Journal of Theoretical and Applied Information Technology
10th February 2013. Vol. 48 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTOMATIC TEST CASE GENERATION BASED ON GENETIC ALGORITHM

DAN LIU, XUEJUN WANG, JIANMIN WANG
School of Information Science and Technology,Shijiazhuang Tiedao University,
Shijiazhuang050043, China
E-mail: liudanld@126.com

ABSTRACT

For the problem that Genetic Algorithm （GA） suffers from large iteration times and low efficiency in test case generation, this paper proposes a Modified Genetic Algorithm （MGA）. The algorithm adopts real number coding and the principles of logic coverage, while the fitness function is to be improved. In addition, it adds genetic-oriented control. The algorithm is conducive to population diversity and avoids premature convergence phenomenon. Experimental result shows that MGA has faster convergence speed and higher test data generation efficiency compared with traditional genetic algorithm.

Keywords: *Genetic Algorithm, Test Case Generation, Logical Coverage*

1. INTRODUCTION

Software testing is an important means to guarantee the quality of software. In current software development process, the cost of testing covers a half of the total cost of software development. Test case is a commonly used tool to realize the effectiveness of software testing, and also a key to guarantee such effectiveness. The manual test case building of testers not only requires heavy workload and long testing period, but also can trigger testing omissions easily[2]. Therefore, automatic generation of test case has always been studied in an extensive range. Automatic generation of test case can reduce the cost of software development, substantially enhance software reliability and shorten the period of software development.

Efficient methods of test case generation can produce highly quailed test cases in a small number, thus reducing the cost of software development. Therefore, studying an effective method of test case generation has its practical significance[1]. Automatic generation of test case refers to the process of seeking a group of test input data that can meet given testing standards within a data domain, and that's why there are thoughts about transforming the generation of test case to path search in recent years. Under general situation, the undecidability of test case generation, as well as the scale and complexity of programs being tested greatly restricts general search algorithm. As a result, genetic algorithm has been applied to the generation of test case. In practical application, due to its own defects of genetic algorithm, premature convergence is a phenomenon in genetic algorithm that cannot be neglected, which is mainly reflected in that all individuals in the group are stuck in the same extreme value while those stopping evolution are the most approximate to optimal solution are always eliminated, namely, misconvergence occurs in the evolutionary process. In addition, if such questions as parameter coding scheme, the selection and control of operator, adjustment strategies of fit value, together with the control of group size and coverage of initial test case cannot be solved evenly, it is quite difficult for genetic algorithm to realize ideal effect.

In order to avoid premature convergence and generate efficient and ideal test case, this dissertation aims at improving genetic algorithm by realizing branch coverage condition in programs through programs being tested, dynamically realizing path set of programs, and improving fitness evaluation function. Then this dissertation adds genetic orientation control, which can work as a guide in selecting genetic operation, and enable it to inherit toward multiple-path coverage direction.

2. TECHNIQUES OF GENETIC ALGORITHM IN SOFTWARE TEST CASE GENERATION

Biological evolution is a wonderful optimization procedure, which generates fine species that can be adapted to environmental changes through such rules as selective elimination, variation and genetic endowment.

Genetic algorithm (Genetic Algorithm) is a global optimization algorithm inspired by the thoughts of biological evolution. The main feature of genetic algorithm lies in that it operates structure target directly without the restriction of derivation and function continuity, and that it has internal implicit parallelism and better capability in global optimization[3][5]. By adopting randomized optimizing method, optimized searching space can be acquired and guided automatically, and searching directions can be adjusted of its own accord without determining any rules. It is, in essence, a direct searching method that does not rely on specific questions. With strong robustness of the category of questions, this method has been extensively applied in many disciplines, such as function optimization, production dispatching, automatic control, image processing, pattern recognition, artificial life and machine learning.

The operation target of genetic algorithm is chromosome, which consists of a string of numbers. In this numerical string, each number is called gene. A combination of a series of chromosomes constitutes a population. Each chromosome has a fitness, which is used to determine the possibility of the survival of its next generation. After the next generation appears, a part of chromosomes cross while a smaller part witness genetic variation.

By coding parameters of programs being tested, which play the role as chromosome, we can introduce genetic algorithm to software test case generation field. The size of population decides the number of chromosomes, with each chromosome has its fitness[4]. In the genetic process, fitness of groups need to be evaluated, test data can be achieved by decoding populations that meet the termination conditions of heredity, and if there they fail to meet such termination conditions, populations require genetic operation (selection, cross and variation), so as to evolve to more optimized ones, thus offering more efficient test cases for us. The mathematical formula applying genetic algorithm to the generation of test case is shown below:

$$GATD = (C, E, P_0, M, S, A, V, T, PT)$$

C presents coding method, E presents fitness function, *M* presents the population size, *S* presents the selection operator, *A* presents a crossover operator, P_0 presents the mutation operator, *T* presents the termination condition, *PT* is the target path required to cover.

3. IMPROVED ALGORITHM

Studies on the introduction of tradition genetic algorithm in automatic generation of test case have important influence on the automation of software testing, the efficiency and quality of case generation is not quite satisfying though. This dissertation combines Delaunay triangulation network in program generation and improves genetic algorithm, so that it can inherit toward a more oriented and ideal direction in case generation in a shorter period of time, thus enhancing the efficiency and quality of case generation[9].

3.1 Coding Scheme

The input parameters of Delaunay triangulation network generation program being tested are uncertain two-dimensional unorganized points. Different from common programs, this program has relatively high requirements in accuracy and a wide indicating range. Therefore, commonly used binary coding method cannot be adopted here in coding. Instead, real-number coding is suggested here, so that it is more convenient to realize genetic searching in a larger space, improve computing complexity of genetic algorithm and enhancing mathematical operation efficiency[10]. Real-number coding uses true values of target variable, so it is also called truth-value coding method. For example, each individual of the program being tested has fifteen points, which can be presented with real-number coding as follows:

$$\begin{bmatrix} 20.356125 & 125.122345 \\ 350.251345 & 213.365421 \\ .... & ... \\ 43.251456 & 167.854294 \end{bmatrix}$$

3.2 Fitness Function

As an important evaluation function indicating advantages and disadvantages of populations, fitness function is a connector with practical questions[6]. As to branch coverage or path coverage, fitness value F_i of individuals in population can be considered as the ratio of a branch *f* (or path) of practical coverage of test case to total branches *z* (or paths) in program being tested, $F_i = f / z$, it indicates individual branch coverage rate. However, in this way, fitness function is merely shown by numerical values, as to which branches are covered and which are not, as well as the influence individual fitness has on that of the whole population are still unclear. That's why a visible population fitness condition is required, so that it can be shown by matrix. Take branches of program being tested as the column and individuals in population as the row. For example, there are *n* branches in program being tested, $(p_1, p_2, p_3, ...p_n)$ and individuals in population are $(1, 2, 3, ...m)$, then we can achieve a coverage matrix: $\begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \cdots & & & \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix}$,

When individual *i* passes branch *j*, v_{ij} item in *D* is 1, otherwise 0. In this way, the coverage of each individual branch can be clearly seen. The bigger the number of 1 in each row gets, the bigger fitness this individual shows. When each column in the matrix has one or more 1, it means all branches are covered, and this population is an ideal one. We can then achieve population fitness: $Pf = sm1/z$, in this formula *z* is the total number of branches in program being tested, and *sm*1 is the number of columns that can meet $\sum_{i=1}^{m} v_{ij} \geq 1 (j = 1, 2, ..., n)$ in matrix D.

3.3 Path Coverage

At present, there are mainly two ways to generate test data in accordance with paths: (1) black-box testing; (2) white-box testing. This dissertation adopts the second method, and combines improved genetic algorithm to generate test case.

It is found out in the study process that the combination of all branches of program being tested contains all paths in the project. For example, there's a program flow, as you can see in Figure 1. There are four branches b, c, e and f, the starting *s* and ending *o* should also be classified into branch, but s and *o* do not combine with other branches. After combination, we have

$$\{sbeo, sbfo, sceo, scfo, sbcefo, ...\}$$

There are four paths in total:

$$sbeo, sbfo, sceo, scfo$$

It can be apparently seen that the combination of branches contain these four paths in the program. However, when evaluating path coverage, we cannot regard all combinations of branches as the target paths. If there are many branches, the paths of such combinations will be far more than the real number of target paths. In order to omit excessive branch combinations and at the same time, guarantee its integrity, path collection should guarantee to be dynamic. In 3.2, "1" with corresponding branch in each row of the branch coverage matrix *D* represents covered branch. Therefore, by recording all covered branches in the row, we can get a path, with which we can initialize path set *PT*. In testing process, if passed path cannot be found in path set *PT*, add this path to set PT. For example, when testing program of a population, we can get its branch coverage condition: $v = [1011011001]$, and such row does not exist in the original *D*, so we convert this path to $path0 = \{0, 2, 3, 5, 6, 9\}$ and add it to set *PT*.

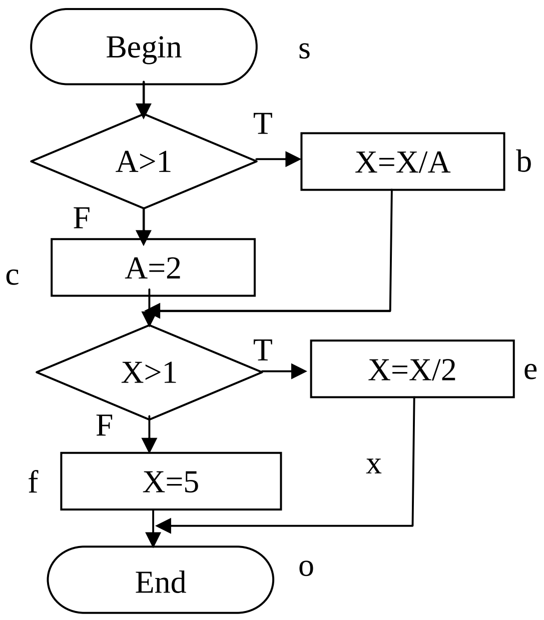

Figure 1. The Program Flow

3.4 Genetic Orientation Control

In the link of genetic operation, if it is proceeded merely with given probability, the fine population after genetic operation cannot be guaranteed. In other words, the whole link has its probability and randomness[8]. In order to accelerate the efficiency of generating satisfying populations, genetic operation should be controlled.

Take branch coverage as an example, branch coverage condition of individual i is $v_i = [1001110111]$, And that of another individual *l* is $v_l = [1000010011]$.

It can thus be seen that branch set l passes is a subset of what i passes, the coverage of v_l is incomplete when comparing with that of v_i, and its debugging capability is lower than v_i. Such individuals can receive variation or other genetic operation in accordance with specific conditions and preset parameter values.

It is assumed that the pre-set value copying to new population by selection is *se* and the crossed value is *cr*. It is found out in practical questions, due to the fact that individuals of program being tested are multi-dimensional data, and the size of individuals is very big, but the probability of attaining new or fine individuals through crossing is not always big[7]. Therefore, we can enhance the probability of variation, reinforce random search ability of algorithm, and at the same time, adjust crossover probability, selective probability and the size of population. In this way, population treatment can better guide genetic direction, and avoid population premature to certain extent. Improved algorithm flow can be seen in Figure 2.

As to path coverage, its principles and thoughts are similar to branch coverage. Fitness function, which is used to evaluate advantages and disadvantages of individuals, needs to take branch coverage of this path into consideration, namely, the ratio of the number of branches to the total number of branches, as well as whether this path is also covered by other individuals in the population. The evaluation of population fitness should rely on the overall path coverage condition of the population.

This way of processing population will guarantee its diversity. At the same time, as path coverage and branch coverage combines to function, it is a further guarantee of debugging capability of generated case.

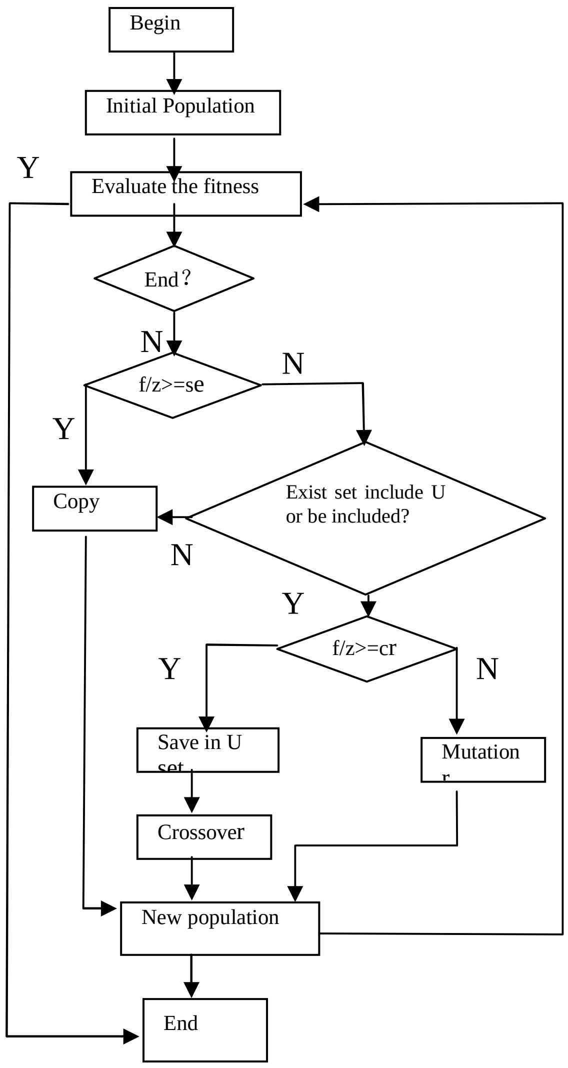

Figure 2. The Program Flow

4. EXPERIMENT RESULT

Take the program generated by Delaunay triangulation network as an example. Firstly, realize automatic plug-in of program being tested. In other words, plug in probe to each branch sentence. When inputting data and the operating program being tested, the probes can record branch coverage

Table 1. Parameter Settings Without Improved Algorithm

Population Size	15
Individual points	150
Maximum Generation	30
crossover probability	0.8
Mutation probability	0.4
Point Range	1~300

condition. Plug-in sentences should be guaranteed to be free of the influence of program itself. Input parameters and begin to generate cases. Parameter settings without improved algorithm can be seen in Table 1. Parameter settings with improved algorithm can be seen in Table 2.

Table 2. Parameter Settings With Improved Algorithm

Population Size	15
Individual points	150
Maximum Generation	30
crossover probability	0.8
Mutation probability	0.4
Point Range	1~500

The result of experiment adopting branch coverage can be seen in Table3.

Table 3. The Result Of Experiment (Branch Coverage)

Algorithm	Without Improvement	With Improvement
Parameter	Table 1	Table 2
Total Number of Branches/branch	34	34
Total Number of Coverage/branch	26	29
Time /s	4.5	6.8
Coverage Rate/%	76.7	84.9

Adopt path coverage and set parameters of improved algorithm to parameter data shown in Table 1, and the experiment result can be seen in Table 4.

Table 4. The Result Of Experiment (Path Coverage)

Algorithm	Without Improvement	With Improvement
Parameter	Table 1	Table 1
Total Number of Coverage/branch	26	29
Time /s	4.5	6.8

It is shown by the experiment result that improved genetic algorithm improves branch coverage rate and path coverage rate, but takes a long time in heredity. How to better improve the algorithm, so that it can cover program branches to a larger extent, and shorten the time heredity costs still requires further studies.

5. CONCLUSION

Branch coverage to a larger extent or automatic generation of path test case is a critical technology to software test automation. The writer of this dissertation puts forward improved algorithm, which improves branch coverage rate and path coverage path, which have been demonstrated through experiment. However, such improved algorithm is not quite ideal in time efficiency and the generation of special test case. Therefore, further studies are also needed in this field.

6. ACKNOWLEDGEMENT

The authors are extremely grateful to the anonymous reviewers for their constructive and valuable comments, which have contributed a lot to the improved presentation of this paper. The work was partially supported by scientific research project of Hebei education department(no.Z2012151).

REFRENCES:

[1] Sthamer H H, Wegener J, Baresel A, 2002 “Using evolutionary testing to improve efficiency and quality in software testing”, *Proceedings of the second Asia Pacific conference on software testing analysis and review*，pp 22-24.

[2] Moataz A, Ahmed, Irman Hermadi, 2007 “GA-basedmultiple paths test data generator”, *Computer Operation Research*, pp 3110-3311.

[3] Amit Paradkar, Kuo Chung Tai, “Test Generation for Boolean Expressions”, *In Proceedings of International Symposium On Software Reliability Engineering*, pp. 106-115, 1995.

[4] Jin Cheng Lin, Pu Lin Yeh, 2001“Automatic Test Data Generation for Path Testing Using GAs”, *An International Journal*, pp. 47-64.

[5] D.E.Goldberg, 1989“Genetic Algorithm in Search”, *Opimization and Machine Learning Addsion Wesley*.

[6] D.E.Goldberg, K.Deb, 1991“A Comparative Analysis of Selection Schemes Used in Genetic Algorithms”, *Foundations of Genetic Algorithms*, pp. 69-93.

[7] M. Srinivas, L. M. Patnaik, 1994 “Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms”, *IEEE Trans. On System, Man and Cybernetics*, pp. 656-667.

[8] Kennedy J, Eberhart R.C, 1995"Particle Swarm Optimization",*International Conf on Neural Networks,IV*, *Perth: IEEE Press*, pp. 1942-1948.

[9] Shi Y, Eberhart R.C, 1998 "A Modified Particle Swarm Optimizer", *International Conference on Evolutionary Computation, NJ: IEEE Press*, pp. 69-73.

[10] Van den Berg F., Engelbrecht A, 2002"A new locally Convergent Particle Swarm Optimizer", *International Conference on System, Man and Cybernetics*.