Generating Test Data for Path Coverage Based on Genetic Algorithm

Shuping Fan[1], Baoying Ma[2], Nianmin Yao[3], Yan Zhang[1*], Chunyan Xia[1] and Dan Zhang[1]

[1] School of Computer and Information Technology, Mudanjiang Normal University, Mudanjiang 157011, China
[2] School of Health Management, Mudanjiang Medical University, Mudanjiang 157011, China
[3] School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China
Corresponding author: Yan Zhang (E-mail: zhangyan@mdjnu.cn)

Abstract. Test data generation is an important part of software testing. The imbalance of data crossing program branches is often ignored in generating test data. As a result, there is much data crossing some branches while little data crossing other branches. To solve the phenomenon so as to generate test data effectively, we introduce branch balance and program balance in the evolutionary generation of test data. First, the number of individuals crossing the true and false branch of each branch node on the target path are computed. Then, the calculation methods of branch balance and program balance are given. Finally, the fitness value function which considers the change of program balance before and after an individual joining is presented. And an individual that can improve the balance will be retained in the evolution process. Experiments show that our method is better than the other method in running time and success rate.

1. Introduction

Software testing is a time-consuming and labor-consuming work, whose goal is to find as many errors as possible with less test data to reduce the cost of software development. An efficient and feasible method to generate test data automatically is of great significance to test process [1]. Path coverage is a test criteria with the highest coverage in white-box testing, which refers to selecting enough test data so that every possible path of the program can be executed at least once. In this paper, we consider the balance of individuals crossing program branches, which is used to generate test data in Genetic Algorithm (GA). So an individual crossing new branch or improving the program balance will obtain higher fitness value and it will be preserved in the evolution process to enhance the efficiency of test data generation.

2. Related Work

In recent years, many scholars have studied the application of evolutionary theory to generate test data satisfying path coverage, and they have developed many new methods for automatically generating test data covering target paths. Ahmed et al. [2] proposed a method to generate multi-path test data for the first time by transforming the test data generation problem into a multi-objective optimization problem, which enables the application of GA to generate multiple test data crossing multiple target paths at one time. Gong et al. [3] presented to generate regression test data based on the existing test

data. Dang et al. [4] proposed to generate executable paths for weak mutation testing by examining the correlation between the true branches of mutation statements, so that test data covering these paths can kill all mutants. Ding et al. [5] used a complete model for fast generation of test data from the perspective of engineering practice of software testing, which realized the path representation for key point. It improved the efficiency for test data generation. In addition, some scholars have discussed the application of other technologies in test data generation. For example, Yao et al. [6] proposed to integrate with neural network, which can effectively reduce the time consumption of running programs comparing with previous methods. Reference [7] applied the capture technology of rare data, which protected these data traversing nodes that are difficult to cover, and then they calculated the individual contribution degree to the generation test data to adjust the individual fitness value.

The goal of software testing is to realize the sufficiency and comprehensiveness of program testing. Though the above techniques effectively generate test data covering target path by applying evolutionary algorithm, they do not consider the balance of test data traversing program branches. If a balanced mechanism is introduced during the evolution of test data so that the generated data can traverse multiple branches in a balanced way, thus the generation efficiency of test data covering target path will undoubtedly be improved.

Therefore, we introduced the balanced mechanism in the evolution of test data. After test data running the program, the number of individuals traversing each branch node on target path is counted, and the fitness function is designed according to it so individuals that can improve the program balance are retained to improve the generation efficiency of test data.

3. Basic Concepts

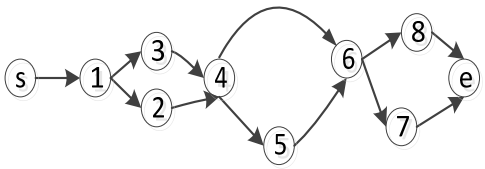
For convenience, the concept of Control Flow Diagram is given.

(1) Control Flow Diagram

Control Flow Diagram called CFD is a graphical representation of the program control structure, which is a directed graph G (*N*, *E*, s, e), where *N* is called the set of nodes in G and each node corresponds to a program statement, and *E* is a set of edges. (n_i, n_j) is called a edge of G, and it indicates that there is control flow from n_i to n_j. Generally, the CFD of each program contains a unique entry node s and exit node e. The Figure 1(b) shows the CFD of the program, and the source program is in Figure 1(a) [8].

```
void Function( )
{
1  if(a>2)
2     c=a;
   else
3     c=b;
4  if(a+b-c>1)
5     w=a-b;
6  if(w >5)
7     printf("It is OK");
   else
8     printf("error");
}
```

(a) Source program



(b) Control Flow Diagram

Figure 1. Source program and its CFD

(2) Branch nodes

In the CFD, nodes with an out-degree greater than or equal to two are called branch nodes. $node_1, node_4, node_6$ in Figure 1(b) are all branch nodes. As the switch statement can be expressed as a double-branch selection structure, and according to Z-path coverage [9], the loop structure can also be converted into a double-branch selection structure basing on the number of times the loop body executing zero and at least once. Therefore, switch nodes and loop nodes are all regarded as branch nodes in this paper.

(3) True and false branches of branch nodes

The two edges starting from the branch node are the true and false branches of the branch node respectively. When the predicate of a branch node is true, the true branch of the branch node is executed, otherwise the false branch is executed.

4. Calculation of the Program Balance

From the CFD in Figure 1(b), it can be seen that the number of test data crossing the sequence node and its direct successor node must be equal. Therefore, the sequence nodes are not considered in the calculation of the program balance to reduce calculation cost. After all the corresponding data of individuals run the program, the program balance is calculated. According to individuals crossing the true and false branches of each branch node, first we calculate the branch balance of each branch node, then we take the sum of all the branch balance as the program balance.

4.1. Calculation of Branch Balance

The branch balance is used to indicate the balance of test data crossing the true and false branches of a branch node. Supposing the number of branch nodes on target path is n. After the corresponding data of all individuals run the program, the number of individuals crossing the true and false branch of the kth branch node in the tth generation population is counted. Here denote it as $Num_{kT}(t)$ and $Num_{kF}(t)$ respectively, so the branch balance of the kth ($k = 1,2\cdots,n$) branch node can be expressed as:

$$bb_k(t) = \begin{cases} 0 & , Num_{kT}(t) = Num_{kF}(t) = 0 \\ |Num_{kT}(t) - Num_{kF}(t)|/max(Num_{kT}(t), Num_{kF}(t)), & \text{else} \end{cases} \quad (1)$$

Where $max(Num_{kT}(t), Num_{kF}(t))$ is to get the maximum value of $Num_{kT}(t)$ and $Num_{kF}(t)$. From Eq. (1), we can see that the defined branch balance actually reflects the balance of all individuals crossing the true and false branches of the kth branch node. If the difference in the number of individuals crossing the true and false branches is small, the value is small, which indicates that the individuals crossing the true and false branches of the node are more evenly. It is not difficult to obtain that the smaller of the branch balance, the better.

4.2. Calculation of Program Balance

According to Eq. (1), the branch balance of all branch nodes on the target path is calculated, and the sum of the branch balance of all branch nodes is taken as the program balance, as is seen in Eq. (2). According to the definition of the branch balance, the smaller the program balance is, the better of the program balance is. The smaller value reflects data crossing the program in a balanced way.

$$pb(t) = \sum_{k=1}^{n} bb_k(t) \quad (2)$$

Taking the program in Figure 1 as an example. Supposing the chosen target path is “s,1,3,4,6,8, e”, which means that there are three branch nodes on the path, they are $node_1$, $node_4, node_6$. Assuming there are four individuals $x_1 \sim x_4$ in the population, and the number of individuals crossing the true branch and the false branch of the branch node of $node_1$ in the 6th generation are $Num_{1T}(6)$ = 3 and $Num_{1F}(6)$ = 1 respectively. The branch balance of $node_1$ is further calculated according to Eq. (1):$bb_1(6) = |Num_{1T}(6) - Num_{1F}(6)|/max(Num_{1T}(6), Num_{1F}(6)) = |3-1|/\max(3,1) \approx 0.67$.

Assuming that the number of individuals crossing the true branch of node_4 and node_6 are 2, 4 respectively, crossing the false branch of these two nodes are 2 and 0 respectively. Then the branch balance of the kth ($k = 2,3$) branch node can be calculated in the same way. The values are $bb_2(6)$=0 and $bb_3(6)$=1. Finally, according to the calculation method in Eq. (2), the program balance is achieved. $pb(6) = \sum_{k=1}^{6} bb_k(6) \approx 0.67 + 0 + 1 \approx 1.67$.

5. Evolutionary Generation of Test Data

To check whether an individual can improve the program balance, the method used is: after deleting an individual, recalculate the branch balance of each branch node on the target path, then calculate the program balance. So the effect of the individual basing on the change of the program balance before and after deleting the individual can be computed, and finally the effect is taken as fitness value of the individual.

5.1. Calculation of Program Balance after an Individual Deletion

To calculate the fitness value of an individual, this section gives the calculation method of program balance after deleting an individual. Supposing the number of individuals crossing the true branch and false branch of the *k*th branch node after deleting the individual x_w is $Num'_{kT}(x_w,t)$ and $Num'_{kF}(x_w,t)$ respectively, the branch balance of the *k*th branch node can be expressed as $bb'_k(x_w,t)$:

$$bb'_k(x_w,t)=\begin{cases}0\ , Num'_{kT}(x_w,t)=Num'_{kF}(x_w,t)=0\\ |Num'_{kT}(x_w,t)-Num'_{kF}(x_w,t)|/max(Num'_{kT}(x_w,t),Num'_{kF}(x_w,t))\ , \text{else}\end{cases} \tag{3}$$

So we get the program balance after x_w is deleted, which can be expressed as:

$$pb'(x_w,t)=\sum_{k=1}^{n} bb'_k(x_w,t) \tag{4}$$

5.2. Calculation of Individual Fitness Value

Generally speaking, individuals with higher fitness value in current population will be copied to the next generation with large probability. In the paper, the calculation of an individual fitness value consider the program balance obtained before and after deleting an individual x_w. And according to the change on the program balance, we get the fitness value of x_w, which can be expressed as:

$$f(x_w,t)=\begin{cases}pb'_w(t)-pb(t), pb'_w(t)>pb(t)\\ 0, \qquad pb'_w(t)\le pb(t)\end{cases} \tag{5}$$

As is seen from Eq. (5) that the change of program balance before and after deleting x_w is considered when calculating the individual fitness value. If the deletion of x_w increases the program balance, that is, the existence of x_w can effectively improve the program balance, then x_w is preferentially retained in the evolution process. If there are multiple individuals like this at the same time, it can be ensured that the greater improvement on the program balance, the larger fitness value of an individual gets by Eq. (5). On the other hand, if the deletion of x_w leaves the program balance unchanged or reduced, that is, x_w cannot improve the program balance, then the individual should be deleted by setting its fitness value to 0.

Taking the program in Figure 1 as an example. In the 6th generation, when deleting x_1, assuming that the number of individuals crossing the true and false branch of $node_1$ is $Num'_{1T}(x_1,6)$=3, $Num'_{1F}(x_1,6)=0$ respectively, thus the branch balance of $node_1$ after deleting $x_1$can be calculated, $bb'_1(x_1,6)=|Num'_{1T}(x_1,6)-Num'_{1F}(x_1,6)|/max(Num'_{1T}(x_1,6),Num'_{1F}(x_1,6))=|3-0|/\max(3,0)=1$. In the same way, we get the branch balance of $node_4$ and $node_6$. So the program balance in this case can be calculated. Supposing the program balance after deleting x_1 by Eq. (4) is : $pb'(x_1,6)=\sum_{k=1}^{n} bb'_k(x_1,t)=1$. And the program balance after deleting other individuals can be calculated in the same way, supposing the program balance after deleting x_2~x_4 are $pb'(x_2,6)=3$, $pb'(x_3,6)=2$, and $pb'(x_4,6)=2.5$ respectively. As the program balance calculated before deleting x_1 is $\mathrm{pb}(6)\approx 1.67$, and the value after deleting it is $pb'(x_1,6)=1$. Since $pb'(x_1,6)<pb(6)$, the fitness value of x_1 can be calculated according to Eq. (5), $f(x_1,6)=0$. Similarly, we get $f(x_2,6)=1.33$, $f(x_3,6)=0.33$,$f(x_4,6)=0.83$. As $f(x_2,6)>f(x_i,6)(i=1,2,3)$, it shows that x_2 can improve the original program balance to the greatest extent in all individuals. Besides, both $f(x_3,6)$ and $f(x_4,6)$ are greater than 0, which indicates that both x_3 and $x_4$can effectively improve the balance of program coverage. These three individuals will have a greater probability to be preserved in evolution process. The fitness value of x_1 is 0 and it will be deleted in the evolution

process. It can be seen that the proposed method can effectively distinguish the effect of different individuals on the program balance.

6. Experiments

To validate the effectiveness of our method, we apply it to triangle classifier program in [7], a benchmark program commonly used in software testing. The program is written in C language. The selected comparison method is described in reference [10]. The two methods adopt the same experimental parameters. Comparing the evaluation times, running time and success rate to find the test data covering the target path, which are used in [7]. The evaluation times means sum of evolutionary generations all individuals in one experiment. The smaller the evaluation times and the shorter the running time, the better the performance of the algorithm, and the success rate refers to the ratio of the number of experiments that successfully generate the test data to the total number of experiments within the maximum running generation. All results are the average values by many experiments. The experimental results are shown in Table 1.

Table 1. Experimental Results of Triangle Classification Program

Experimental settings			The proposed method			The method in [10]		
Data range	Population size	Maximum generation	Evaluation times	Time (ms)	Success Rate(%)	Evaluation times	Time (ms)	Success rate(%)
[1,128]	30	3000	4966.0	19.7	100	41974.0	84.9	100
[1,256]	50	8000	7750.0	20.5	100	337880.0	628.3	40
[1,512]	80	10000	11226.7	32.7	100	670917.3	1285.1	33.3
[1,1024]	150	30000	28690.0	74.4	100	3415370.0	7217.0	46.7
[1,2048]	300	80000	65360.0	256.7	100	8003560.0	19422.8	66.7

As can be seen from Table 1:

(1) When the data range is [1,128], the average value of evaluation times in [10] and our method are 41974.0 and 4966.0 respectively, which is more than our method. When the data range is [1,2048], the value in [10] is 8003560, which is also more than ours of 65360. For different experimental conditions in Table 1, the evaluation times of our method are obviously less than that of the other method. This shows that comparing with the similar method in [10], ours can successfully generate test data with less evaluation times. That is because less generations of GA is used to get the terminal conditions of GA by our method.

(2) The average running time required by our method is less than that of [10] for different experimental conditions. For example, When the data range is [1,2048], the average running time of [10] is 19422.8ms and 256.7ms of our method, which is less than the value in [10]. Besides, under different data ranges, population size and maximum evolution generations in Table 1, our method all consumes less time. This is because the method in [10] considers the layer proximity and branch distance when calculating the fitness value, which needs large amount of computation and consumes more time. Our method mainly needs to calculate the branch balance and program balance before and after an individual joining, which needs less time.

(3) From the average success rate, the value of the proposed method can all reach 100% in different conditions, while the method in [10] can't. For example, When the data range is [1,1024], the value of [10] is 46.7%, which is obvious lower than our method. It demonstrates the effectiveness of the proposed method in generating test data.

We can conclude from the experimental results above that compared with the method in [10] , our method is advantageous in evaluation times, running time and success rate for triangle classifier program.

7. Conclusion

The paper presents a test data generation method based on GA. After all the corresponding data of individuals run the program, the number of test data traversing each branch node on the target path is

counted. Based on this, a calculation method for the effect of individuals on the program balance is designed. The individual traversing the new path is preferentially retained. If all the generated data do not traverse the new path, the effect of each individual on the program balance is calculated. Furthermore, the fitness value of individuals which can improve the program balance and the test data generation efficiency is raised. So these individuals will be retained in evolution process. Experimental results show that the test data generation efficiency of our method is higher than that of the comparison method. However, the experimental results are just on the triangle classification program, and we will further study the experiments on other programs in the future.

8. Acknowledgement

This study was jointly funded by the Research Projects of Basic Scientific Research Business Expenses in Institutions of Higher Learning of Heilongjiang Province(Grant Nos.1353ZD003 and 2018-KYYWFMY-0104); the Scientific and Technological Plan Project of Mudanjiang City (Grant Nos.Z2018g023, Z2016s0027 and Z2018g022); the Science and technology research project of Mudanjiang Normal University(Grant No.YB2019003);the National Natural Science Foundation of China (Grant No. 61573362); the Natural Science Foundation of Heilongjiang(Grant No. F2016039); and the Innovation Foundation of Science and Technology of Dalian (Grant No. 2018J12GX045).

9. References

[1] MIAO X X, ZENG P J. Research on an Automatic Generation Algorithm of Test Data[J]. Measurement and control technology. 2018, 37, 36-38.
[2] Ahmed M A, Hermadi I. GA-based multiple paths test data generator [J]. Computers and Operations Research. 2008, 35(10): 3107-3124.
[3] Gong D W, Ren L. Evolutionary Generation of Regression Test Data [J]. Chinese Journal of Computers. 2014, 3: 489-499.
[4] Dang X Y, Gong D W, Yao X J. Feasible path generation of weak mutation testing based on statistical analysis [J]. Chinese Journal of Computers. 2016, 39(19): 1-17.
[5] Ding R, Dong H B, Zhang Y. Fast Automatic Generation Method for Software Testing Data Based on Key-Point Path[J]. Journal of Software, 2016, 27(4): 814-827.
[6] Yao X J, Gong D W, Li B. Evolutional Test Data Generation for Path Coverage by Integrating Neural Network[J]. Journal of Software. 2016, 27(4): 828-838.
[7] Zhang Y, Gong D W. Evolutionary Generation of Test Data for Paths Coverage Based on Scarce Data Capturing[J]. Chinese Journal of Computers. 2013, 36(12): 2429-2439.
[8] Xia C Y, Zhang Y, Song L. Evolutionary generation of test data for paths coverage based on node probability[J]. Journal of Software, 2016,27(4):802-813
[9] Xia H, Song X, Wang L. Research of test case auto generating based on Z path coverage[J]. Modem Electronics Technique. 2006, 6: 92-94.
[10] McMinn P. Evolutionary search for test data in the presence of state behavior [D]. University of Sheffield, England, 2005