Chinese Journal of Electronics
Vol.24, No.1, Jan. 2015

Test Data Generation for Multiple Paths Based on Local Evolution*

YAO Xiangjuan[1], GONG Dunwei[2] and WANG Wenliang[1]

(1. *College of Science, China University of Mining and Technology, Xuzhou 221116, China*)

(2. *School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, China*)

Abstract — Generating test data by genetic algorithms is a promising research direction in software testing, among which path coverage is an important test method. The efficiency of test data generation for multi-path coverage needs to be further improved. We propose a test data generation method for multi-path coverage based on a genetic algorithm with local evolution. The mathematical model is established for all target paths, while in the algorithm the individuals are evolved locally according to different objective functions. We can improve the utilization efficiency of test data. The computation cost can be reduced by using fitness functions of different granularity in different phases of the algorithm.

Key words — Test data generation, Path coverage, Genetic algorithm, Local evolution.

I. Introduction

Software testing is an expensive, tedious and labor-intensive task, and accounts for up to 50% of the total cost of software development[1]. One of the most important issues in software testing is the generation of effective test data satisfying the selected test adequacy criteria.

The common test adequacy criteria include statement coverage, branch coverage, condition coverage and path coverage, etc. It has been proved that many testing methods can be transformed into that of generating test data for paths coverage[2,3]: for a given target path, search for a test datum in the input domain of the program, such that the traversed path of the test datum is just the desired one.

Test data generation for path coverage remains a challenging problem in software testing. If we can generate test data automatically, it will undoubtedly reduce the cost of software testing, so as to improve the quality of software. Using the Genetic algorithm (GA) to solve the test data generation problem for complex software is a promising direction in recent years.

Most GA-based test data generation methods for path coverage intend to cover target paths one by one. When there are many target paths to be covered, the generator has to be run many times, which makes the existing test data generators inefficient for multiple paths coverage. In fact, the generated test data when trying to cover one path may just cover other target paths. So we can improve the efficiency of test data generation by considering multi-path at the same time.

In view of this, we propose a multi-path generation method based on a GA with local evolution. First, the optimization model for the problem of covering multiple target paths is established, in which coarse-grained functions are used as the objective functions. Second, we propose a GA with local evolution to solve the optimization problem. In each evolutionary phase of the algorithm, individuals are divided into several groups according to the objective functions. Then evolutionary operators are implemented to the individuals in each group. After evolution, the individuals are selected according to the fine-grained fitness functions. The proposed method can improve the utilization efficiency of test data by considering multi-path at the same time. In addition, the computation cost can be reduced by using fitness functions of different granularity in different phases of the algorithm.

The rest of the paper is organized as follows: Section II briefly reviews the related works on GA-based software testing. Section III gives some basic concepts and presents the evaluation functions we will use later. The mathematical model of generating test data for multi-path coverage is established in Section IV. Section V proposes a GA with local evolution to solve the model. Experiments and conclusions are presented in Section VI and Section VII, respectively.

II. GA-Based Software Testing

GA is an efficient search-based optimization algorithm, and shows special advantage and efficiency in solving many complex problems[4]. To solve the problem of test data generation using GAs, we must formulate it as an optimization problem. The goal is to find a solution that is sufficiently good

*Manuscript Received Dec. 2012; Accepted Sept. 2014. This work is supported by Natural Science Foundation of China (No.61203304, No.61375067), Natural Science Foundation of Jiangsu Province (No.BK2012566), and the Fundamental Research Funds for the Central Universities (No.2012QNA41).

according to the defined objective function from the domain of program under test.

Since Xanthakis[5] firstly use the GA to generate test data for path coverage problem in 1992, the application of GAs has achieved great development in almost all fields of software testing.

Yao and Gong[6] proposed a testability transformation method based on dominant relations for statement coverage testing. The basic idea is: for a target statement involving flag variables, if another target statement (or target statement set) dominates the original one, then the original target statement is substituted with the new one to generate test data.

Miller *et al.*[7] used GAs to generate test data satisfying branch coverage criterion. In their work, the fitness values can be automatically obtained based on branch predicates in the program, and the experimental results show that the test suite obtained by GAs can achieve or be very close to branch coverage. Baars[8] presented an algorithm for constructing fitness functions that improve the efficiency of search-based testing when trying to generate branch adequate test data.

Michael *et al.*[9] used a GA to generate test data for condition coverage criterion. In their work the function is minimized by using one of two genetic algorithms in place of the local minimization techniques used in earlier research.

Bühler and Wegener[10] applied an evolutionary algorithm to functional testing. Watkins and Hufnagel[11] used two GAs to generate a couple of test data, and then trained a decision tree using them, in order to obtain an agent model which distinguishes the merit of test data.

Automatic test data generation for paths coverage remains a challenging problem[12]. Zhang and Gong[13] presented a method of generating test data for path coverage based on automatic reduction of search space. Bueno and Jino[14], and Watkins and Hufnagel[15] also applied GAs to generate test data fulfilling path coverage.

Hermadi *et al.*[16] has observed that existing GA-based test data generators can generate only one test datum for one test goal at a time. When there are many target paths to be covered, the generator has to be run many times. In fact, the generated individuals when trying to find test data covering a path may be the just test data covering other target paths. This, hence, makes those existing test data generators inefficient in trying to generate test data for multiple paths.

Wegener *et al.*[17] developed a fully automatic GA-based test data generator for path coverage problem. In their approach, all generated individuals are evaluated with regard to all unachieved partial aims. But they only considered one partial aim for optimization at a time, which means that they solved the problems of generating test data one by one.

Ahmed and Hermadi[18] proposed a GA-based test data generator for multiple paths. In their work, the problem of generating test data for multiple paths is regarded as a multi-objective optimization problem, and solved by a multi-objective evolutionary algorithm. In fact, the problem of generating test data for multiple paths is strictly different from traditional multi-objective optimization problems. Therefore, it is necessary to solve the multi-path coverage problem using other methods.

Gong and Zhang[19] also proposed a test data generation method for milt-path coverage. They represent a target path using Huffman encoding method, and design the fitness function according to the Huffman codes of target paths. Their method is simple and has better performance than Ahmed's method, but the fitness function cannot distinguish individuals well.

III. Foundation

In this section, we first introduce several concepts. Then, the evaluation functions are presented.

1. Basic concepts

The Control flow graph (CFG) of a program Φ is a directed graph $D = (N, E, s, e)$, where N is the set of nodes, E is the set of edges, s and e are unique entry and exit nodes of the graph, respectively. Each node corresponds to a statement in the program, and each edge (n_i, n_j) represents a transfer of control from node n_i to node n_j.

A path of the CFG is a sequence $P = n_1, n_2, \cdots, n_k$, such that there exists an edge from node n_i to n_{i+1}, $i = 1, 2, \cdots, k-1$.

For large-scale programs, the sequence of a path may be very long. We represent a path using a (0, 1)-string for simplicity. Suppose that there are l conditional statements in path P, denoted as $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_l$. Define

$$\gamma_i = \begin{cases} 0, & P \text{ inludes the false branch of } \mathcal{C}_i \\ 1, & P \text{ inludes the true branch of } \mathcal{C}_i \end{cases}$$

Thus we obtain a (0, 1)-string $\gamma_1\gamma_2\cdots\gamma_l$ of length l. In program Φ, the mapping between a path and such a (0, 1)-string is one to one. So path P can be represented by $\gamma_1\gamma_2\cdots\gamma_l$, *i.e.* $P = \gamma_1\gamma_2\cdots\gamma_l$.

2. Evaluation functions

The key problem of applying GAs to test data generation is the construction of suitable evaluation function for the individuals. In this paper, we use two evaluation functions, one is the coarse-grained objective function that is taken in the mathematical model for the problem of multi-path coverage, and the other is the fine-grained fitness function that is used to select individuals during their evolution.

1) Coarse-grained evaluation function

Suppose that X is an input of program Φ, and P is a target path. The path traversed by X is denoted as $P(X)$. We define the approach level of input X for path P as the number of characters between the first dissimilar character of P and $P(X)$ to the last character of P, denoted as $f_P(X)$.

From the definition we can see that, the earlier path $P(X)$ deviates from target path P, the greater $f_P(X)$ will be. In particular, If $P(X) = P(X)$, then $f_P(X) = 0$ and X is just a test datum covering P. So the problem of generating test data for path P can be transformed into that of minimizing $f_P(X)$.

This evaluation function is coarse-grained, and therefore cannot distinguish individuals very well. But its calculation is relatively simple. So we use approach level as the objective function to decreasing the complexity of the mathematical model for multi-path coverage problem.

2) Fine-grained evaluation function

As we have said, the approach level cannot distinguish the test data accurately. So we will present a more exquisite evaluation function by combining approach level and branch distance.

The branch distance assesses how close the predicate comes to evaluating either true or false branch[20]. Branch distances of different kinds of simple branch conditions are listed in Table 1. For a complex branch condition, branch distance is the composite of those of all simple conditions included in it.

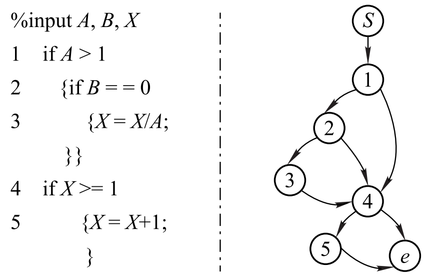
Table 1. Branch distances of simple branch conditions

Branch condition	$a \geq b$	$a > b$	$a = b$	$a \neq b$
True	0	0	0	0
False	$b-a$	$b-a+0.1$	$\|b-a\|$	1

Suppose that $B_P(X)$ is the branch distance of X to the branch from which $P(X)$ deviates P, then we define the fine-grained evaluation function of test datum X for target path P as follows:

$$g_P(X) = f_P(X) + (1 - 1.01^{-B_P(X)}) \tag{1}$$

$f_P(X) = 0$ if and only if the traversed path of X is P, *i.e.* $P(X) = P$; furthermore, the smaller the value of $f_P(X)$, the nearer X is to the data covering P.



```
%input A, B, X
1  if A > 1
2     {if B = = 0
3        {X = X/A;
     }}
4  if X >= 1
5     {X = X+1;
     }
```

Fig. 1. An example program and its CFG

For example, as shown in Fig.1. Suppose that the target path is $P = s\ 1\ 2\ 3\ 4\ 5\ e$. There are three conditional statements in P, *i.e.* statements 1, 2 and 4, respectively. P traverses the true branches of all these statements. So we also write $P = 111$. Suppose that $X_1 = (-10, 5, 10)$, $X_2 = (2, 5, 10)$, and $X_3 = (2, 1, 10)$. We obtain $P(X_1) = 01$, $P(X_2) = 101$ and $P(X_3) = 101$. Thus $f_P(X_1) = 3 - 0 = 3$, $f_P(X_2) = 3 - 1 = 2$, and $f_P(X_3) = 3 - 1 = 2$.

In addition, $P(X_1)$ deviates P from the first conditional statement, so the branch distance of X_1 to the first branch of P is $B_P(X_1) = 11$; similarly, we get $B_P(X_2) = 5$, and $B_P(X_3) = 1$. Thus

$$\begin{aligned} g_P(X_1) &= 3 + 1 - 1.01^{-11} = 3.1037 \\ g_P(X_2) &= 2 + 1 - 1.01^{-5} = 2.0485 \\ g_P(X_3) &= 2 + 1 - 1.01^{-1} = 2.0099 \end{aligned}$$

Although the approach level of X_2 and X_3 are the same, but the branch distance of X_3 is smaller than that of X_2. Thus X_3 obtains a better value than X_2 by the fine-grained evaluation function. We can see that the fine-grained evaluation function can distinguish the individuals better than the coarse-grained one, but needs more computation cost.

IV. Mathematical Model for the Problem of Multiple Paths Coverage

Suppose that there are n target paths for coverage, denoted as $P_1, P_2, \cdots, P_n$. Then the problem we will solve can be described as follows: find a test suite $\{X_1, X_2, \cdots, X_n\}$, such that $P(X_i) = P_i$. In order to improve the efficiency of test data generation, we will consider all these target paths at the same time. In the previous section, we construct two evaluation functions, *i.e.* the coarse-grained and fine-grained ones. If we use the fine-grained function to evaluate all test data for all target paths, the computation cost will be very huge. So we will use the coarse-grained function as the objective one in the mathematical model.

In view of this, let $f_i(X)$ be the coarse-grained evaluation function of test datum X for target path P_i, then the problem of test data generation to cover path $P_1, P_2, \cdots, P_n$ can be modeled as the following optimization problem:

$$\begin{cases} \min f_1(X_1) \\ \min f_2(X_2) \\ \cdots \\ \min f_n(X_n) \end{cases} \\ \text{s.t. } X_1, X_2, \cdots, X_n \in \mathit{Input}(\Phi) \tag{2}$$

This model includes n sub-problems, each of which is a minimization problem, and all objective functions have the same domain.

V. Genetic Algorithm Based on Local Evolution

In this section we propose a GA with local evolution to solve problem (2). In our algorithm, the individuals are not evolved consistently, but divided into different groups according to their objective functions. In addition, we will use the fine-grained evaluation function to select individuals in the evolutionary phase.

1. Initialization of population

In our algorithm, an individual is an input vector of the program under test, and several individuals compose a population. Suppose that the population size is s, then the initial population can be generated by randomly sampling s test inputs, denoted as $X_1, X_2, \cdots, X_s$.

2. Grouping individuals according to objective functions

For individual X_i $(i = 1, \cdots, n)$, we can obtained its objective functions for all target paths by only running the program once. Let $F(X_i) = (f_1(X_i), f_2(X_i), \cdots, f_n(X_i))$. X_i may has good value for several objective functions while has bad value for the other ones. If individuals in the whole population cross, we cannot ensure their children also have good performance. So we will first divided the individuals into several groups according to the objective functions, and then the individuals in different groups will evolve independently.

For a given parameter t $(\lfloor \frac{s}{n} \rfloor \leq t < s)$, we first sort the individuals according to the first objective function, *i.e.* $f_1(X)$, and select t best individuals into the first group, denoted as G_1; Then sort the individuals according to the second objective function, *i.e.* $f_2(X)$, and select t best individuals into the second group, denoted as G_2; and continue this process until we obtain group G_n for the n-th objective function $f_n(X)$.

Denote that an individual may exist in multiple groups while several individuals may be not in any group.

3. Evolution operators

We will implement crossover, mutation and selection operators to the individuals in the same group.

1) Crossover operator

We use single-point crossover to generate children test data. Suppose that $X_i = (x_{i1}, x_{i2}, \cdots, x_{im})$ and $Y_i = (y_{i1}, y_{i2}, \cdots, y_{im})$ are two selected individuals for crossover in group G_i. A single crossover point is selected randomly, denoted as j $(1 \le j < m)$. All data beyond the j-th one are swapped between X and Y. The children are $X' = \{x_{i1}, \cdots, x_{ij}, y_{i,j+1}, \cdots, y_{im}\}$ and $Y' = \{y_{i1}, \cdots, y_{ij}, x_{i,j+1}, \cdots, y_{im}\}$, respectively.

2) Mutation operator

We use single-point mutation operator to amend the selected individual further. Suppose that $X_i = (x_{i1}, x_{i2}, \cdots, x_{im})$ is an individual. A single crossover point is selected randomly, denoted as j $(1 \le j \le m)$. If the j-th element of X is 0, then it will be changed to 1; if the j-th element is 1, then it will be changed to 0.

3) Selection operator

In order to select individuals more accurately, we will use the fine-grained evaluation function as the fitness function. Suppose that the parent individuals in group G_i are $\{X_{i1}, \cdots, X_{it}\}$, and their children are $\{X'_{i1}, \cdots, X'_{it}\}$. Because group G_i is faced to target path P_i, we just need to calculate the value of $g_i(X)$ for each individual in this group. Then we select t best individuals as the candidate ones.

For each group, we have t candidate individuals, and thus we have nt candidates, denoted as $\{X_{ij} | i = 1, \cdots, n, j = 1, \cdots, t\}$. We will select m individuals from all these candidates by roulette selection method according their fitness functions. In order to balance the target paths, we will normalize the fitness function in each group using z-score method. Let

$$\text{fitness}(X_{ij}) = \frac{g_i(X_{ij}) - \mu_i}{S_i} \tag{3}$$

where

$$\mu_i = \frac{\sum_{j=1}^{t} f_i(X_{ij})}{t}, S_i = \sqrt{\frac{1}{t-1}\sum_{j=1}^{t}(f_i(X_{ij}) - \mu_i)^2}$$

4. Reconstruction of sub-optimization problems

We investigate the i-th sub-optimization problem in Eq.(2). If there is an individual X satisfying $f_i(X) = 0$, then X is a test datum covering path P_i. So the problem of covering path P_i is solved. It is natural to save it and remove the corresponding objective function from Eq.(2). By doing this, the amount of objective functions becomes less and less till all target paths have been covered. Without loss of generality, assuming that the test data of the first k paths have been found, the optimization problem of test data generation will be simplified as follows:

$$\begin{cases} \min f_{k+1}(X_1) \\ \cdots \\ \min f_n(X_n) \end{cases} \tag{4}$$
$$\text{s.t. } X_{k+1}, \cdots, X_n \in Input(\Phi)$$

5. Steps of the algorithm

Based on the above discussion, the main steps of the proposed algorithm are as follows:

Step 1 Set the values of parameters, such as the number of target paths n, population size s, maximum termination generation T, crossover probability P_c, and mutation probability P_m. Population initialization, and let $g = 0$.

Step 2 For each individual X, calculate the values of objective functions:

$$F(X) = (f_1(X), f_2(X), \cdots, f_n(X))$$

Step 3 If there is a $f_i(X) = 0$, which means that X covers path P_i, then X is an optimal solution of the i-th optimization problem. In this case, save X, delete P_i from the target paths, and reconstruct the optimization problem by deleting the i-th sub-optimization problem, and let $n = n - 1$.

Step 4 If the number of target paths n becomes 0, or the number of generations g is larger than T, terminate the evolution of the algorithm, and output the test data; otherwise, go to Step 5.

Step 5 Divide the individuals into n groups, and implement genetic operations on each group to generate offspring population. let $g = g + 1$ and go to Step 3.

VI. Experiments

A group of experiments are conducted so as to investigate the performance of the proposed method. First, subjects are introduced. Afterwards, experimental design is characterized. Finally, empirical results are presented and discussed.

1. Subjects

In order to evaluate the proposed method, we select twelve programs for experiments. Table 2 shows some basic information of each program, including its name, size, and simple description. Table 2 is sorted by the sizes of the programs. These subjects include not only laboratory programs, but also nontrivial industry ones. In addition, their lengths and functions are different from each other. These programs have been used very thoroughly by other researches in the literature of software testing and analysis [9,21,22].

Note that it is a very difficult problem to generate test data for full path coverage. In addition, the feasibility of paths are an undecidable problem[21] that we will not discuss here. So we randomly choose several feasible paths to cover for each program under test. The number of target paths for each program is also listed in Table 2.

2. Experiment setting

In order to validate the performance of the proposed method in this study (for short, our method), we compare it with other two methods, namely Ahmed's method and the random method. The reason why we choose Ahmed's method for comparison is that it is also for multiple-path coverage. And the random method is a basic one and widely used. All methods (except random one) apply the same values of parameters, which are listed in Table 3.

Table 2. Subjects

Program	Loc	Description	No. of paths
Insertion sort	16	Array sorting	4
Bubble sort	18	Array sorting	4
Triangle	20	Return the type of a triangle	4
Binary search	37	Key number searching	7
Look	135	Find words in the system dictionary or lines	30
Cal	160	Print a calendar for a specified year or month	30
Controller	172	Internal states regulation	30
Tcas	173	Altitude separation	30
Tot_info	365	Statistics computation	50
Schedule2	374	Priority scheduler	50
Printtok	400	Lexical analyzer	50
Schedule	412	Priority scheduler	50

Table 3. Parameter settings

Parameter	Value
Population size	300
Selection operator	Roulette wheel
Crossover operator	One-point crossover
Crossover rate	0.6
Mutation operator	One-point mutation
Mutation rate	0.1
Maximum generation	50000
Encoding style	Binary
Variable range	[0, 1023]

3. Experimental results

We repeat the experiment 30 times for each program under test and each method in the same conditions, and record the time consumption of each run and each method. The experiment results are listed in Table 4, in which Ave. and S.D. are the sample average and standard deviation of time consumption for each program and method, respectively.

We can see from Table 4 that, 1) For all programs under test, the results of our method is the best; 2) The least time cost of our method is 5.62s for Bubble sort program, while the time cost of Ahmed's method and the random method for the same program are 9.27s and 12.32s, respectively; 3) The greater time cost of our method is 84.27s for Tot_info program, while the time cost of Ahmed's method and the random method for the same program are 125.83s and 147.34s, respectively.

In order to present more scientific analysis for the experiment results, we use T-test method to do hypothesis testing with Spass software. Suppose that T_1 and T_2 are the statistics for the comparison of our method and Ahmed's method, and our method and the random method, respectively. The results for different programs are listed in the last two columns of Table 4.

From the results we can see that the values of T_1 and T_2 are all less than $-Z_\alpha = -2.325$. Then we can conclude that the time consumption of our method is significantly less than that of Ahmed's method and the random method.

VII. Conclusions and Discussion

Many problems of test data generation can be transformed into a path coverage problem in software engineering. It is an promising direction to solve the problem of path coverage with GAs. But the efficiency of test data generation for multi-path coverage needs to be further improved.

This paper proposes a multi-path generation method based on a GA with local evolution. First, we use the coarse-grained evaluation function to establish the mathematical model for the problem of covering multiple target paths. Then, we propose a GA with local evolution to solve the optimization problem. The individuals are evolved in different groups that are divided according to the objective functions. In the selection phase, individuals are selected according to a fine-grained fitness functions.

The experiment results show that our method needs less time consumption compared with Ahmed's method and the random method.

One possible threat to the validity of the proposed method may be parameter settings. Appropriate choice of parameters can improve the performance of an algorithm and therefor enhance its efficiency in generating test data. The second threat to the validity maybe have relation with the difficulty of the selected target paths.

Table 4. Experiment results

Programs	Our method		Ahmed's method		Random method		T_1	T_2
	Ave.(s)	S.D.	Ave.(s)	S.D.	Ave.(s)	S.D.		
Insertion sort	17.26	6.24	26.39	8.54	38.90	11.23	–4.73	–9.23
Bubble sort	5.62	4.96	9.27	5.63	12.32	4.17	–2.66	–5.66
Triangle	13.58	5.29	20.82	7.73	28.73	8.72	–4.23	–8.14
Binary search	36.04	10.82	54.84	10.93	59.61	13.36	–6.70	–7.51
Look	18.02	7.88	25.72	9.79	31.82	10.9	–3.36	–5.62
Cal	35.92	10.02	60.52	13.07	82.73	15.33	–8.18	–14.00
Controller	49.93	12.78	71.81	13.83	90.26	15.53	–6.36	–10.98
Tcas	35.92	9.95	52.89	11.44	77.28	19.53	–6.13	–10.34
Tot_info	84.27	21.89	125.83	29.78	147.34	32.25	–6.16	–8.86
Schedule2	78.83	22.51	110.73	35.75	135.37	46.31	–4.14	–6.01
Printtok	30.64	16.93	52.34	20.42	63.44	29.54	–4.48	–5.28
Schedule	44.93	15.63	63.54	18.92	83.43	24.34	–4.15	–7.29

References

[1] B. Beizer, *Software Testing Techniques*, International Thomson Computer Press, 1990.

[2] L.A. Clarke, "A system to generate test data and symbolically execute programs", *IEEE Transactions on Software Engineering*, Vol.2, No.3, pp.215–222, 1976.

[3] W. Miller and D. Spooner, "Automatic generation of floating point test data", *IEEE Transactions on Software Engineering*, Vol.2, No.3, pp.223–226, 1976.

[4] J.H. Holland, *Adaptation in Nature and Artificial System*, Michigan: The university of Michigan press, 1975.

[5] S. Xanthakis, C. Ellis, C. Skourlas, *et al.*, "Application of genetic algorithms to software testing", *Proceedings of 5th International Conference on Software Engineering and Its Applications*, Toulouse, French, pp.625–636, 1992.

[6] X. Yao and D. Gong, "Testability transformation based on dominant relation of target statements", *Acta Electronica Sinica*, Vol.41, No.12, pp.2523–2528, 2013. (in Chinese)

[7] J. Miller, M. Reformat and 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.

[8] A. Baars, M. Harman, Y. Hassoun, *et al.*, "Symbolic search-based testing", *Proceedings of IEEE/ACM International Conference on Automated Software Engineering*, Lawrence, USA, pp.53–62, 2011.

[9] C. Michael, G. McGraw and M. Schatz, "Generating software test data by evolution", *IEEE Transactions on Software Engineering*, Vol.27, No.12, pp.1085–1110, 2001.

[10] O. Bühler and J. Wegener, "Evolutionary functional testing", *Computers and Operational Research*, Vol.35, No.10, pp.3144–3160, 2008.

[11] A. Watkins, E. Hufnagel, D. Berndt and L. Johnson, "Using genetic algorithms and decision tree induction to classify software failures", *International Journal of Software Engineering and Knowledge Engineering*, Vol.16, No.2, pp.269–291, 2006.

[12] I. Hermadi and M.A. Ahmed, "Genetic algorithm based test data generator", *Proceddings of Congress on Evolutionary Computation*, Canberra, Australia, Vol.1, pp.85–91, 2003.

[13] Y. Zhang and D. Gong, "Evolutionary generation of test data for path coverage based on automatic reduction of search space", *Acta Electronica Sinica*, Vol.40, No.5, pp.1011–1016, 2012. (in Chinese)

[14] P.M.S. Bueno and M. Jino, "Automatic test data generation for program paths using genetic algorithms", *International Journal of Software Engineering and Knowledge Engineering*, Vol.12, No.6, pp.691–709, 2002.

[15] A. Watkins and E. Hufnagel, "Evolutionary test data generation: a comparison of fitness functions", *Software Practice and Experience*, Vol.36, No.1, pp.95–116, 2006.

[16] I. Hermadi, C. Lokan and R. Sarker, "Genetic algorithm based path testing: Challenges and key parameters", *Proceedings of Second World Congress on Software Engineering*, Wuhan, China, pp.241–244, 2010.

[17] J. Wegener, A. Baresel and H. Sthamer, "Evolutionary test environment for automatic structural testing", *Information and Software Technology*, Vol.43, No.14, pp.841–854, 2001.

[18] M. Ahmed and I. Hermadi, "GA-based multiple paths test data generator", *Computer and Operations Research*, Vol.35, No.10, pp.3107–3127, 2008.

[19] D. Gong and Y. Zhang, "Novel evolutionary generation approach to test data for multiple paths coverage", *Acta Electronica Sinica*, Vol.38, No.6, pp.1299–1304, 2010. (in Chinese)

[20] B. Korel, "Automated software test data generation", *IEEE Transactions on Software Engineering*, Vol.16, No.8, pp.870–879, 1990.

[21] P.M.S. Bueno and M. Jino, "Identification of potentially infeasible program paths by monitoring the search for test data", *Proceedings of IEEE International Conference on Automated Software Engineering*, Grenoble, Frence, pp.209–218, 2000.

[22] J. Wegener, K. Buhr and H. Pohlheim, "Automatic test data generation for structural testing of embedded software systems by evolutionary testing", *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, USA, pp.1233–1240, 2002.

YAO Xiangjuan was born in Hebei. She received the Ph.D. degree in control theory and control engineering from China University of Mining and Technology, Xuzhou, Jiangsu, China, in 2011. She is an associate professor in College of Science, China University of Mining and Technology. Her main research interest is search based software testing. (Email: yxjcumt@126.com)

GONG Dunwei was born in Jiangsu. He received the Ph.D. degree in control theory and control engineering from China University of Mining and Technology, Xuzhou, Jiangsu, 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 and intelligence optimization. (Email: dwgong@vip.163.com)

WANG Wenliang was born in Shandong. She received the B.E. degree in College of Science, Linyi University, Linyi, Shandong, China, in 2012. Since 2012, she has been a postgraduate student in College of Science, China University of Mining and Technology. Her research interest is search-based software engineering. (Email: 974931719@qq.com)