Evolutionary generation of test data for multiple paths coverage with faults detection

Yan Zhang [#*1], Dunwei Gong [#2]

[#] *School of Information and Electrical Engineering, China University of Mining and Technology*
Xuzhou, Jiangsu, 221116, P. R. China
[1] zhangyancumt@126.com
[2] dwgong@vip.163.com
[*] *Department of Computer Science and Technology, Mudanjiang Normal University*
Mudanjiang, Heilongjiang, 157012, P. R. China

***Abstract*—The aim of software testing is to find faults in the program under test. Generating test data which can reveal faults is the core issue. Although existing methods of path-oriented testing can generate test data which traverse target paths, they cannot guarantee that the data find the faults in the program. In this paper, we transform the problem into a multi-objective optimization problem with constrains and propose a method of evolutionary generation of test data for multiple paths coverage with faults detection. First, we establish the mathematical model of this problem and then a strategy based on multi-objective genetic algorithms is given. Finally we apply the proposed method in some programs under test and the experimental results validate that our method can find specified faults effectively. Compared with other methods of test data generation for multiple paths coverage, our method has greater advantage in faults detection and testing efficiency.**

I. INTRODUCTION

Test data generation is an important issue in software testing. Shan et al.[1] proposed that many problems of software testing can come down to the ones of generating test data for path coverage. These problems can be described as follows: for a target path of a program under test, search for a test datum in the input domain of the program so that the traversed path of the test datum is just the target one.

In recent years, various evolutionary algorithms have been employed in automatic generation of test data, among which genetic algorithms (GAs) are the most widely used [2]. Miller et al. used GAs and program dependence graphs to generate test data, leading to an effective result in path coverage [3]. Ahmed et al. proposed a method of searching for test data to traverse multiple paths in one GA run [4]. We presented an efficient approach to generate test data for multiple paths coverage where all target paths are encoded and the fitness of an individual is the degree of the traversed path matching the target ones. The method solves the generation of test data for multiple paths coverage of complex software [5]. These methods of path-oriented testing can generate test data to traverse target paths, however, these test data cannot guarantee to detect faults in target paths effectively. The aim of software testing is to find and modify faults in the program under test [6], so generating test data that can expose the faults of a program is very important.

978-1-4244-6439-5/10/$26.00 ©2010 IEEE

In software testing for path coverage, whether a fault can be detected depends on the probability of a fault being detected [7]. It is difficult for an ordinary method of software testing for path coverage to detect faults with a small probability. Therefore, we presented an evolutionary method of generating test data for path coverage with faults detection [8]. Employing the method above, the generated test data can not only traverse the target path, but also detect faults in the path effectively. However, the method above is only suitable for the case of one target path, i.e., one run of the method above can generate test data to traverse only one target path. Since many real-world programs under test often contain lots of target paths, if employing the method above, we have to run it for many times, suggesting that the method above is impracticable for real-world programs under test.

In addition, it is common that a fault may exist in several paths. If considering only one target path in each run of the method above, we have to tackle the same fault for several times, resulting in waste of time and resources, indicating that it is necessary to consider multiple paths coverage with faults detection.

The purpose of our study is to find effective methods of generating test data for multiple paths coverage with faults detection, which can not only guarantee that the test data traverse target paths but also detect faults as many as possible. To achieve that goal, first, we formulate the problem above as a bi-objective optimization problem with one constraint and propose a method of solving the model above, i.e., transforming multiple objectives into one objective by weighting, and changing the constraint into a penalty item. We apply the proposed method in the bubble sort program, and the experimental results show that our methods can not only generate test data which traverse multiple target paths rapidly but also generate high quality test data.

The main contributions of our study are as follows: (1) we construct a mathematical model of the problem of test data generation for multiple paths coverage with faults detection, which is a multi-objective optimization problem with constraints; (2) we present an efficient multi-objective evolutionary optimization algorithm to solve the problem above and design a function to calculate the fitness of an individual. Our method is of great significance in generating high quality test data for faults detection.

The rest of this paper is organized as follows: Section 2 constructs the mathematical model of the problem of test data generation for multiple paths coverage with faults detection; Section 3 presents a strategy of solving the model above; the application of our method in a benchmark program is given in Section 4; finally, Section 5 draws conclusions and provides possible opportunities for future research.

II. MATHEMATICAL MODEL OF TEST DATA GENERATION FOR MULTIPLE PATHS COVERAGE AND FAULTS DETECTION

Test data generation for multiple paths coverage with faults detection is to find test data which traverse all feasible paths and detect faults in them as many as possible. In this section, we will construct a mathematical model of the problem above.

Suppose that the number of target paths is n, where n is larger than one; x is a decision variable. Denote $p(x)$ as the traversed path by x and $|p(x)|$ as the number of nodes belonged to $p(x)$, determined by the representation means of a path. If a path's node is represented with a branch belonged to the path, $|p(x)|$ is the number of branches traversed by x; whereas if a path's node is represented with a statement's code, $|p(x)|$ is the number of codes x.

A. Mathematical model of test data generation for path coverage with faults detection

If we consider only one target path, denoted as p_0, the mathematical model of test data generation for path coverage with faults detection can be formulated as follows. (For details, please refer to [8].)

$$\begin{aligned} \max \quad & f_1(x) = \sum_{i=1}^{|p(x)|} r_i \\ & f_2(x) = \bigvee_{i=1}^{|p(x)|} (\eta_{i1} \vee \eta_{i2} \vee \cdots \vee \eta_{ir_i}), \qquad (1) \\ s.t. \quad & \frac{\alpha(x)}{|p_0|} = 1. \end{aligned}$$

where r_i is the number of faults detected in the *i*-th node with their risk levels being $\eta_{i1}, \eta_{i2}, \cdots, \eta_{ir_i}$, respectively. The total number of faults detected in $p(x)$ can be denoted as $f_1(x)$. The risk level of $p(x)$'s *i*-th node is equal to the largest one of $\eta_{i1}, \eta_{i2}, \cdots, \eta_{ir_i}$, i.e. $\eta_{i1} \vee \eta_{i2} \vee \cdots \vee \eta_{ir_i}$, where "$\vee$" denotes the maximum operator. The highest risk level of all faults in $p(x)$ is denoted as $f_2(x)$. $\alpha(x)$ denotes the number of same nodes from the first node until the first different node between $p(x)$ and p_0. The constraint $\frac{\alpha(x)}{|p_0|} = 1$ indicates that all nodes in the path traversed by a test datum are the same as those in p_0.

B. Mathematical model of test data generation for multiple paths coverage with faults detection

The problem considered in our study is still to generate test data for faults detection. Different from the former case for one target path coverage, what we considered is for multiple paths coverage, i.e. the problem of test data generation for multiple paths coverage with faults detection.

In order to generate test data that cover multiple target paths and detect faults as many as possible, we still employ the two objectives in equation (1). Since the goal is to search for test data that traverse target paths and detect faults in them, we require that the path traversed by a test datum is just one of target paths.

In software testing, it is common that the same fault lies in several paths. If a test datum traverses a target path and detects a fault, it is not necessary to consider the same fault when generating test data in traversing other target paths. We employ a set to record the faults having been detected and only search for those faults that are not included in this set.

The set of all detected faults are denoted as S_{fault}. If a fault detected by a test datum does not belong to S_{fault}, it will be added into S_{fault} along with its type and the number of the node that it lies in.

We denote the set of faults that do not belong to S_{fault} in the *i*-th node as E_i, $|E_i|$ as the total number of its elements, and e_{ij} as the *j*-th fault that belongs to E_i with its risk level being $\eta_{e_{ij}}$. We have the following mathematical model of the problem above.

$$\begin{aligned} \max \quad & f_1(x) = \sum_{i=1}^{|p(x)|} |E_i|, \quad E_i \cap S_{fault} = \varnothing, \\ & f_2(x) = \bigvee_{i=1}^{|p(x)|} (\eta_{e_{i1}} \vee \eta_{e_{i2}} \vee \cdots \vee \eta_{e_{i|E_i|}}), \; e_{ij} \in E_i (j = 1, \cdots, |E_i|), \quad (2) \\ s.t. \quad & \bigvee_{k=1}^{n} \frac{\alpha(x)}{|p_k|} = 1. \end{aligned}$$

where p_k is the *k*-th target path, $k = 1, 2, \cdots, n$.

There are two differences between equation (2) and (1): one is that the constraint of equation (2) is for multiple paths, whereas that of equation (1) is for a single path; the other one is that although both two objectives of these two equations are about faults, equation (2) considers new faults, i.e. faults having been detected will be shielded successfully because only the faults that are not included in the fault set will be counted and the risk level of which will be evaluated.

III. EVOLUTIONARY GENERATION OF TEST DATA FOR MULTIPLE PATHS COVERAGE WITH FAULTS DETECTION

As we all know, the fitness of an individual is required in order to compare the quality of different individuals. Our method to solve the model is similar to that of reference [8]: first, we transform the weighted accumulation of two objectives into one objective and then transform the constraint as a penalty item into the objective, so we transform the optimization problem with one constraint into an optimization

problem without constraint. So the fitness of an individual is designed as follows:

A. Weighted Accumulation Objective

First, given different objectives have different ranges, we need to normalize each objective using an adaptive weight approach, i.e. adjusting each objective based on the maximal and the minimal values of the current population, then the adjusted objective function can be subjected to [0,1). For this purpose, we consider a population in some generation and denote the maximal and the minimal values of the *i*-th objective function as $f_i^{\max}$ and $f_i^{\min}$, respectively. The *i*-th objective function of x before and after adjustment are denoted as $f_i(x)$ and $f_i^{'}(x)$, respectively. We have:

$$f_i^{'}(x) = \frac{f_i(x) - f_i^{\min}}{f_i^{\max} - f_i^{\min} + \varepsilon}, \quad i = 1,2 \qquad (3)$$

where ε is a small positive value to guarantee that the denominator of equation (3) is larger than zero. On condition of no confusion, we still denote the value of the *i*-th objective after normalization as $f_i(x)$, and transform the problem into a single-objective problem using the weighted accumulation, shown as equation (4):

$$f(x) = \omega_1 f_1(x) + \omega_2 f_2(x) \qquad (4)$$

where ω_1 and ω_2 are the coefficients related to $f_1(x)$ and $f_2(x)$, respectively, and their values need to be set in prior.

B. Constraint Handling Based on Penalty

We transform the constraint into a penalty item and design the penalty function as equation (5):

$$\psi(x) = \bigvee_{k=1}^{n} \frac{\alpha_k(x)}{|p_k|} - 1 \qquad (5)$$

It is easy to observe that if the path traversed by the decision variable is the *k*-th target path, $\frac{\alpha_k(x)}{|p_k|}$ is equal to one, hence $\bigvee_{k=1}^{n} \frac{\alpha_k(x)}{|p_k|}$ is equal to one, making $\psi(x)$ to be zero. In this case, no penalty will be exerted to the test datum; on the contrary, if the traversed path is different from the target one, $\bigvee_{k=1}^{n} \frac{\alpha_k(x)}{|p_k|}$ will be smaller than one, making $\psi(x)$ to be negative. In this case, some penalties will be exerted to the test datum by the penalty item.

C. Computing Method of individual's fitness

For the sake of clearness, we still denote the coding of a decision variable as *x* and weight combination between the objective function and the penalty item, shown as equation (6):

$$\begin{aligned} fitness(x) &= F(x) + \omega_3 \psi(x) \\ &= \omega_1 f_1(x) + \omega_2 f_2(x) + \omega_3 \psi(x) \end{aligned} \qquad (6)$$

where ω_3 is a coefficient related to the penalty item, a positive value and set in prior.

D. Steps of Algorithm

The steps of our algorithm are as follows:

Step 1: Assign the values of the parameters used in our algorithm, encode the statements of the program under test, determine the target path, instrument the program under test and initialize the set of faults;

Step 2: Initialize a population;

Step 3: Decode an individual, execute the instrumented program, judge whether there exists an individual that traverses one target path. If yes, record the individual and the corresponding detected faults and remove the target path from the set of target paths. If there exists an individual that traverses one of the target paths that have been found, indicating that there exists a test datum having traversed the target path, judge whether the capability in detecting faults of the new individual is superior over that of the former one. If yes, replace the former individual with the new one; add the new faults detected by each new individual that traverses the target path into the fault set and count the number of new faults detected by each individual and evaluate their maximal risk level.

Step 4: Judge whether the termination criteria is met. If yes, go to Step 6;

Step 5: Calculate the fitness of an individual according to equation (6), perform selection, crossover and mutation operators to generate offspring, go to Step 3;

Step 6: Stop the evolution, decode the optimal solutions, and output test data, traversed paths, the number of faults and the risk level of the faults.

IV. Experiments and Analysis

In order to validate the effectiveness of our method, we apply it in test data generation for multiple paths coverage with faults detection. We choose the *bubble sort* program as the benchmark program. For details of the methods of faults detection, instrument and the representation of paths, please refer to reference [8].

To verify the performance of our method, we compare it, test data generation for multiple paths coverage with faults detection method (FDM, for short) with the random method (RM, for short) and the traditional one of test data generation for multiple paths without faults detection (TM, for short).

A. Implementation Platform and Parameter Settings

All programs under test are written in C language and run in the environment of VC++ 6.0. The hardware platform is a PC with 2.2 GHz Intel Core 2 Duo CPU and 3GB physical memory. Relevant parameter settings are different in different experiments; the same parameter settings of each experiment are as follows: roulette wheel selection, one-point crossover and one-point mutation. The probabilities of crossover and mutation are 0.9 and 0.3, respectively. The population size is 50 and the maximal number of generations is 2000.

In addition, the fault types and the risk level mentioned in the implementation are the same with those in reference [8]. They are UVF (uninitialized variable fault), OBAF (out of bounds array access fault), NPDF (null pointer dereference fault), ILCF (illegal computing fault) and DOF (data overflow fault), respectively. The risk level and quantitative value are listed in Table I.

TABLE I
RISK LEVEL AND QUANTITATIVE VALUE OF DIFFERENT FAULTS

Type	UVF	OBAF	NPDF	ILCF	DOF
Risk level	high	low	high	high	high
Quantitative value	0.9	0.5	0.9	0.9	0.9

In order to reduce the impact of randomicity on these experimental results, each experiment is conducted for 15 independent runs. In each run, a program will end only when all target paths have been found or the population has evolved for the maximal number of generations. The number of faults detected in the traversed path and their maximal risk level are calculated. The time consumption, the number of generations and the successful rate of detecting faults of each run are recorded, and then their averages of 15 independent runs are calculated.

B. Experiments on Bubble Sort Program

In this experiment, we select the *bubble sort* program, shown as Fig. 1, as the program under test, consider eight integers to be sorted and set the range of all these integers to be [0, 6,000]. Since statement 2 is "*short num* = 0; ", when the sum of all elements in array *a* exceeds 32767, a DOF appears; besides, since statement 10 is "*a[j+1]=t;*", an OBAF appears when *j* is equal to *N-1*. So the program in Fig. 1 has two faults by static analysis. The performance of our method is efficient if the generated test data can detect the faults above effectively.

```
float bubble(short a[N])
  {
1   short i , j , t , n ;
2   short sum=0 ;
3   for(i=0;i<N;i++)//①
4     sum+=a[i]; //DOF
5   for(i=0;i<N-1;i++)//②
6   for(j=N-1;j>i;j--) //③
7   if(a[j]<a[j-1]) //④
8     { t=a[j];
9       a[j]=a[j-1];
10    a[j+1]=t; }//OBAF
11  rutum 1.0*sum/N;
    }
```

Fig. 1. *Bubble sort* program

1) Comparison of different methods

First, three paths are selected as target paths, in which path 1 is the one with all "if statements" being false in statement 7 of Fig. 1(a), path 2 is the one with all "if statements" being true and path 3 is the one with part of "if statements" being true and the other part of "if statements" being false. We investigate the performance of our method with the values of ω_1 , ω_2 and ω_3 being one, one and one, respectively. The experimental results of three methods are shown as Fig. 2.

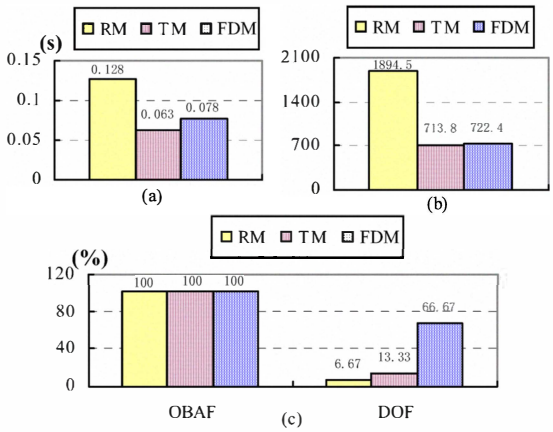

Fig. 2 Performance of different methods (three target paths) where (a) time consumption, (b) number of generations and (c) successful rate of detecting faults.

We can observe from Fig. 2(a) and (b) that our method is close to the traditional one and better than the random one in time consumption and the number of generations, indicating that our method is efficient in generating test data for paths coverage.

But for the successful rate of detecting faults, we can observe from Fig. 2(c) that the detection efficiency of different types of faults is different by different methods. For instance, the OBAF can be detected by any test datum that traverses the target path which includes it, therefore the successful rate of all these methods in detecting it is 100 percent; whereas for DOF, the successful rate of our method is higher than that of the traditional and the random ones, indicating that our method is efficient in generating test data for faults detection, especially for the fault with a small probability, such as DOF.

In order to evaluate the performance of our method in even larger number of target paths, we increase the number of target paths to ten. The experimental results of three methods are shown as Fig. 3.

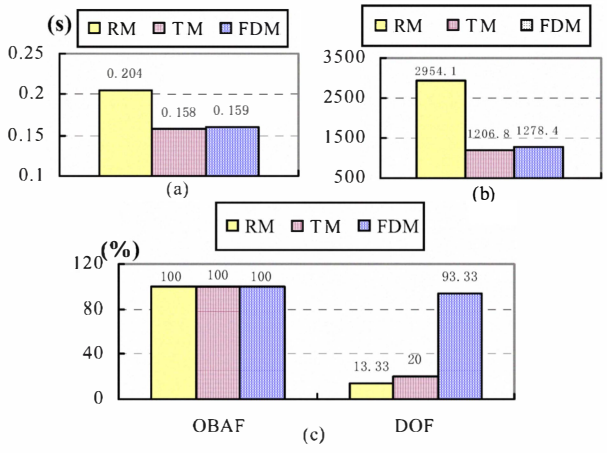

Fig. 3. Performance of different methods (ten target paths) where (a) time consumption, (b) number of generations and (c) successful rate of detecting fault.

We can observe from Fig. 3 that the time consumption and number of generations of all methods increase compared with those of the experiments with three target paths (shown as Fig. 2), e.g. time consumption of our method is 0.159s in the experiment with ten target paths, whereas that with three target paths is only 0.078s, indicating that the more the target paths, the larger the numbers of generations and the more the time consumption.

Our method is higher than the random and the traditional ones in successful rate of detecting DOF, suggesting that our method is effective in faults detection for multiple target paths.

We can conclude that, compared with the other two methods, our method can generate test data that both traverse the target paths and detect faults in them with a higher successful rate.

2) Effect of different values of weights on performance of our method

In order to validate the effect of different values of these weights on the performance of our methods, we set the *bubble sort* program as the benchmark program, select three target paths of it, use our method with the values of weights being 8:8:1, 5:5:1, 1:1:1, 1:1:5, 1:1:8, respectively. The experimental results are listed in Table II.

TABLE II
EFFECT OF VALUES OF WEIGHTS ON PERFORMANCE OF OUR METHOD

weight	1:1:8	1:1:5	1:1:1	5:5:1	8:8:1
time consumption(s)	0.068	0.065	0.078	0.101	0.204
number of generations	596.4	663.7	722.4	1223.2	1676.6
successful rate of detecting DOF(%)	46.67	60	66.67	86.67	100

It can be observed from Table II that the larger the values of these weights related to faults, the larger the number of generations, and the more time consumption required to generate test data, as a result, the higher the successful rate of detecting DOF. The highest successful rate of our method reaches to 100 percent with the values of $\omega_1, \omega_2, \omega_3$ in equation (6) being eight, eight and one, respectively. Therefore our method is efficient in faults detection for multiple target paths coverage given that the values of these weights related to faults are enlarged appropriately.

V. CONCLUSION

The problem of evolutionary generation of test data for multiple paths coverage with faults detection is discussed in our study. Our method is well-directed because it not only determines the path to be traversed but also counts the number of faults detected in the path and evaluates the risk level of these faults. Our method is efficient in generating test data. The traditional method hardly detects the hidden faults while only generates test data for path coverage, but our method not only overcomes this shortcoming but also solves the problem of redundant search in evolutionary generation of test data for single path coverage with faults detection.

First, we establish a mathematical model for the problem. In order to guarantee that a test datum traverses one of the target paths, we select the approach degree as the constraint; furthermore, for the purpose of detecting faults as more as possible and judging the risk level of these faults as higher as possible, we choose the number of faults detected in the path traversed by a test datum and the risk level of these faults as the objectives, thus, the problem of paths coverage with faults detection is transformed into a multi-objective optimization problem with one constraint.

We use the method of multi-objective evolutionary optimization with constraints to solve the problem above, transform multiple objectives into a single objective according to weighted accumulation and tackle the constraint by a penalty approach. The fitness of an individual does well in guiding the process of test data generation as expected.

We applied the proposed method in the *bubble sort* program and compared it with the random method and the traditional one without faults detection; in addition, we investigate the performance of our method with different values of weights. The experimental results confirm that our method is effective and has a higher successful rate in detecting specified type of faults.

However, the type of faults considered in our study is limited. To investigate more types of faults and test more complex programs are our future research topics.

ACKNOWLEDGMENT

This study was jointly funded by the Six Peak Talents Projects of Jiangsu Province with grant No. 2008125, the Youth Research Funds of Mudanjiang Normal University with grant No. G2008005, and the Youth Research Funds of China University of Mining and Technology with grant No. 2008A034.

REFERENCES

[1] J. H. Shan, J. Wang, Z. C. Qi, "Survey on Path-Wise Automatic *Generation* of Test Data," *Acta Electronica Sinica*, vol. 32, no. 1, pp. 109-113, 2004.

[2] M. Harman, A. Mansouri, and Y. Zhang, Search based software engineering: A comprehensive analysis and review of trends techniques and applications, *Department of Computer Science, King's College London, Technical Report TR-09-03*, 2009.

[3] J. Miller, M. Reformat, H. Zhang, "Automatic test data generation using genetic algorithm and program dependence graphs," Information and Software Technology, vol. 48, no. 7, pp. 586-605, 2006.

[4] M. A. Ahmed, I. Hermadi, "GA-based multiple paths test data generator," Computer & Operations Research, vol. 35, no. 10, pp. 3107-3127, 2008.

[5] D. W. Gong, Y. Zhang, "Novel evolutionary generation approach to test data for multiple paths coverage", *Acta Electronica Sinica*, vol. 38, no. 6, pp. 1299-1304, 2010.

[6] J. A.Whittaker, "What is software testing? And why is it so hard?" IEEE Software, vol. 17, no. 1, pp. 70-79, 2000.

[7] Y. Z. Gong, "An introduction to the software technique oriented software faults test model", *Journal of Academy of Armored Force Engineering*, vol. 18, no. 1, pp. 21-25, 2004.

[8] D. W. Gong, Y. Zhang, "Evolutionary generation of test data for path coverage with fault detection," under review.