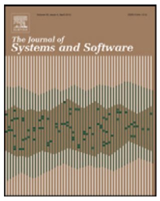
The Journal of Systems and Software 85 (2012) 2531–2540

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Grouping target paths for evolutionary generation of test data in parallel

Dunwei Gong[a], Tian Tian[a,*], Xiangjuan Yao[b]

[a] School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu 221116, PR China
[b] School of Science, China University of Mining and Technology, Xuzhou, Jiangsu 221116, PR China

ARTICLE INFO

Article history:
Received 31 August 2011
Received in revised form 18 May 2012
Accepted 23 May 2012
Available online 31 May 2012

Keywords:
Software testing
Test data
Multiple paths coverage

ABSTRACT

Generating test data covering multiple paths using multi-population parallel genetic algorithms is a considerable important method. The premise on which the method above is efficient is appropriately grouping target paths. Effective methods of grouping target paths, however, have been absent up to date. The problem of grouping target paths for generation of test data covering multiple paths is investigated, and a novel method of grouping target paths is presented. In this method, target paths are divided into several groups according to calculation resources available and similarities among target paths, making a small difference in the number of target paths belonging to different groups, and a great similarity among target paths in the same group. After grouping these target paths, a mathematical model is built for parallel generation of test data covering multiple paths, and a multi-population genetic algorithm is adopted to solve the model above. The proposed method is applied to several benchmark or industrial programs, and compared with a previous method. The experimental results show that the proposed method can make full use of calculation resources on the premise of meeting the requirement of path coverage, improving the efficiency of generating test data.

1. Introduction

Software testing is an important mean of guaranteeing software quality. The process is referred to as white-box testing if the source code of software under test is required during testing (Beizer, 1990). To check whether software meets the given requirements or not, some test adequacy criteria are needed for white-box testing. Among various criteria, the most commonly used one is path coverage, which requires test data generated can traverse the target path(s) of the program under test (Shan et al., 2005).

There are various methods of generating test data meeting path coverage. These methods are divided into static and dynamic ones according to whether software under test is executed, and dynamic methods need to execute software under test when generating test data (Shan et al., 2004). Generating test data covering target paths using genetic algorithms is one of dynamic methods which has brought to wide-ranging attention and achieved fruitful achievements (Xu, 2007). Genetic algorithm is a random search method inspired from biology evolution and heredity mechanism (Holland, 1975). For generating test data using genetic algorithms, the problem of generating test data is converted into an optimization one which is solved by genetic algorithms (Xanthakis et al., 1992).

Real-world software often contains multiple paths. It will undoubtedly improve the efficiency of generating test data if test data covering multiple target paths are generated in one run of a genetic algorithm. To this end, the problem of generating test data covering multiple paths can be converted into a multi-objective optimization one which is solved by existing evolutionary multi-objective optimization methods. By this way, the desired test data can be generated (Ahmed and Hermadi, 2008; Cao et al., 2010; Gong and Zhang, 2010).

But when there are many target paths, an optimization problem with many objectives will be formed which is very difficult to solve. If existing evolutionary multi-objective optimization methods are used to solve the problem, it will be very difficult, or even impossible to get test data meeting the requirement of path coverage. Therefore, a complicated optimization problem can be converted into several simpler sub-optimization problems after target paths have been divided into several groups according to appropriate methods, among which each group contains relatively fewer target paths. Each sub-optimization problem is solved by evolving a sub-population to generate test data covering target paths belonging to the corresponding group. Consequently, test data covering all target paths are generated (Gong et al., 2011). Unfortunately, existing grouping methods often make a great difference in the number of target paths belonging to different groups, and the efficiency of generating test data using genetic algorithms needs to be further improved.

The problem of grouping target paths when generating test data covering multiple paths using genetic algorithms is investigated in this study. An appropriate method of grouping target paths is expected which allows a small difference in the number of target

* Corresponding author. Tel.: +86 516 83995312; fax: +86 516 83995312.
E-mail address: tian_tiantian@126.com (T. Tian).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.05.071

paths belonging to different groups and a great similarity among target paths in the same group. To this end, the number of groups is uniquely determined by calculation resources available, and the base path is selected according to the sum of path similarities. And target paths belonging to a group are selected according to similarities among base paths and others. After grouping target paths, a mathematical model is built for the problem of generating test data, and a strategy of generating test data using a multi-population parallel genetic algorithm is adopted.

The rest of the paper is organized as follows. The proposed method of grouping target paths is given in Section 2, which includes a strategy of grouping target paths, a mathematical model for the problem of generating test data after grouping target paths, a strategy of evolutionary generation of test data, and the steps of the proposed algorithm. In Section 3, the proposed method is applied to some benchmark and industrial programs, and the experimental results and analysis are also given. Section 4 reviews related work. Finally, Section 5 concludes this paper, and provides possible opportunities for future research.

2. Proposed method

For convenience of description, several related concepts are introduced before the method of grouping target paths. More detailed contents about these concepts can be found in Gong et al. (2011).

Node: Denote a program under test as P. A node is a statement block of P, and all statements contained in the node are either executed or not at all when P is run. A node may be the condition of a loop, the condition of a select statement, or one or more successive statements.

Path: It is a serial of nodes traversed by the control flow when P is run under a given input, and denoted as p. The number of nodes contained in a path is called the path length.

Similarity of paths: Consider two target paths p_i and p_j of P, and their nodes are $p_{i1}, p_{i2}, \ldots, p_{i|p_i|}$ and $p_{j1}, p_{j2}, \ldots, p_{j|p_j|}$, respectively, where $|p_i|$ and $|p_j|$ are the path length of p_i and p_j, respectively. Compare p_j and p_i from p_{j1} to check whether they have the same nodes, and denote the number of successively same nodes as $|p_i \cap p_j|$. The similarity between p_i and p_j is defined as the ratio of the number of successively same nodes to the larger path length, and denoted as $s(p_i, p_j)$, whose expression is as follows:

$$s(p_i, p_j) = \frac{|p_i \cap p_j|}{\max\{|p_i|, |p_j|\}} \tag{1}$$

Denote P's input as x, where x may be either a scalar or a vector. If x is a vector, the types of different components may be different. Denote the set of inputs as D and m target paths to be covered as $p_1, p_2, \ldots, p_m$. According to Ahmed and Hermadi (2008), the problem of generating test data covering these m target paths can be converted into a multi-objective optimization one, and its mathematical model is as follows:

$$\begin{aligned} &\min(f_1(x), f_2(x), \ldots, f_m(x)) \\ &\text{st.}\, x \in D \end{aligned} \tag{2}$$

where $f_i(x)$ represents the objective related to the ith target paths. $f_i(x)$ will reach the minimum if and only if the traversed path is just the ith target one when P is run under the generated test data.

2.1. Grouping target paths

There are two factors to be considered when grouping target paths: the number of target paths and the composition of each group. The idea of the proposed method of grouping target paths in this study can be described as follows. The number of target paths in each group is determined according to calculation resources available to narrow the difference of the number of target paths in different groups. Target paths belonging to a group are determined according to similarities of paths to improve the similarity between the base path and the others in the group.

Suppose that there are n_r calculation resources available, and they are homogeneous. For example, n_r computer nodes with the same configuration can be used to simultaneously generate test data covering target paths. To reduce the time consumption spent in generating test data, the load of these calculation resources should be balanced by and large. In other words, we expect to generate test data using different calculation resources, and the number of target paths to be covered has a small difference in different calculation resources.

Owing to only n_r calculation resources available, m target paths should be divided into n_r groups, denoted as $g_1, g_2, \ldots, g_{n_r}$, and test data covering target paths of each group are generated only by one calculation resource, so that the number of test data generated by each calculation resource is the smallest. The detailed strategy is as follows:

Denote the number of target paths contained in g_i as $|g_i|$. For the first group, g_1,

$$|g_1| = \left\lfloor \frac{m}{n_r} + 0.5 \right\rfloor \tag{3}$$

For the ith group, g_i, $i = 2, 3, \ldots, n_r$,

$$|g_i| = \left\lfloor \frac{m - \sum_{j=1}^{i-1} |g_j|}{n_r - i + 1} + 0.5 \right\rfloor \tag{4}$$

Now, we illustrate that the groups obtained by the method above have a small difference in the number of target paths.

The number of target paths belonging to the first group, g_1, can be expressed as:

$$|g_1| = \left\lfloor \frac{m}{n_r} + 0.5 \right\rfloor = \begin{cases} \left\lfloor \dfrac{m}{n_r} \right\rfloor \\ \left\lfloor \dfrac{m}{n_r} \right\rfloor + 1 \end{cases} \tag{5}$$

When $|g_1|$ is equal to $\lfloor \frac{m}{n_r} \rfloor$, we have

$$\frac{m}{n_r} - 0.5 \le \left\lfloor \frac{m}{n_r} \right\rfloor \le \frac{m}{n_r} \tag{6}$$

Combining formulas (4)–(6), we have:

$$\begin{aligned} \left\lfloor \frac{m}{n_r} \right\rfloor &= \left\lfloor \frac{m}{n_r} + 0.5 \right\rfloor = \left\lfloor \frac{m - (m/n_r)}{n_r - 1} + 0.5 \right\rfloor \le |g_2| \\ &= \left\lfloor \frac{m - \lfloor m/n_r \rfloor}{n_r - 1} + 0.5 \right\rfloor \le \left\lfloor \frac{m - (m/n_r) + 0.5}{n_r - 1} + 0.5 \right\rfloor \\ &= \left\lfloor \frac{m}{n_r} + 0.5 + \frac{0.5}{n_r - 1} \right\rfloor \le \left\lfloor \frac{m}{n_r} + 1 \right\rfloor \le \left\lfloor \frac{m}{n_r} \right\rfloor + 1 \end{aligned} \tag{7}$$

According to formulas (5) and (7), we have:

$$\left\lfloor \frac{m}{n_r} \right\rfloor \le |g_2| \le \left\lfloor \frac{m}{n_r} \right\rfloor + 1 \tag{8}$$

When $|g_1|$ is equal to $\lfloor \frac{m}{n_r} \rfloor + 1$, we have:

$$\frac{m}{n_r} - 1 \le \left\lfloor \frac{m}{n_r} \right\rfloor \le \left\lfloor \frac{m}{n_r} \right\rfloor - 0.5 \tag{9}$$

Combining formulas (4), (5) and (8), we have:

$$\left\lfloor \frac{m}{n_r} \right\rfloor \le \left\lfloor \frac{m}{n_r} + 0.5 - \frac{0.5}{n_r - 1} \right\rfloor = \left\lfloor \frac{m - ((m/n_r) - 0.5 + 1)}{n_r - 1} + 0.5 \right\rfloor$$
$$\le |g_2| = \left\lfloor \frac{m - (\lfloor m/n_r \rfloor + 1)}{n_r - 1} + 0.5 \right\rfloor \le \left\lfloor \frac{m - (m/n_r)}{n_r - 1} + 0.5 \right\rfloor$$
$$= \left\lfloor \frac{m}{n_r} + 0.5 \right\rfloor = \left\lfloor \frac{m}{n_r} \right\rfloor + 1 \tag{10}$$

In all circumstances, we have:

$$\left\lfloor \frac{m}{n_r} \right\rfloor \le |g_2| \le \left\lfloor \frac{m}{n_r} \right\rfloor + 1 \tag{11}$$

Using similar deduction, we obtain:

$$\left\lfloor \frac{m}{n_r} \right\rfloor \le |g_i| \le \left\lfloor \frac{m}{n_r} \right\rfloor + 1 \tag{12}$$

for $i = 3, \ldots, n_r$. Thus, the difference between any two groups is no more than one in the number of target paths, suggesting that the method of grouping target paths above makes a balance by and large in the load of calculation resources.

The problem of the number of target paths belonging to a group is solved by the method above. Now, the method of forming a group by selecting target paths is presented. As shown beforehand, we expect a great similarity among target paths belonging to the same group. To this end, the following similarity matrix of target paths is constructed by calculating similarities among different paths according to formula (1), and denoted as S:

$$S = \begin{bmatrix} s(p_1, p_1) & s(p_1, p_2) & \ldots & s(p_1, p_m) \\ s(p_2, p_1) & s(p_2, p_2) & \ldots & s(p_2, p_m) \\ \vdots & \vdots & \ldots & \vdots \\ s(p_m, p_1) & s(p_m, p_2) & \ldots & s(p_m, p_m) \end{bmatrix} \tag{13}$$

The sum of all elements in the same row of the matrix above is calculated to reflect the similarity between a path and the others. For the ith row, the sum of all elements is $\sum_{j=1}^{m} s(p_i, p_j)$. It is evident that the larger the value of similarity between p_i and the others, the larger the value of $\sum_{j=1}^{m} s(p_i, p_j)$ is.

To determine the target paths belonging to g_1, the base path is first selected from the set of target paths, $\{p_1, p_2, \ldots, p_m\}$, and donated as p_1^b, which satisfies the condition $p_1^b = \arg\max_{i=1,2,\ldots,m} \sum_{j=1}^{m} s(p_i, p_j)$; then, the similarities between p_1^b and all paths (including p_1^b) are sorted in a descending order, and the target paths corresponding to the first $|g_1|$ similarities are selected to form g_1.

Having determined the target paths contained in g_1, these paths are deleted from $\{p_1, p_2, \ldots, p_m\}$, and the set of remaining target paths is obtained. Based on the set above and formula (13), the similarity matrix with a reduced order can be achieved. It is easy to understand that it is a matrix of $m - |g_1|$ order. The target paths belonging to $g_2, g_3, \ldots, g_{n_r}$, respectively, can be obtained in order by using the same method as forming g_1.

Now, we analyze the rationality of selecting target paths to construct different groups by the method above. Without loss of generality, the process of constructing g_1 is investigated. First, the path with the maximal value of $\sum_{j=1}^{m} s(p_i, p_j)$ is selected from the set of target paths as the base path, denoted as p_1^b, which guarantees the greatest overall similarity between p_1^b and the other paths. Then the target paths with the first $|g_1|$ greatest similarities between them and p_1^b are chosen, which leads to a great similarity between p_1^b and the other paths contained in g_1. In other words, the paths most similar to p_1^b belong to g_1. Hence, there is a great similarity among target paths in g_1, indicating that the method of selecting target paths to construct different groups is rational.

2.2. Mathematical model of generating test data

After all target paths have been divided into n_r groups, a multi-objective optimization problem is formulated based on target paths of each group. The original m-objective optimization problem, represented with formula (2), can thus be converted into n_r sub-optimization problems, and the ith one has $|g_i|$ objectives which corresponds to generating test data covering the target paths of the ith group. For convenience of illustration, denote the objectives corresponding to the target paths of the ith group as $f_{i1}(x), f_{i2}(x), \ldots, f_{i|g_i|}(x)$. According to formula (2), the ith sub-optimization problem can be described as follows:

$$\begin{aligned} &\min(f_{i1}(x), f_{i2}(x), \ldots, f_{i|g_i|}(x)) \\ &\text{st.}\, x \in D \end{aligned} \tag{14}$$

Having grouped target paths using the method proposed in this study, the mathematical model of generating test data covering multiple paths can be formulated as follows:

$$\begin{cases} \min(f_{11}(x), f_{12}(x), \ldots, f_{1|g_1|}(x)) \\ \text{st.}\, x \in D \end{cases}$$
$$\begin{cases} \min(f_{21}(x), f_{22}(x), \ldots, f_{2|g_2|}(x)) \\ \text{st.}\, x \in D \end{cases} \tag{15}$$
$$\vdots$$
$$\begin{cases} \min(f_{n_r 1}(x), f_{n_r 2}(x), \ldots, f_{n_r |g_{n_r}|}(x)) \\ \text{st.}\, x \in D \end{cases}$$

We investigate the relations between formula (15) and the existing model, namely formula (4) presented in Gong et al. (2011).

First, both of them are obtained based on grouping target paths, and convert an optimization problem with many objectives into several ones with fewer objectives to simplify the difficulty of the problem. Thus the efficiency of generating test data is improved.

Second, there is a great difference between the two models which are mainly embodied in the following two aspects.

(1) There are different numbers of converted sub-optimization problems between them. In formula (4), the number of sub-optimization problems is not empirically determined in advance, but obtained according to the similarity threshold and the selected base paths. Therefore, the number of sub-optimization problems depends on selecting the threshold and the base paths. In Gong et al. (2011), however, the base paths are randomly selected, and selecting the similarity threshold has no rules to obey, suggesting that if an inappropriate threshold is selected, it will result in too many or too small number of sub-optimization problems, which makes it difficult to solve the original problem. In formula (15), the number of sub-optimization problems is determined according to calculation resources available. That is to say, if there are a lot of calculation resources, there will be many sub-optimization problems, and vice versa, which makes full use of each calculation resource. Therefore, the efficiency of solving the original problem is improved.

(2) The number of objectives is different in different sub-optimization problems. In formula (4), the number of objectives is determined by the similarity threshold and the selected base paths. Even if the same threshold is used, different base paths will also lead to different grouping. This means that the number of objectives of different sub-optimization problems may be greatly varied, and if homogeneous calculation resources are used, the time consumption spent by these resources will have a great difference to solve

Table 1
Information of programs under test.

Program	Structure	# of inputs	Range of inputs	# of target paths
insert sort	Two nested loops	6	$[0, 8191]^6$	39
flex	One loop and one nested select statement	26	$[0, 127]^{26}$	23
make	One loop and one nested select statement	11	$[0, 511]^{11}$	14
grep	One loop and one nested select statement	22	$[0, 127]^{22}$	39
sed	One select statement and two loops nested by each select branch	3	$[0, 127]^3$	60
space	Two select statements and three loops nested by the first select statement	22	$[0, 127]^{22}$	16
bash	Several nested select statements	4	$[0, 511]^4$	12

these problems. As mentioned above, in formula (15), there is a difference of no more than one among the number of objectives of different sub-optimization problems. So, when homogeneous calculation resources are used to solve these problems, there is a small difference among the time consumption spent by them, which is helpful to reduce the overall time consumption in generating test data.

It can be shown from the analysis above that grouping target paths using the method proposed in this study can make full use of calculation resource available, therefore, improving the efficiency of generating test data.

2.3. Evolutionary generation of test data

We use a multi-population parallel genetic algorithm to solve n_r sub-optimization problems above, and each subpopulation optimizes a sub-optimization problem evolutionarily. Consequently, the whole population is divided into n_r subpopulations. The strategies in Gong et al. (2011) are used by each subpopulation to generate the initial population, calculate an individual's fitness, perform such operations as selection, crossover and mutation. Besides, the corresponding sub-optimization problem is simplified after the test datum covering a target path has been found to further reduce the complexity of the sub-optimization problem. More details can be found in Gong et al. (2011) for interested readers, which will not be repeated here.

2.4. Steps of parallel genetic algorithm

Step 1: Divide m target paths into n_r groups according to the method in Section 3.1.
Step 2: Assign the values of the parameters used in this algorithm and initialize n_r sub-populations.
Step 3: Calculate individuals' fitness in each sub-population.
Step 4: Judge whether the termination criteria are met or not. If yes, go to Step 7.
Step 5: Judge whether a target path is traversed or not. If yes, save test data and the target path, delete the traversed path from its corresponding group, simplify its corresponding sub-optimization problem, and recalculate the individuals' fitness in its corresponding subpopulation.
Step 6: Perform genetic operations to generate offspring, go to Step 4.
Step 7: Form and output the set of test data, output covered target paths, and terminate the algorithm.

3. Experiments

To confirm the validity of the method proposed in this study, we apply the proposed method to several benchmark or industrial programs, and compare it with method in Gong et al. (2011). Both methods are implemented in a cluster environment. Hardware configuration of calculation resources is 2 × Intel Xeon E5606 CPU, 2 × 6 GB memory, 2 × 146 GB RAID hard disks and the thousand trillion ether network. Software configuration is Linux operation system, Chess cluster management software and C programming language.

3.1. Programs under test

We choose *insert sort* as a benchmark program. In addition, there are six industrial programs, *flex*, *make*, *grep*, *sed*, *space* and *bash*, and each of them contributes a function. These programs are widely used to verify the performance of different methods of generating test data, such as in Ahmed and Hermadi (2008), Cao et al. (2010), Gong et al. (2011), and Alba and Chicano (2008). The structure, number of inputs, range of inputs, and number of target paths of these programs are listed in Table 1.

3.2. Genetic strategies and parameter settings

When generating test data covering target paths, different numbers of calculation resources and population sizes are used for different programs under test, as listed in Table 2. Binary coding is used to encode an individual in each subpopulation, and its genetic operations are roulette-wheel selection, one-point crossover, and one-point mutation with their probabilities of 0.9 and 0.3, respectively. Besides, the largest number of generations is set to 10,000 for all programs.

It should be noted that appropriate genetic operations and parameter values will improve the efficiency of generating test data. Since the goal of the experiments in this section is to verify the influence of different methods of grouping target paths on the efficiency of generating test data, we use the same genetic operations and parameter values for different methods of grouping for a fair comparison despite that they may not be optimal.

3.3. Performance indicators

To compare the performance of different methods, the following indicators are used.

Longest time consumption (Ltc). It means the longest time spent among all calculation resources in generating test data meeting the requirement of path coverage. As different calculation resources optimize different groups, the required time is different for generating test data covering target paths of different groups. Ltc is viewed as the runtime of the whole parallel algorithm. The less the longest time consumption is, the higher an algorithm's efficiency is.

Table 2
Calculation resources and population size.

Program	# of calculation resources	Population size
insert sort	4	10 × 4
flex	4	30 × 4
make	2	30 × 2
grep	4	15 × 4
sed	6	15 × 6
space	3	60 × 3
bash	5	60 × 5

Similarly, we can obtain the shortest time for generating test data, and refer it to the shortest time consumption.

Time difference (Td). It is the difference between the longest and the shortest time consumption, which reflects the difference of time spent by calculation resources in generating test data covering target paths of different groups. The less the time difference is, the higher the resources utilization rate is, so calculation resources are used more balancedly.

Average time consumption (Atc). It means the average time spent by all calculation resources in generating test data covering target paths of the groups corresponding to them. The less the average time consumption is, the less the total time consumption of all resources is.

Average number of generations (Ang). For a calculation resource, the number of generations required by the corresponding subpopulation to generate test data covering the target paths of the corresponding group is called the number of generations of the group. And the average of the number of generations of all groups is called the average number of generations.

Average path coverage rate (Apcr). For a calculation resource, the ratio of the number of covered target paths to that of all target paths in the corresponding group is called the path coverage rate of the group. And the average of the path coverage rate of all groups is called the average path coverage rate.

It is easy to know that on the premise of the same Apcr, along with the decrease of Ltc, Td and Atc, the calculation resources of a method are well utilized and the total time consumption is decreased. Therefore, its performance is good.

To reduce the influence of stochastic factors on the performance of a method, for each program, each method is run for 50 times independently, and the same initial population is used for the two methods in each run. The experimental results of the performance indicators above of each method are recorded, and then their averages of 50 runs are calculated. To confirm whether the difference of these methods is significant in the performance indicators above or not, one-side *t*-test is used based on the averages of these indicators. Let the significant level be 0.05, then the value of is 1.66. If the statistical value of in an indicator is greater than 1.66, it means that the method with a smaller value is significant better in this indicator than the other for Td, Atc, Ang and Ag, whereas it means that the method with a greater value is significant better than the other for Apcr.

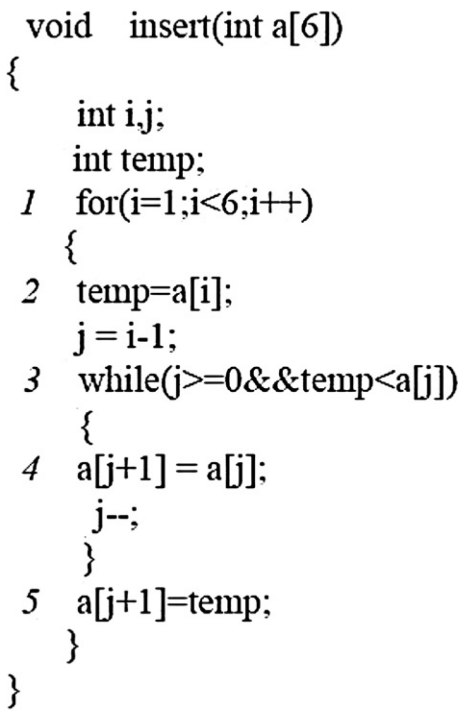
3.4. Experimental results in insert sort

The source code of *insert* sort is provided, as shown in Fig. 1, where 1, 2, 3, 4 and 5 are nodes obtained according to Section 3. The similarity threshold is first set to 0.4. When the base paths, shown as the bold parts in Table 3, are selected, according to Gong et al. (2011), target paths are divided into 14 groups, $g_1, g_2, \ldots$, and g_{14}, which contain 6, 2, 5, 5, 1, 3, 3, 3, 2, 4, 1, 1, 2 and 1 path(s), respectively, as shown in Table 3.

When the similarity threshold is set to 0.3, and the base paths, shown as the bold parts in Table 4, are selected, target paths are divided into eight groups, $g_1, g_2, \ldots$, and g_8, which contain 8, 5, 8, 6, 3, 6, 2 and 1 path(s), respectively, as shown in Table 4.

When we set the same value of the similarity threshold, and change the base paths into the counterparts, shown as the bold parts in Table 5, target paths are divided into nine groups, $g_1, g_2, \ldots$, and g_9, which contain 8, 5, 6, 6, 4, 6, 1, 2 and 1 path(s), respectively, as shown in Table 5.

As can be observed from Tables 3–5, under method in Gong et al. (2011), different groups of target paths are obtained by setting different similarity thresholds. If different bash paths are selected, there is still a great difference in grouping target paths, even if the same similarity threshold is set.



```
void    insert(int a[6])
{
        int i,j;
        int temp;
  1     for(i=1;i<6;i++)
        {
  2     temp=a[i];
        j = i-1;
  3     while(j>=0&&temp<a[j])
          {
  4     a[j+1] = a[j];
          j--;
          }
  5     a[j+1]=temp;
        }
}
```

Fig. 1. Source code of *insert sort*.

There are four calculation resources available for this program under test. To make full use of all calculation resources available, we divide target paths into four groups. To this end, the value of the similarity threshold is set to 0.2 after several tries. According to method in Gong et al. (2011), the groups of target paths are listed as Table 6. It can be observed that g_1, g_2, g_3 and g_4 contain 13, 16, 6 and 4 paths, respectively, making a great difference among different groups in the number of target paths.

Based on the method proposed in this study, the groups of target paths, g_1, g_2, g_3 and g_4 are listed as Table 7, which contain 10, 10, 10 and 9 paths, respectively.

Therefore, compared with method in Gong et al. (2011), the proposed method makes the groups of target paths more uniform.

The first and second rows of Table 8 list the averages and the variances of the two methods for *insert sort*. In addition, the statistical value of *t* for each performance indicator is listed in the third row.

It can be observed from Table 8 that:

There is no obvious difference between the two methods in Atc, Ang and Apcr. Compared with method in Gong et al. (2011), the proposed method reduces Ltc by 14.7% and Td by 25.5%, which means that although both methods require the same total time consumption, the proposed method can better utilize calculation resources, and greatly improve the execution efficiency.

The main cause of the results above is that grouping of target paths using method in Gong et al. (2011) increases time in generating test data covering the target paths of g_2, whereas decreases time in doing it for covering the target paths of g_4, which increases the longest time consumption and time difference of method in Gong et al. (2011). Based on the proposed method, the number of target paths of different groups is roughly similar. Therefore, there is little difference among time in generating test data covering the target paths of different groups, which makes a smaller time difference and the longest time consumption, indicating that the proposed method can improve the efficiency in utilizing calculation resources, hence reducing the time consumption spent in generating test data.

Table 3
Groups obtained by method in Gong et al. (2011) for similarity threshold being 0.4.

Group	Paths

g_1	**1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,5,1**;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_2	**1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1**; 1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1.

g_3	**1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_4	**1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1.

g_5	**1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**.
g_6	**1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1**; 1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1; 1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1
g_7	**1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1**; 1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1; 1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.
g_8	**1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1**; 1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1; 1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1.
g_9	**1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1**; 1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

g_{10}	**1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_{11}	**1,2,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1**.
g_{12}	**1,2,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1**.
g_{13}	**1,2,5,1,2,4,3,5,1,2,4,3,5,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**; 1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1.
g_{14}	**1,2,4,3,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**.

Table 4
Groups obtained by method in Gong et al. (2011) for similarity threshold being 0.3 (1).

Group	Paths

g_1	**1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,5,1**;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

g_2	**1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1.

g_3	**1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_4	**1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

g_5	**1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**; 1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1; 1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_6	**1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_7	**1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**; 1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.
g_8	**1,2,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1**.

3.5. *Experimental results in industrial programs*

According to calculation resources available, we set the similarity threshold to 0.4. Method in Gong et al. (2011) divides target paths of *flex* into four groups, g_1, g_2, g_3 and g_4 which contain 8, 6, 6 and 3 paths, respectively. The four groups obtained by the proposed method, however, contain 6, 6, 6 and 5 paths, respectively.

For *make*, we set the similarity threshold to 0.4. The two groups obtained by method in Gong et al. (2011), g_1 and g_2, contain 11 and 3 paths, respectively. Each group of the proposed method contains 7 paths.

Method in Gong et al. (2011) divides targets paths of *grep* into the following four groups with the similarity threshold 0.4, g_1, g_2, g_3 and g_4 which contain 30, 6, 2 and 1 path(s), respectively. Using the proposed method, each of the other three groups contains 10 paths except g_4 that contains 9 paths.

For *sed*, the groups obtained by method in Gong et al. (2011), $g_1, g_2, \ldots,$ and g_6 contain 23, 22, 2, 3, 7 and 3 paths, respectively, with the similarity threshold 0.6. But in the proposed method, each group contains 10 paths.

Method in Gong et al. (2011) divides *space*'s target paths into three groups with the similarity threshold 0.4, g_1, g_2 and g_3, which contain 6, 8 and 2 paths, respectively. The three groups of the proposed method, however, contain 5, 6 and 5 paths, respectively.

For *bash*, the last program, method in Gong et al. (2011) divides its target paths into five groups with the similarity threshold 0.6, g_1, g_2, g_3, g_4 and g_5, which contain 6, 1, 1, 3 and 1 path(s), respectively. The five groups in this method, however, contain 2, 3, 2, 3 and 2 paths, respectively.

The experimental and hypothesis testing results of the six programs are listed in Table 9.

As can be observed from Table 9,

For *flex*, there is little difference between groups of target paths of the two methods. Except Ang, the other four indicators of the proposed method are slightly superior to those of method in Gong et al. (2011). However, statistical analysis indicates no significant difference between them.

For *make*, except Td, the other four indicators of the proposed method are superior to those of method in Gong et al. (2011). Compared with method in Gong et al. (2011), the proposed method reduces Ltc by 10.9%, Atc by 31.0% and Ang by 37.0%, and improves Apcr by 12.9%. However, Td of method in Gong et al. (2011) is better than that of the proposed method. The reason is illustrated as follows: each group of the proposed method contains 7 paths wherein paths of g_1 can be covered by test data more easily than those of g_2. With the method in Gong et al. (2011), g_1 contains 11 paths of

Table 5
Groups obtained by method in Gong et al. (2011) for similarity threshold being 0.3 (2).

Group	Paths

g_1	**1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,5,1**;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

g_2	**1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1.

g_3	**1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_4	**1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_5	**1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_6	**1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

g_7	**1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**.
g_8	**1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1**; 1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1.
g_9	**1,2,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1**.

Table 6
Groups obtained by method in Gong et al. (2011) for similarity threshold being 0.2.

Group	Paths

g_1	**1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,5,1**;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1.

g_2	**1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1.

g_3	**1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_4	**1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,5,1**;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1.

which 7 belong to g_1 and 4 to g_2 of the proposed method, while the remaining 3 belong to g_2. Due to higher path complexity, more time is needed to generate test data covering g_2's paths, although it contains only 3. Meanwhile, there are 4 paths with higher complexity among g_1, therefore, it is more difficult to generate test data covering g_1. As a result, a relatively long time is taken to generate test data for g_1 and g_2, resulting in a less Td.

For *grep* and *sed*, there is no significant difference between the two methods in Atc and Apcr. Compared with method in Gong et al. (2011), the proposed method reduces Ltc by 21.2% and 23.6%, respectively. In addition, Td is reduced by 29.9% and 41.5%, respectively.

For *space*, Atc and Ang of the two methods indicate no significance. The proposed method reduces Ltc by 8.6% and Td by 52.5%, and improves Apcr by 8.9%.

For *bash*, Atc of the two methods has no significant difference. The proposed method reduces Ltc by 20.0% and Td by 20.1%, and improves Apcr by 9.2%.

For *grep*, *sed* and *bash*, Ang of the method in Gong et al. (2011) is better than that of the proposed method. This is due to that the groups of target paths obtained by method in Gong et al. (2011) are not even, containing from many paths to several or even one path. The number of generations required to generate test data covering a smaller number of target paths is less, hence reducing Ang.

4. Related work

Generating test data meeting a given adequacy criterion using genetic algorithms has been broadly studied, and achieved positive results in recent years (McMinn, 2004). The first work in which genetic algorithms are used to automatically generate test data was in 1992 by Xanthakis et al. (1992). The main idea can be described as follows. The problem of generating test data is first converted into an optimization one according to a given adequacy criterion, such as path coverage. Having defined a suitable fitness function and encoding test data, genetic operations are performed and software under test is executed for a number of generations. Consequently, the optimal solution for the optimization problem can be obtained. Finally, test data meeting the requirement can be yield by decoding the optimal solution.

Adopting branch and loop coverage as adequacy criteria, Sthamer (1996) investigated the influence of such genetic parameters as crossover and mutation probabilities as well as those in fitness on generating test data. Directed by the control dependence graph of the program under test, Pargas et al. (1999) adopted a genetic algorithm to search for test data to meet node and branch coverage. Girgis (2005) provided a technique for data flow coverage that uses the ratio of the number of covered def-use paths by a test datum to the total number of def-use paths as an

Table 7
Groups obtained by proposed method.

Group	Paths

g_1	**1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1**;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1.

g_2	**1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,5,1**;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,5,1,2,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1.

g_3	**1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1**;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,4,3,5,1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,5,1;
	1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1,2,5,1,2,4,3,4,3,5,1.

g_4	**1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1**;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,5,1,2,5,1,2,4,3,5,1,2,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,5,1,2,4,3,4,3,5,1,2,4,3,4,3,4,3,4,3,4,3,5,1;
	1,2,5,1,2,4,3,5,1,2,4,3,4,3,4,3,5,1,2,4,3,4,3,5,1,2,5,1.

individual's fitness. Wegener et al. (2002) proposed an evolutionary test environment, which automatically generates test data for most structural test methods and is applied to embedded systems. The fitness function includes the branch distance and the approach level. Bueno and Jino presented a tool that uses genetic algorithms directed by the control flow and data flow to generate test data. This method can also be used to detect potential infeasible paths (Bueno and Jino, 2000). Furthermore, Bueno and Jino (2002) presented that test data covering similar paths have a similarity, and existing test data are preferentially selected as the individuals of the initial population for generating test data covering similar paths. Bueno et al. (2011) also used the well-known search techniques, such as genetic algorithms, as optimization tools to generate the diversity oriented test sets. Xie et al. (2008) presented three methods of designing a fitness function based on the similarity for generating test data covering given paths. Baars et al. (2011) reduced the number of fitness evaluations by incorporating information obtained from symbolic execution into the fitness function for branch coverage. Harman et al. (2007) used a dependence analysis for predicates to reduce the size of the search space when generating test data, thereby improving the efficiency of generating test data. McMinn et al. (2012) introduced a search-based approach for generating human readable test cases involving string inputs by formulating web queries. Arcuri and Yao (2008) applied search algorithms to automatically generate test data of object-oriented software, and designed the fitness function based on the targets to be covered and the length of a test sequence. Arcuri (2010) studied the role of the length of a test sequence on branch coverage, and showed that a longer test sequence makes their testing trivial. However, a long test sequence is more difficult to be understood and analyzed. Fraser and Arcuri (2011) presented different techniques to overcome the problem of length bloat. To produce small yet representative test cases, Fraser and Arcuri (2012) proposed a search-based method in which whole test suites are evolved with the aim of covering all goals at the same time, while keeping the total size as small as possible. Gross et al. (in press) presented EXSYST, a system test generator for interactive Java programs, which uses a genetic algorithm to evolve a population of GUI test suites with the goal of achieving the maximal possible coverage.

The results above provide multiple approaches which greatly enrich the theory of software testing, and prompt the application of genetic algorithms in automatic generation of test data. Taking advantage of the fact that an individual in evolutionary process can be shared by multiple paths, namely, some of the required test data can be readily available as by-products when searching for other test data, Ahmed et al. converted the problem of generating test data into a multi-objective optimization one, and the number of objectives is equal to that of target paths. While adopting a genetic algorithm to solve the optimization problem above, the degree that an individual meets each objective is calculated. Hence Ahmed's method can synthesize multiple test data to cover multiple target paths in one run (Ahmed and Hermadi, 2008). Cao et al. (2010) presented an overlapping path structure to describe the paths of a program, and designed a method of generating test data covering multiple target paths in one search. Based on Huffman Coding, we used a genetic algorithm with an improved fitness function to generate test data covering multiple paths (Gong and Zhang, 2010). For symbolic execution of multiple paths, a typical static strategy of generating test data, Santelices and Harrold (2010) presented a technique which exploits control and data dependencies to iteratively partition paths into several groups to obtain the symbolic result. With dynamic methods of generating test data above (Ahmed and Hermadi, 2008; Cao et al., 2010; Gong and Zhang, 2010), the number of objectives of an optimization problem is equal to that of target paths. As for a program including dozen or even hundreds of target paths, the converted optimization problem will be very difficult to solve because of numerous objectives.

Aiming at the problem of generating test data covering many target paths, we ever divided these target paths into several groups according to their similarities, each group forming a sub-optimization problem independently, in which the number of objective functions is far less than that in the original problem, making the problem resolved more easily. Accordingly, a complicated optimization problem is transformed into several simpler sub-optimization problems, which are solved by using multi-population genetic algorithms to generate test data covering these target paths under serial environment (Gong et al., 2011). According to the schema theory for generating test data by genetic algorithms (Harman and McMinn, 2010), it was theoretically proved that grouping target paths according to their similarities can effectively increase the probability of finding test data.

Compared with serialization, parallelization of multi-population genetic algorithms will bring the parallel potential of genetic algorithms to the full more effectively. Studies on parallelization of genetic algorithms applied to automatically generate test data, however, are rarely reported. Alba and Chicano (2008) generated test data for branch coverage using parallel and serial genetic algorithms, respectively. Alshraideh et al. (2011) proposed a multi-population genetic algorithm to avoid local optima and plateau when searching for test data that cover the target branch. McMinn et al. presented a strategy of factoring out paths according to the searched target, and searched for test data for each path by dedicated genetic algorithms' operating

Table 8
Values of indicators for *insert sort*.

	Ltc(s)		Td(s)		Atc(s)		Ang		Apcr(%)	
	Avg.	*Vac*.	Avg.	*Vac*.	Avg.	*Vac*.	Avg.	*Vac*.	Avg.	Vac.
Method in Gong et al. (2011)	0.474	0.017	0.372	0.014	0.233	0.021	7112.22	1,098,318.46	96.3	0.001
Proposed method	0.404	0.012	0.277	0.018	0.224	0.021	7313.36	1,355,941.45	97.1	0.001
t	2.916		3.800		0.321		0.907		1.333	

Table 9
values of indicators for industrial programs.

	Ltc(s)		Td(s)		Atc(s)		Ang		Apcr(%)	
	Avg.	Vac.	Avg.	Vac.	Avg.	*Vac*.	Avg.	Vac.	Avg.	Vac.
flex										
Method in Gong et al. (2011)	2.027	0.002	1.055	0.902	1.256	0.584	10,000	0	38.3	0.005
Proposed method	2.019	0.002	1.049	0.928	1.247	0.574	10,000	0	40.0	0.006
t	1		0.031		0.059		0		1.214	
make										
Method in Gong et al. (2011)	1.503	0.005	0.790	0.471	1.107	0.155	9766.30	881,029.56	73.1	0.0011
Proposed method	1.338	0.010	1.150	0.067	0.763	0.022	6148.28	674,983.77	86.0	0.0004
t	9.705		3.495		5.830		20.509		25.80	
grep										
Method in Gong et al. (2011)	0.522	0.031	0.521	0.031	0.154	0.003	1969.97	356,581.02	99.7	0.00004
Proposed method	0.411	0.031	0.365	0.029	0.162	0.007	2880.10	603,299.34	99.7	0.00004
t	3.171		4.588		0.571		6.568		0	
sed										
Method in Gong et al. (2011)	0.055	0.0003	0.053	0.0003	0.019	0.0001	119.07	717.10	100	0
Proposed method	0.042	0.0003	0.031	0.0001	0.017	0.0003	154.27	1096.82	100	0
t	4.333		11		1		5.844		0	
space										
Method in Gong et al. (2011)	3.685	0.080	3.685	0.080	2.257	1.281	10,000	0	33.6	0.001
Proposed method	3.365	0.014	1.747	2.403	2.210	1.276	10,000	0	42.5	0.001
t	7.441		8.729		0.207		0		14.833	
bash										
Method in Gong et al. (2011)	0.748	0.005	0.748	0.005	0.199	0.003	3899.40	105,963.72	78.1	0.003
Proposed method	0.598	0.014	0.597	0.014	0.189	0.008	4140.06	472,351.73	87.3	0.002
t	7.894		7.947		0.714		2.237		9.200	

in parallel. Therefore, feasible paths contribute more toward the search landscape, improving the efficiency of generating test data (McMinn et al., 2006). Chen and Zhong (2008) used multi-population genetic algorithms implemented in Matlab to generate test data for path coverage, and the experimental results shows that multi-population genetic algorithms have a better performance than traditional single-population ones. The number of target paths to be covered, however, is only one.

Generating test data covering multiple paths using multi-population parallel genetic algorithms is considered in this paper. For the method in Gong et al. (2011) of generating test data covering multiple paths, the limitation of calculation resources available is not taken into account. Besides, different similarity thresholds which are empirical values make different results of grouping target paths. The greater the threshold is, the more the number of groups will be, and vice versa. As illustrated in Section 3.1, for example, target paths are divided into 14 groups when the threshold is set to 0.4; whereas the number of groups goes down to 8 for the threshold going down to 0.3. And when different base paths are selected with the same threshold of 0.3, target paths are divided into 9 groups. More importantly, there is a large difference in the number of target paths belonging to different groups, hence reducing the efficiency of parallelization of multi-population genetic algorithms. Section 3.1 still shows that the biggest difference in the number of target paths among different groups is 6 and 7 when the threshold is set to 0.4 and 0.3, respectively, and even worse it reaches to 29 for *grep* of Section 3.5. This study can be viewed as proposing the solution aiming to those problems. In this study, the proposed method evenly divides target paths into several groups according to calculation resources available such that the difference between any two groups is not more than one in the number of target paths, while ensuring a great similarity among target paths in the same group. Given the difference between the two methods of grouping target paths, the mathematical model for generating test data formulated in this study differs from the one in Gong et al. (2011) in the number of converted sub-optimization problems and that of objectives in each sub-optimization problem, although both convert an optimization problem with many objectives into several ones with fewer objectives.

5. Conclusions

As an important method of generating test data, generating test data covering multiple paths in parallel using multi-population genetic algorithms has gained widespread attentions in recent years, and achieved some valuable research results. Appropriately grouping target paths is the premise of efficiently generating test data. Unfortunately, effective methods of grouping target paths have been absent up to date.

We present a novel method of grouping target paths for evolutionarily generating test data covering multiple paths. Unlike the existing methods, the proposed method determines the number of target paths of each group according to calculation resources available, making a small difference in the number of target paths belonging to different groups. At the mean time, our method selects the base paths based on the maximum of the sum of path similarities, and the target paths belonging to the same group are determined by similarities between the base path and all paths, which makes a great similarity among target paths in the same group. After grouping these target paths, we build a mathematical model for generating test data covering multiple paths in parallel, and indicate the differences and relations between it and the model in Gong et al. (2011).

The proposed method is applied to one benchmark and six industrial programs, which is implemented in a cluster environment. The results show that compared with method in Gong et al. (2011), the proposed method effectively utilizes calculation resources available on the premise of meeting the requirement of path coverage, hence improving the efficiency of software testing. This study not only presents a novel method of automatically generating test data covering multiple paths, but also lays a foundation for the application of parallel computing in software testing.

Notably, the aim of grouping target paths is to enhance the efficiency in utilizing calculation resources available so that the efficiency of generating test data can be improved. When grouping target paths, this study only considers the number of target paths of each group and the similarity among paths, but neglects the difficulty in covering a path. Actually, the difficulty of covering different paths is very variable. For some paths, it is easy to generate test data to cover them, whereas, it may be a very different thing for others, indicating that although target paths are evenly divided using the proposed method, there may still be a great difference among time in generating test data covering different groups. At this point, if information of the process of generating test data is used to adjust the groups to which the remaining paths belong, the load of calculation resources will be balanced for a further step, and the overall time in generating test data will be reduced. Therefore, our further work will focus on studying appropriate methods of dynamically grouping target paths.

Acknowledgments

This study was jointly funded by National Natural Science Foundation of China with Grant No. 61075061, Natural Science Foundation of Jiangsu Province with Grant No. BK2010187 and Graduate Student Research Innovation Program of Jiangsu Province College with Grant No. CXZZ11_0292.

References

Ahmed, M.A., Hermadi, I., 2008. GA-based multiple paths test data generator. Computer & Operations Research 35 (10), 3107–3127.

Alba, E., Chicano, F., 2008. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers & Operations Research 35, 3161–3183.

Alshraideh, M., Mahafzah, B.A., Al-Sharaeh, S., 2011. A multiple-population genetic algorithm for branch coverage test data generation. Software Quality Journal 19 (3), 489–513.

Arcuri, A., Yao, X., 2008. Search based software testing of object-oriented containers. Information Sciences 178 (15), 3075–3095.

Arcuri, A., 2010. Longer is better: on the role of test sequence length in software testing. In: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation, pp. 469–478.

Baars, A., Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., Tonella, P., Vos, T., 2011. Symbolic search based testing. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 51–62.

Beizer, B., 1990. Software Testing Techniques. International Thomson Computer Press.

Bueno, P.M.S., Jino, M., 2000. Identification of potentially infeasible program paths by monitoring the search for test data. In: Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering, pp. 209–218.

Bueno, P.M.S., Jino, M., 2002. Automatic test data generation for program path using genetic algorithms. International Journal of Software Engineering and Knowledge Engineering 12 (6), 691–709.

Bueno, P.M.S., Jino, M., Wong, W.E., 2011. Diversity oriented test data generation using metaheuristic search techniques. Information Sciences, http://dx.doi.org/10.1016/j.ins.2011.01.025.

Cao, Y., Hu, C.H., Chen, S.B., Li, L.M., 2010. Automatic test data generation for multiple paths and its applications. Computer Engineering and Applications 46 (27), 32–35.

Chen, Y., Zhong, Y., 2008. Automatic path-oriented test data generation using a multi-population genetic algorithm. In: Proceedings of the 4th International Conference on Natural Computation, pp. 566–570.

Fraser, G., Arcuri, A., 2011. It is not the length that matters, it is how you control it. In: Proceedings of IEEE International Conference on Software Testing, Verification and Validation, pp. 150–159.

Fraser, G., Arcuri, A., 2012. Whole test suite generation. IEEE Transactions on Software Engineering, http://dx.doi.org/10.1109/TSE.2012.14.

Girgis, M.R., 2005. Automatic test data generation for data flow testing using a genetic algorithm. Journal of Universal Computer Science 11 (5), 898–915.

Gong, D.W., Zhang, Y., 2010. Novel evolutionary generation approach of test data for multiple paths. Acta Electronica Sinica 38 (6), 1299–1304.

Gong, D.W., Zhang, W.Q., Yao, X.J., 2011. Evolutionary generation of test data for many paths coverage based on grouping. Journal of Systems and Software 84 (12), 2222–2233.

Gross, F., Fraser, G., Zeller, A. EXSYST: search-based GUI testing. In: Proceedings of the 34th International Conference on Software Engineering, in press.

Harman, M., McMinn, P., 2010. A theoretical and empirical study of search based testing: local, global and hybrid search. IEEE Transactions on Software Engineering 36 (2), 226–247.

Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., Wegener, J., 2007. The impact of input domain reduction on search-based test data generation. In: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 155–164.

Holland, J.H., 1975. Adaption in Natural and Artificial Systems. MIT Press, Ann Arbor.

McMinn, P., Harman, M., Binkley, D., Tonella, P., 2006. The species per path approach to search-based test data generation. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 13–24.

McMinn, P., Shahbaz, M., Stevenson, M., 2012. Search-based test input generation for string data types using the results of web queries. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation, pp. 141–150.

McMinn, P., 2004. Search-based software test data generation: a survey. Software Testing, Verification and Reliability 14 (2), 105–156.

Pargas, R.P., Harrold, M.J., Peck, R.R., 1999. Test data generation using genetic algorithms. Journal of Software Testing, Verification, and Reliability 9 (4), 263–282.

Santelices, R., Harrold, M.J., 2010. Exploiting program dependencies for scalable multiple-path symbolic execution. In: Proceedings of the 19th International Symposium on Software Testing and Analysis, pp. 195–206.

Shan, J.H., Wang, J., Qi, Z.C., 2004. Survey on path-wise automatic generation of test data. Acta Electronica Sinica 32 (1), 109–113.

Shan, J.H., Jiang, Y., Sun, P., 2005. Research progress of software testing. Acta Scientiarum Naturalium Universitatis Pekinensis 41 (1), 134–145.

Sthamer, H., 1996. The automatic generation of software test data using genetic algorithms. Dissertation of University of Glamorgan, UK.

Wegener, J., Buhr, K., Pohlheim, H., 2002. Automatic test data generation for structural testing of embedded software systems by evolutionary testing. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1233–1240.

Xanthakis, S., Ellis, C., Skourlas, C., 1992. Application of genetic algorithms to software testing. In: Proceedings of the 5th International Conference on Software Engineering and its Applications, pp. 625–636.

Xie, X.Y., Xu, B.W., Shi, L., Nie, C.H., 2008. Survey of evolutionary testing. Journal of Frontiers of Computer Science and Technology 2 (5), 449–466.

Xu, Z.R., 2007. Software Reliability Engineering, 1st ed. Tsinghua University Press, Beijing.

Dunwei Gong received Ph.D in control theory and control engineering from China University of Mining and Technology, Xuzhou, Jiangsu, P R China, in 1999. Since 2004, he has been a professor and taken up the director of Institute of Automation, China University of Mining and Technology. His research interest is search-based software engineering.

Tian Tian received M.S. degree in computer application technology from Qufu Normal University, Rizhao, Shandong, P R China, in 2010. Since 2010, she has been a PH.D candidate in China University of Mining and Technology. Her research interest is genetic algorithms and software testing.

Xiangjuan Yao received Ph.D in control theory and control engineering from China University of Mining and Technology, Xuzhou, Jiangsu, P R China, in 2011. Since 2008, she has been an associate professor in China University of Mining and Technology. Her research interest is generation of test data of complex software.