The Species per Path Approach to GEMGA-Based Test Data Generation

Zhenghua Xin
School of Information Engineering, SUZHOU University, Suzhou, China, 18955711392, 0086
begin0000@qq.com

Liangyi Hu
School of Economics and Management, SUZHOU University, Suzhou, China, 18955711391, 0086
huliangyi@126.com

Na Li
School of Computer Science and Engineering, Handan University, Handan, China 0086

***Abstract*—This paper discusses the use of Species per Path approach[1] and gene expression messy genetic algorithm (GEMGA) for automatic software test data generation. This research finds another path problem and extends Species per Path approach on dynamic test data generation. In increasing the search space by program transformation for the path potentially suffering from the path problem, this research differs from previous Species per Path. Transforming the program under test can factor out and increase several paths to reach the same target. As a result, each species uses a fitness function tailored for the space for the path. All together the effort of the fitness functions can guide the search to reach the target. The function is minimized by using GEMGA. The work describes the implementation of Species per Path Approach to GEMGA-based approach and examines the effectiveness on a TRITYP program. Compared with other approaches, the experimental results show that it can generate higher quality test data more efficiently, and should be applied to larger applications.**

Keywords—Species per Path approach (SpP); gene expression messy genetic algorithm (GEMGA); testability transformation; blackbox search (BBO)

I. INTRODUCTION

Efficient and Effective test data generation is one of the major laborious and time-consuming tasks within the software process. A number of metaheuristic techniques have been developed to it. Test data generation is generally categorized into two classes: static [2] [3] [4] and dynamic. By far the most common dynamic test data generation in the literature is structural test data generation [5] [6] [7]. Structural testing criteria help testers in the generation and evaluation of a test case set T.

Once a test adequacy criterion has been selected, the question that arises next is how one should go about creating a set which meets that criterion. Since this can be difficult to do by hand, there is an obvious need for automatic test data generation. Many articles consider the problem of dynamic test data generation is reduced to one of the function minimization [5] [8]. The paper also treats parts of the program as functions that can be evaluated by executing the program, and whose value is minimal for those inputs that meet the adequacy criterion. There has simulated annealing, genetic algorisms and hill climbing for structural test data generation [9] [10] [11] [12].

This paper uses the extension of Species per Path approach which transforms the program under test into a version in which multiple paths to the search for the target are factored out. The primary contributions of the paper are:

- extend the Species per Path.
- apply GEMGA [13] [14] [15] to generate test data automatically.

II. BACKGROUND

This section presents an overview of the background to the SpP to GEMGA-based test data generation, including the test case set and the computation of the fitness function for the coverage of individual structural targets.

A test case set predicates to be satisfied to consider the testing activity ended and generally require the execution of a set (P) of paths, capable of exercising certain elements in the program under the test. Determining P is an important and hard job. This paper uses a control flow graph to generate a path set [1] [16].

The fitness function is made up of two components-the approach level and the branch distance. The approach level measures how close an input vector was to executing the current structure of interest, on the basis of how near the execution path was to reaching it in the program's control flow graph. Central to this metric is the notion of a critical branching node with an exit that, if taken, causes the target to be missed. The detailed work so far for structural test data generation has mainly addressed statement, branch or condition coverage. The basic form of the minimizing objective function is :

Approach level + m_branch_dist

The strategy in which approach level and m_branch_dist are computed varies according to the coverage type in question [10].

The computation of the fitness function for the coverage of individual structural targets is explained in references [1] [16].

This paper expresses m_branch_dist.

$$\text{m\_branch\_dist} = \frac{\text{BranchDistance}}{\max(\text{BranchDistance})}$$

Max (Branch Distance) is the maximum of the Branch Distance in the current population of the current generation.

The fitness function derived for a target is only concerned with the branching nodes which absolutely must be executed

This work was supported by Open Research of Suzhou University using Intelligence Information Processing Lab (No.2010YKF14), Master Foundation of Suzhou University (No. 2009YSS17) and Young Foundation of Anhui education office (No. 2011SQRL157).

978-1-61284-774-0/11/$26.00 ©2011 IEEE

in a certain way in order for the target to be reached in the control flow graph.

III. THE PATH PROBLEM

The path problem occurs when the input domain for the program is largely dominated by paths through the program which miss the target, potentially suffocating the fitness function and preventing the search from receiving adequate guidance to the required test data as explained in (a) of Figure 1. Let P be the set of feasible paths which are required by structural testing criteria. The path problem occurs when the total number of input vectors executing each path in P account for only a small proportion of overall input domain [1].

This paper finds that the path problem also occurs even if the total number of input vectors executing each path is not a small proportion of overall input domain, for example input vectors with a small probability occurrence.

In TRITYP, the number of three equal inputs is large in proportion to the number of three inputs. But there is little probability of generating three equal inputs through the path of equilateral triangle.

In the SpP approach, resources are guaranteed to feasible paths, and species allocated to these paths are equipped with a higher level of search guidance [1]. On the basis of the new discovered path problem, the SpP approach is extended in this paper which will be explained in the later section 5.

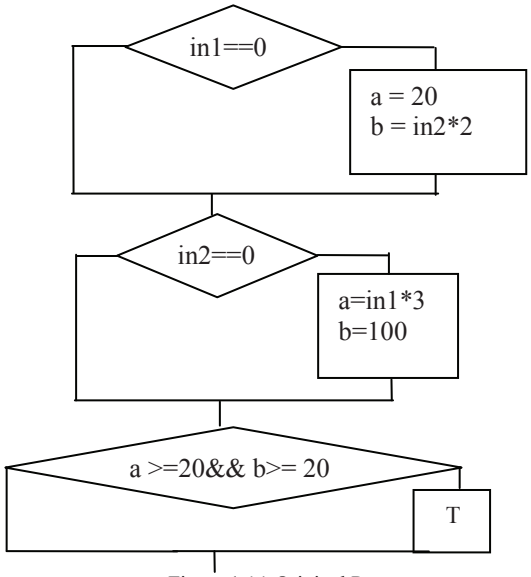

Figure 1 (a) Original Program

The longest or most difficult covering path may be easier to reach. Because the search space has much more constraints and also reduces significantly, the search space can be divided several species. It alleviates pressure of the search algorithm.

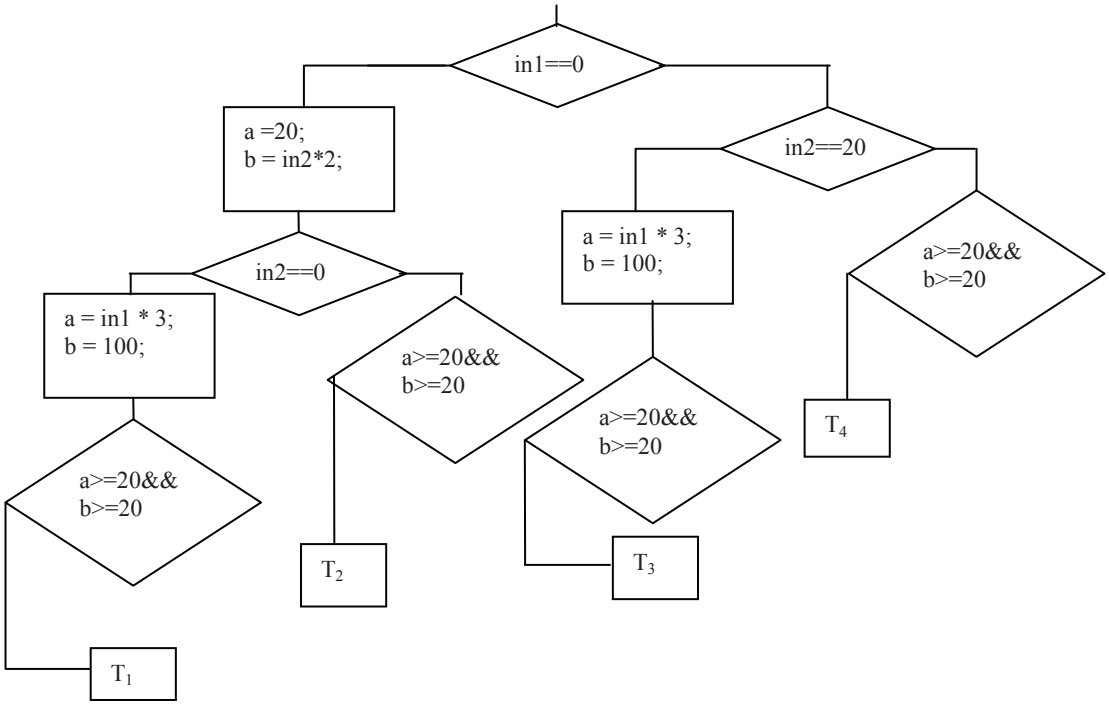

Figure 1 (b) Transformed Program

IV. GEMGA-BASED TEST DATA GENERATION

First, the blackbox search (BBO) concept is reviewed. Ingredients of black-box optimization are Problem size n, Search space S_n , A finite set {1, 2, 3… N} (where N depends on n). The set F_n of all possible "fitness functions" f from S_n

into the set {1, 2, 3… N}.

The fitness function f will be unknown to the algorithm and will act as the "blackbox". GEMGA-based test data generation begins with a random population of solutions (chromosomes). Its search space is the domain of the input vectors. Its fitness function is unknown until the execution of the program. GEMGA-based test data generation utilizes Messy GA's prominent feature that can optimize complicated problems without prior knowledge about schema arrangement in chromosomes, so that it can improve concurrency level of searching and test coverage. Blackbox optimization algorithms (BBO) such as genetic algorithms, simulated annealing, and their numerous cousins have found many applications in system identification, optimization, and prediction problems in this field. Design of GEMGA [17] is based on the alternate perspective of evolution, developed by SEARCH that emphasizes the computational role of gene Expression (DNA $\rightarrow$ RNA $\rightarrow$ Protein) in evolution.

Represents

Traditionally, genetic algorithms use a binary encoding of the solutions [18]. It is now accepted widely that binary encodings of the solution space are inferior compared to more natural solution space representations [19]. Using a natural encoding scheme can give significant performance improvements of removing the overheads of encoding and decoding [20]. This paper also uses real coded solutions.

GEMGA has four primary operators, namely: (1) transcription, (2) class selection, (3) string selection, and (4) recombination. Each of them is described in the following [10].The approach is illustrated in the Figure 2.

```
Step 1: Create CDGPaths
Step 2: Transform the program
Step 3: Create CurPopulation
        While (target not reached)
Step 4: Compute fitness values of CurPopulation
        using CDGPaths
Step 5: Transcription operator
Step 6: Selection operator
Step 7: Recombination operator
Step 8: Execute Program on each member of the
        Generated NewPopulation
        End while
```

Figure 2 GEMGA for automatic test data generation.

V. EXTENDED SpP TO GEMGA-BASED TEST DATA GENERATION

Because the fitness function for a test case set T combines the results of the computation along each path over the areas of the input domain for which those individual paths are executed. The SpP approach separates this information out into the landscape for each species, adding further information to guide the search to the execution of each individual path. Again, this is due to the extra fitness information which comes from the inclusion of extra branching nodes in the fitness function. Although the feasible landscapes produced by the SpP approach do contain areas of ridges and local optima, the basin of attraction to the global optimum is considerably larger.

This paper computes the fitness function for the coverage of the target in Figure 3 and Figure 4. Also, in additional to the fitness function of the Figure 1, Figure 3 and Figure 4 for reaching the same target, exchange the position of the expression a>=b, b<=c, c==d, we can get other five permutations. So eight fitness functions for the same target significantly increase the search spaces. In the TRITYP experiment, equilateral triangle is the hardest path to be covered. In this paper, we use three methods compute the fitness functions of the path. We illustrate the original problem as in Figure 5. After transforming the TRITYP program, the computation of the fitness function is illustrated in the Figure 6, Figure 7, and Figure 8 which exchange the position of the expression Side1==Side2, Side2==Side3, and Side1 == Side3. So the search space can be much larger than original space.

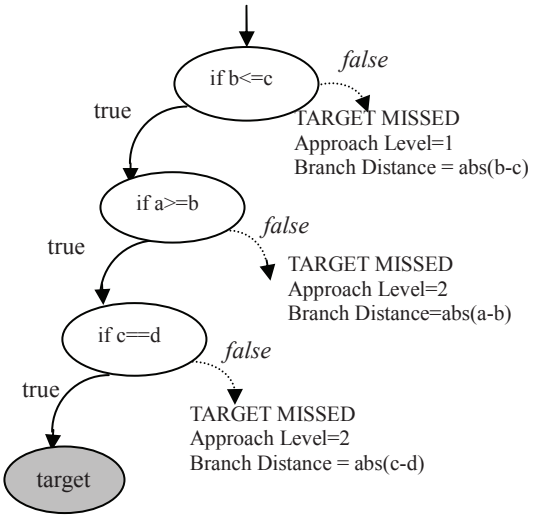

Figure 3 Calculating the fitness function for Evolutionary testing

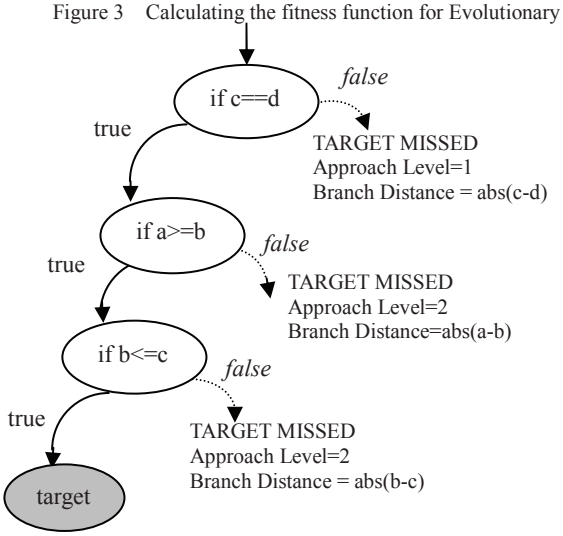

Figure 4 Calculating the fitness function for Evolutionary testing

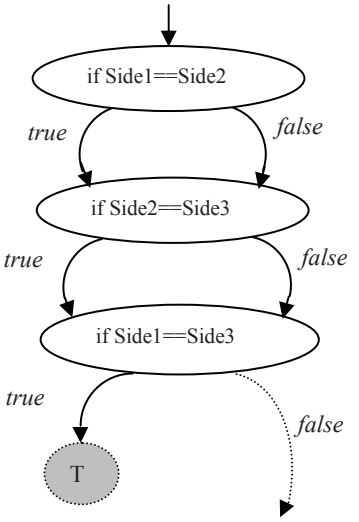

Figure 5 Calculating the fitness function for Evolutionary testing

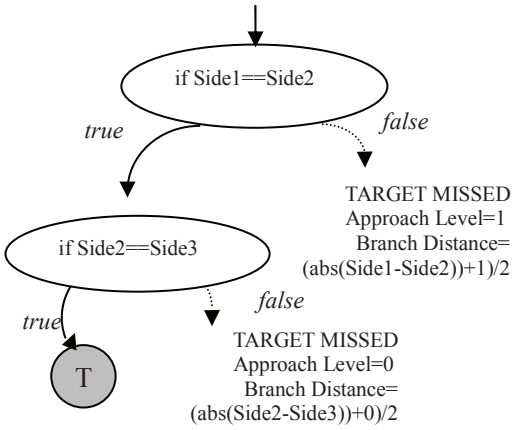

Figure 6 Calculating the fitness function for the path of equilateral triangle after transformed program.

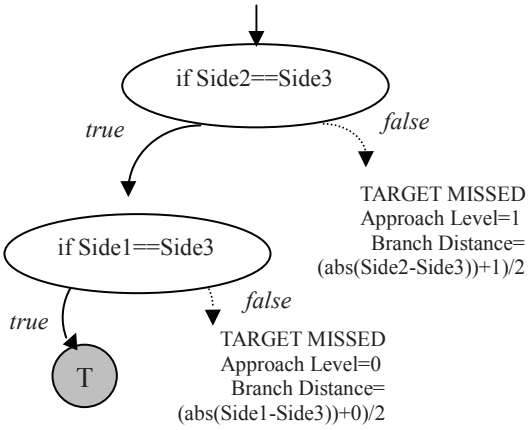

Figure 7 Calculating the fitness function for the path of equilateral triangle after transformed program.

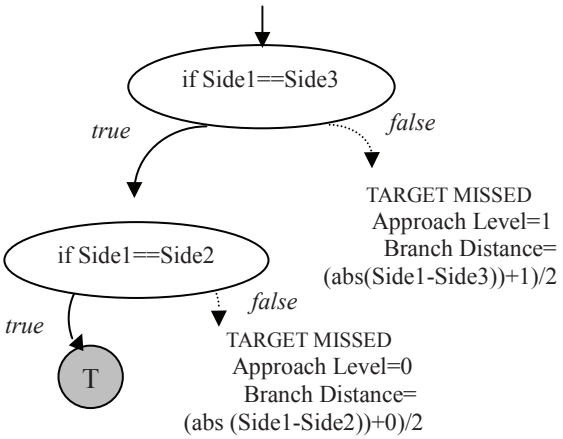

Figure 8 Calculating the fitness function for the path of equilateral triangle after transformed program.

VI. RESULTS

This paper uses a real encoding for the chromosome, and the simulated trial was implemented with matlab language. The real number was measured with the precision of 0.1 when the range is from 0 to 10000. When the range is from 0 to 100, the random real number was measured with the precision of 0.0001. Time was measured in seconds with the precision of 0.1 second.

Search results employed by other authors in the field of the Chaining approach for test data generation were used [21].

Table 1 different algorithms for TRITYP

Measures / algorithms	SR	C	ASuc	MSuc	AUnSuc	MUnSuc
Random		77%				
Path-Oriented	98%	100%	15	280	300	300
Goal-Oriented	26%	84%	0.1	1	0.1	1
Chaining(level 1)	61%	100%	0.6	2	1.1	3
Chaining(level 3)	99%	100%	1.3	17	5	11
SpP-GEMGA	100%	100%	3.5	8.4	0	0

Table2 different input domain for the SpP to GEMGA-based test data generation

Input Domain	0~100	0~1000	0~10000
time period	3.828000	4.375000	5.1710

Time was measured in seconds with the precision of 0.000001 second.

VII. CONCLUSIONS

This paper addresses the problem where previous SpP approach can be extended to have sufficient fitness information to find paths to execute the target. This paper uses the Species per Path and GEMGA to generate test data. The approach is shown to generate test data efficiently for test objects experiencing the path problem, which is identified in this paper as problem for GEMGA-based test data generation. The factoring out of paths by a testability transformation provides each species with more fitness information.

These claims are supported by a verification empirical study that compares the other approaches with the SpP to GEMGA-based approach.

The approach will be extended to larger and more complex system.

ACKNOWLEDGMENT

The author also acknowledges many useful discussions with my husband and Colleague Tan.

REFERENCES

[1] P. McMinn, M. Harman. Tonella The Species per Path Approach to Search-Based Software Test Data Generation. Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2006), Portland, ME, USA, 2006, 13-24.

[2] Silvia Regina Vergilio, José Carlos Maldonado, Constraint based structural testing criteria, Journal of Systems and Software, June 2006, 756 - 771

[3] Paulo Marcos Siqueira Bueno, Mario Jino:Identification of Potentially Infeasible Program Paths by Monitoring the Search for Test Data. Electronic Edition (IEEE Computer Society DL) BibTeX ASE 2000, 209-218

[4] Arnaud Gotlieb, Bernard Boella. Automatic Test Data Generation Using Constraint Solving Technique [J]. Software Engineering Note , 1998 , 23 (2): 53 - 62.

[5] Korel B. Automated software test data generation. IEEE Trans. on Software Engineering, 1990, 16(8):870-879.

[6] Michael, CC. Generating software test data by evolution. IEEE TSE,. 27(12): 1085-1110. 2001.

[7] Jones B, Sthamer H. Automatic structural testing using genetic algorithms[J]. Software Engineering Journal, 1996, 11(5) : 299-306.

[8] Ramanmoorthy CV, HoS BF, On the automated generation of program test data[J]. IEEE Trans. on Software Engineering, 1976, S E 2(4): 293-300.

[9] Pargas R, Harrold M. Test-data Generation Using Genetic Algorithms. Journal of Software Testing, Verification & Reliability, 1999, 9(4): 263-282

[10] Phil McMinn, Search-based Software Test Data Generation:A Survey. Software Testing Verification and Reliability, 2004//02, 105-156.

[11] Mark Harman, Phil McMinn: Theoretical & Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation. ISSTA 2007: 73-83

[12] Hermadi I, Ahmed M A. Genetic algorithm based on test data generator[C]. CEC'03,2003:85-91.

[13] Bandyopadhyay, S., Kargupta. Revisiting The GEMGA: Scalable Evolutionary Optimization Through linkage learning. In Proceedings of the IEEE International Conference on Evolutionary Computation, 603-608 IEEE Press (1997)

[14] H.Kargupta, B. Park.The Gene Expression Messy Genetic Algorithm For Financial Applications. Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering. IEEE Press, 2001. 155--161.

[15] Kargupta H. The gene expression messy genetic algorithm. In: Proc. of the Int'l Conf. on Evolutionary Computation. Nagoya: IEEE Computer Society Press, 1996. 814-819

[16] McMinn P., and Holcombe M. Evolutionary testing using an extended chaining approach. Evolutionary Computation 14 (1):41-64, 2006.

[17] H. Kargupta. Performance of the gene expression messy genetic algorithm on real test functions. In Proceedings of the IEEE International Conference on Evolutionary Computation, pages 631-636, 1996. IEEE Press.

[18] N. Tracey, A Search-Based Automated Test-Data Generation Framework for Safety-Critical System, Ph.D. Thesis, University of York, 2000

[19] Z.Michslewicz: Genetic Algorithms + Data Structures = Evolution Programs Springer- Verlog, 1996.

[20] Janikow C I, Michalewicz I. An Experimental Comparison of Binary and Floating Point Representation in Genetic Algorithms. In Proc 4th Int Conf Genetic Algorithms, 1991. 31～36

[21] Roger Ferguson, Bogdan Korel, The Chaining Approach for Software Test Data Generation Jan. 1996 ACM Transactions on Software Engineering and Methodology (TOSEM)