of the same bits of \( P(X) \) and \( P_j \), the matching degree and the value of \( f_j(X) \) will be higher.

In addition, if there are more consecutive identical bits starting from the first bit, the similarity degree between the execution path and the target path will be higher. Therefore we have to adjust the Formula (3). We introduce a variable \( c_j(P(X)) \), which represents the number of the consecutive identical bits between \( P(X) \) and \( P_j \) from the first bit. The adjusted fitness function \( f_j(X) \) is shown in Formula (4).

\[
f_j(X) = (c_j(P(X)) + 1) f_j(X) \tag{4}
\]

If there are more numbers of the same bits of \( P(X) \) and \( P_j \), and more consecutive identical bits starting from the first bit, the matching degree and the value of \( f_j(X) \) will be higher.

For an example, when the target path \( P_j = '100' \), the execution path \( P(X) = '101' \) and \( P(X) = '001' \), the calculations of their matching degrees are as follows:

![Figure 4. The sample about Triangle Classifier.](image-url)
From the result, the matching degree of $P(X_i)$ and $P_1$ is higher than the matching degree of $P(X_2)$ and $P_1$. In conclusion, the fitness function of automatic test data generation for multiple-path testing is shown in Formula (5).

$$\text{fit}(X_i) = \frac{1}{m} \sum_{j=1}^{m} f_j(X_i)$$  \hspace{1cm} (5)$$

In this function, the test data $X$ is an n-dimensional vector in the input domain $D(X \in D)$, the target paths are $P_1, P_2, \ldots, P_m$, and $m$ is the number of remaining the target paths.

### 3 NEW FITNESS FUNCTION

The definition of the fitness function of Gong \[11] is unreasonable when the execution path and the target path have different lengths. Especially when the lengths of the target path and the execution path are very different, this situation is more serious.

For example, when the target path $P_1 = \text{‘1001’}$ and the execution path $P(X_1) = \text{‘1000’}$, the fitness value $f_1(X) = 24$. And when the target path $P_1 = \text{‘1000000000’}$ and the execution path $P(X_1) = \text{‘1000111111’}$, the fitness value $f_1(X) = 50$. In general, the former fitness value should be much higher than the latter fitness value. But the former is lower than the latter, because the design of the fitness function is unreasonable.

Therefore, we introduce a variable $F_j$, which indicates the fitness value when the execution path entirely matches with the target path. By this way, the adjusted fitness function is shown in Formula 6.

$$\text{fit}(X_i) = \frac{1}{m} \sum_{j=1}^{m} \frac{f_j(X_i)}{F_j}$$ \hspace{1cm} (6)$$

According to this, when the target path $P_1 = \text{‘1001’}$ and the execution path $P(X_i) = \text{‘1000’}$, the fitness value $f_1(X) = 0.48$. And when the target path $P_1 = \text{‘1000000000’}$, and the execution path $P(X_i) = \text{‘1000111111’}$, the fitness value $f_1(X) = 0.0826$. By this way, we effectively solve the problem that the matching error when the lengths of the target path and the execution path are different.

### 4 EXPERIMENTS

#### 4.1 Parameters setup

In order to get the optimized scheme of the parameters, we have conducted the experiments on the basic parameters of ET-based test data generation for multiple-path testing. Then we gave the conclusion that how the basic parameters impacted on the method efficiency, and suggested the setting range of parameters.

As a frequently-used benchmark program, Insertion Sort\[11,16,17\] has abundant target paths(sum to 120) and complicated logical structure. Due to the random factors have little impact on its results, and the experimental results are typical, We select the Insertion Sort program as the benchmark program.

As a replacement strategy of the individuals, generation gap indicates the scale of the generation genes replacement in the evolution\[18\]. Different generation gaps have different effects on the convergence rate of the method.

We ran the program 200 times and took the average when the crossover probability was 0.7, the mutation probability was 0.7, the population size was 5, the number of parameters was 5, the range of the values was $[0, 25-1]$, and the maximum number of the iteration was 500. The experimental results are shown in Figure 5.

As shown in the figure, generation gap impacted on the convergence rate of ET-based test data generation for multiple-path testing, has some rules as follows: at first, when generation gap setting is low(0-0.4), ET cannot cover the whole target paths; on the second, the convergence rate rapidly accelerates with the gradually enlarging generation gap(0.4-0.7); in the third, as the generation gap reaches a certain zone(0.7-1), the convergence rate is speedy and stable.

![Figure 5. Influence of generation gap on the convergence rate.](image)
Therefore, in ET-based test data generation for multiple-path testing, the recommended setting range of generation gap should be [0.7,1].

In addition, as the important parameters of ET, crossover probability and mutation probability have significant impact on the convergence rate of the method. As one of the primary operators to produce new individuals has some impact on the convergence rate. Besides, mutation keeps the population's diversity during evolution, avoiding the evolution halted and the immature convergence.

We ran the program 200 times and took the average when generation gap was 0.9, the population size was 5, the number of the parameters was 5, the range of the values was [0, 25-1]5, and the maximum number of the iteration was 1000. The experimental results are shown in Figure 6.

The figure shows that the impact on the convergence rate from the crossover probability is relatively weak. Instead, the mutation probability has a relatively significant impact on the convergence rate. The impact of mutation probability on the convergence rate is shown in Table 1.

Table 1. The impact of mutation probability on the convergence rate.

<table>
<thead>
<tr>
<th>Mutation probability (Pm)</th>
<th>Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cannot cover the whole target paths</td>
</tr>
<tr>
<td>0&lt;Pm&lt;0.1</td>
<td>the rate of convergence rapidly accelerates</td>
</tr>
<tr>
<td>0.1≤Pm≤0.9</td>
<td>the rate of convergence is speedy and stable</td>
</tr>
<tr>
<td>0.9&lt;Pm&lt;1</td>
<td>the rate of convergence rapidly decelerate</td>
</tr>
<tr>
<td>1</td>
<td>cannot cover the whole target paths</td>
</tr>
</tbody>
</table>

From this, we can see that the suggested setting range of mutation probability of ET-based test data generation for multiple-path testing should be [0.1,0.9].

4.2 Experiments design

To validate that our method has faster convergence rate and the better stability, we did the contrast experiments on 4 different mainstream benchmark programs. These benchmark programs are Triangle Classifier[11-17], Minimaxi[11,17,20], Bubble Sort[11,17,21], Insertion Sort. The description of these benchmark programs are shown in Table 2, and the descriptions of their parameter settings are shown in Table 3.

In order to make the results more accurate and typical, we got each result following these steps:

Step 1: do 200 times of experiments, and take the average as $R$;
Step 2: do extra 50 times of experiments, and take the average as $R_{50}$;
Step 3: calculate the difference between the two averages: $\xi = R_{50} - R$;
Step 4: if $|\xi| < 10^{-5}$, get the average as the result, otherwise return to step 2.

4.3 Experimental results and analysis

We have conducted these experiments as the above. The results of these experiments are shown in Figure 7, and the data of the experimental results are shown in Table 4.

According to the experimental results, the method in this paper has little advantage in the convergence rate and the stability, when the benchmark programs are Triangle Classifier, Minimaxi, and Bubble Sort. On the contrary, when the benchmark program is Insertion Sort, our method expresses the tremendous advantage in the convergence rate and the stability.

As the typical of the complicated logical structure programs, Insertion Sort has abundant target paths with different lengths. And our target is to improve the efficiency when the programs under test have complicated logical structure. Therefore, the experiments of Insertion Sort can fully demonstrate the advantages of our method.

In conclusion, the method in this paper has a slight advantage for the programs under test which have simple logical structure and same lengths of the target paths. And, when the programs under test have complicated logical structure and abundant target paths with different lengths, our method expresses the significant advantage that our method has the higher convergence rate and the higher stability.

![Figure 6. The impact of crossover probability and mutation probability on the convergence rate.](image-url)
In this paper, we have conducted studies on the research field about ET-based test data generation for multiple-path testing, and we have conducted experiments on our method and existing method. Then we gave the conclusion how the basic parameters impacted on the algorithm efficiency, and suggested the setting range of parameters. In addition, we have improved the efficiency and the stability of the method by improved the fitness function of currently existing method.

### REFERENCES


