Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparison of BDBFF & ALBFF for Basis Path Testing Using GA

Deepak Garg*, Pallvi Garg
DCSA, KUK, Haryana,
India

Abstract*— Automatic path oriented test data generation is not only a crucial problem but also a hot issue in the research area of software testing today. In this paper genetic algorithm (GA) has been used as a robust metaheuristic search method under basis path testing coverage criteria. Two types of fitness function have been used, one is branch distance based fitness function (BDBFF) and other is approximation level based fitness function (ALBFF). Triangle classification program has been used to compare performances of these two fitness functions. Experimental results showed that BDBFF based approach can generate path oriented test data more effectively and efficiently than ALBFF based approach.

Keywords*— Approximation Level Based Fitness Function, Basis Path Testing, Branch Distance Based Fitness Function, Genetic Algorithm, Software Testing.

I. INTRODUCTION

Software testing is a process to identify the quality and reliability of software, which can be achieved through the help of proper test data. However, doing this manually is a difficult task due to large number of predicated nodes, loops, infeasible paths in the module. In general, manual software testing accounts for approximately half of the elapsed time and more than half of the total cost in software development [1, 2] & automated software testing is a promising way to cut down time and cost.

Test data generation in program testing, is the process of identifying a set of test data, which satisfies the given testing coverage criterion. Coverage criteria are used to measure how well the program is exercised by a test. Basis path testing is a coverage criterion of software testing that can detect almost two third of errors in the program under test [3] and various structural test data generation problems can be represented into a path oriented test data generation problem [4]. Furthermore, a basic path coverage criteria covers branch coverage and path coverage. So, basis path testing has been selected in this paper as coverage criteria for software testing.

Many automatic tools for test data generation are already present (for generation of test data), but they are not good for large scale problems as they require knowledge of solution space and are also not robust to dynamic environment. So, Genetic Algorithm has been used in this paper for generating test data set from a pool of randomly generated test data automatically [5, 6]. Furthermore related works [7, 8] indicate that genetic algorithms based test data generation outperforms other dynamic approaches and static approaches.

The rest of the paper has been organized as follows. In section 2, a brief introduction to flow of basis path testing using GA is given. In section 3, two most important fitness functions are given Approximation level distance based fitness function (ALBFF) and Branch distance based fitness function (BDBFF). In section 4, experimental settings and results based on a triangle classification program has been given. Finally, conclusions and direction for future work has been given in section 5.

II. FLOW OF BASIS PATH TESTING USING GA

A problem can be solved by Genetic algorithm if it is in the optimization form so here steps are represented to convert testing problem in to optimization problem [9]. Here the main role is of fitness function construction which converts this test data searching problem into optimization problem. Flow of Basis path testing using GA is represented by fig. 1.

Steps of Test Data Generation [9]

1. CFG Construction- Control flow graph of a program under test may be constructed manually or automatically with related tools. It helps tester to select target paths.
2. Target Basis Set Selection- On the basis of cyclomatic complexity computed by connection matrix automatically and based on probability of occurrence of a specific path, target basis set is selected. Furthermore, GA can also be used as automatic tool to select target basis set.
3. Fitness Function Construction- It is used to evaluate the distance between the executed path and the target path.
4. Program Instrumentation- Here probes are inserted at the beginning of every block of the source code to monitor the program's execution (e.g. fitness values of individuals.).

5. GA Working- Initially random test data are generated from problem domain. On the basis of instrumented program and fitness evaluation GA generates new test data to achieve the target path and at last after suitable iterations on the basis of termination condition, new optimized test data is generated automatically.

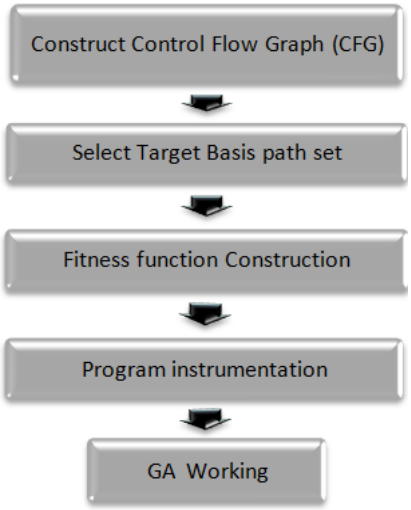

Fig. 1: Test Data Generation

III. TWO FITNESS FUNCTION FOR BASIS PATH TESTING BASED TEST DATA GENERATION

A fitness function plays a crucial role in GA. An appropriate fitness function can not only improve the probability to find a solution, but also consume fewer system resources in the process [10]. Branch distance based fitness function and approximation level based fitness function [11] are both used as basic fitness functions for GA based path oriented test data generation.

A. Approximation Level Based Fitness Function

It is used to distinguish between different test data individual's executed path from the target path by counting the number of branching nodes not traversed by current executed path, so aim is to minimize approximation level [11], [12]. As example Fig. 2 illustrates ALBFF for the target path Tp (in highly dark lines) which contains three decision nodes: A, B and C. If the individual path p1 diverges from the target path at the level of node A, then approximation level used for calculating the fitness function will be 2 (means p1 missed 2 decision nodes to traverse to achieve target path Tp). If the individual diverges at level of node B, then it will be 1 and if traverses all nodes then approximation level will be 0. And our aim is to minimize the approximation level for a path as fitness function.

Tp=set of nodes traversed={A,B,C}

F(P)=Approximation level of path P or number of non traversable nodes by path P corresponding to target path Tp and aim is to minimize it.

P1={A} F(P1)=2

P2={A,B} F(P2)=1

P3={A,B,C} F(P3)=0

So, path P3 is best among above to match target path Tp.

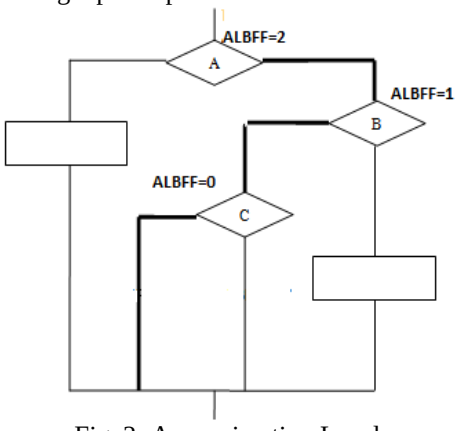

Fig. 2: Approximation Level

B. Branch Distance Based Fitness Function

It is used to distinguish between different individuals who execute the same program target path [13]. Branch distance is calculated for an individual by using branching condition in the branching node in which the target node is missed.

Every branch is composed of logical expressions. To force branch (to be true or false) to follow target path we have to adjust or search or optimize the input data of that branch.

Branch output depends upon input and logical expressions to get desired output from branch. Here some branch distance based functions are placed on the basis of logical expression in the branch. So, overall Branch distance function is based on logical expression in branch and required output. The branching conditions are evaluated based on a table as here table I shows a branch distance function [13].

Let C= (x+y>z) AND (y+z>x) AND (z+x>y) AND x>0 AND y>0 AND Z>0.

Required output of C is TRUE. So, F(C)=F(C1)+F(C2)+F(C3)+F(C4)+F(C5)+F(C6),where C1=(x+y>z),C2=(y+z>x),C3=(z+x>y),C4=(x>0),C5=(y>0),C6=(z>0) and these all are also required to be TRUE.

So, F(C1)=(z-x-y), F(C2)=(x-y-z), F(C3)=(y-z-x), F(C4)= -x, F(C5)= -y, F(C6)= -z, Thus F(C)= -2*(x+y+z) and our aim is to minimize F(C).

Table I : Korel's Branch Distance Function

Logical expression in branch C	**F(C)=Branch Distance If branch output=0=false**	**F(C)=Branch Distance If branch output=1=true**
x=y	-abs(x-y)	abs(x-y)
x~=y	abs(x-y)	-abs(x-y)
x>y	x-y	y-x
x>=y	x-y	y-x
x<y	y-x	x-y
x<=y	y-x	x-y
C1 OR C2	F(C1)+F(C2)	Min (F(C1), F(C2))
C1 AND C2	Min (F(C1),F(C2))	F(C1)+F(C2)

IV. EXPERIMENTAL STUDIES

A. Experimental Settings

This section involves triangle classifier as a benchmark problem, control flow graph generation for benchmark problem; basis set selection, program instrumentation to return fitness value, min to max fitness conversion, parameters settings.

Triangle Classification program: Triangle classification program has been widely used in the research area of software testing. Triangle classification aims to determine if three input edges can form a triangle and so what type of triangle can be formed by them. Fig. 3 gives program's source code under MATLAB.

CFG Construction: Control flow graph of the triangle classifier program is represented in fig. 4; control flow graph can be constructed manually or automatically with related tools. CFG helps to select target paths and helps to compute cyclomatic complexity, which comes out to be four.

```
function[ ]=triangleclassifier (x, y, z)
   if ( x+y>z)&(y+z>x)&(z+x>y) & (x>0) & (y>0) & (z>0)        %1
      if (x~=y)&(y~=z)&(z~=x)                                 %2
         disp ('Scalane');                                    %3
      else
         if (x==y) & (y~=z)||(y==z) & (z~=x) || (z==x) & (x~=y)  %4
            disp ('Isosceles');                               %5
         else
            disp ('Equilateral');                             %6
         end
      end
   else
      disp ('Not-a-Triangle');                                %7
   end
end
```

Fig. 3: An example program

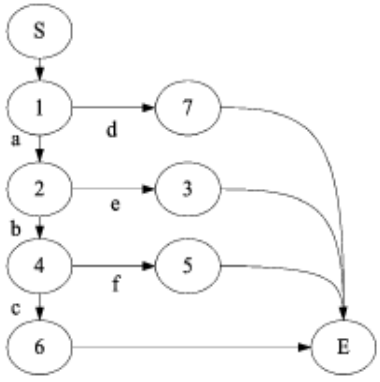

Fig. 4: Control flow graph of the triangle classifier

Basis set of independent paths: Basis set is a finite set of linearly independent paths through a standard flow graph [CAB1982]. So, there are 4 linear independent paths in triangle classifier flow graph.

Path 1: < d > //Not a triangle
Path 2: < a e> //scalene
Path 3: < a b f > //Isosceles
Path 4: < a b c > //Equilateral

According to the probability theory, the probability of achieving the Path4 is 2^{-24} (that is (2^{12} *1*1)/ (2^{12}*2^{12}*2^{12}), if each positive integer edge is 12 bits), which means it will take random testing 2^{24} tests to achieve it. Thus, Path 4: < a b c > is the most difficult path to be covered in path testing. Therefore, firstly the path < a b c > is selected as the target path.

Instrumented Program to return fitness value: On the basis of Branch distance based fitness function (BDBFF) discussed in section 3 and to cover target path 4 < a b c > for equilateral triangle, source code of the instrumented program (for improved fitness function) of triangle classifier is represented in fig. 5 and corresponding to Approximation level based fitness function (ALBFF) instrumented program is represented in fig. 6, taking individual as input and returning its fitness value (Fit) and aim is to minimize value of Fit in GA.

```
function [ fit ] =BDBFFtriangleclassifier (x, y, z)
   BDBFF(1)= -2*(x+y+z);
   if ( x+y>z) & (y+z>x) & (z+x>y) & (x>0) & (y>0) & (z>0)      %1
      BDBFF(2)=min(min(abs(x-y),abs(y-z)),abs(z-x));
      if (x~=y) & (y~=z) & (z~=x)                               %2
          disp ('Scalane');                                     %3
      else
         BDBFF(3)= -abs(y-z)-abs(z-x)-abs(x-y);
         if (x==y) & (y~=z) || (y==z) & (z~=x) || (z==x) & (x~=y) %4
            disp ('Isosceles');                                 %5
         else
            disp ('Equilateral');                               %6
         end
      end
   else
     disp ('Not-a-Triangle');                                   %7
   end
fit = Sum (BDBFF);
end
```

Fig. 5: BDBFF Instrumented program

Min to Max fitness Conversion: Simply combined fitness function and proposed improved fitness function are aim to minimize the fitness value. To make these run with Simple genetic algorithms, fitness function is converted to maximization form. So, value of Fit is negated.

Fit1=Sum (BDBFF);
Fit2=ALBFF;
Fit=Fit1+Fit2;
Fit= - (Fit) ;

```
function [ fit ] =ALBFFtriangleclassifier (x, y, z)
   ALBFF=2;
   if ( x+y>z) & (y+z>x) & (z+x>y) & (x>0) & (y>0) & (z>0)      %1
      ALBFF=1;
      if (x~=y) & (y~=z) & (z~=x)                               %2
          disp ('Scalane');                                     %3
      else
         ALBFF=0;
         if (x==y) & (y~=z) || (y==z) & (z~=x) || (z==x) & (x~=y) %4
            disp ('Isosceles');                                 %5
         else
            disp ('Equilateral');                               %6
         end
      end
   else
     disp ('Not-a-Triangle');                                   %7
   end
fit = ALBFF
end
```

Fig.6: ALBFF Instrumented program

Parameter Settings

Settings of SGA are as followings:

(1) Coding: Standard binary string
(2) Length of chromosome: 12 bits*3=36 bits and each input variable ranges from 1 to 4096.
(3) Population size=40
(4) Selection method: Tournament selection
(5) Two-point crossover (pc): 0.8
(6) Mutation probability (pm): 0.03
(7) Replacement: Steady state replacement
(8) No. of Generations: 10

The first generation of test data was generated from their domain at random. For example, by executing the command (below) in Matlab, 40 three dimensional vectors as the first generation of test data with values of each input variable ranges from 1 to 4096 was generated.

Round (unifrnd (1, 4096, 40, 3));

B. Experimental Results

In this section, average number of test data was generated after 40 experiments (covering four paths) for basis path testing by Approximation level based fitness function & Branch distance based fitness function and then results were compared. At first, results were conducted with same initial population throughout 40 experiments and then results were conducted with random initial populations in each of the 40 experiments.

- **Experimental results with identical initial population**

The initial population in this section was the same in each of the 40 experiments. This population was generated at random by executing the command (below) in MATLAB and was named as Pop40. In this population Pop40 (of 40 population size), 20 individuals formed a not-a-triangle, 20 formed scalene, no any individual formed isosceles and equilateral. After forty experiments on same Pop40 as initial population, average number of test data (covering four paths) by ALBFF and BDBFF were generated. This result is represented in bar chart form in fig. 7.

Pop40 = Round (unifrnd (1, 4096, 40, 3));

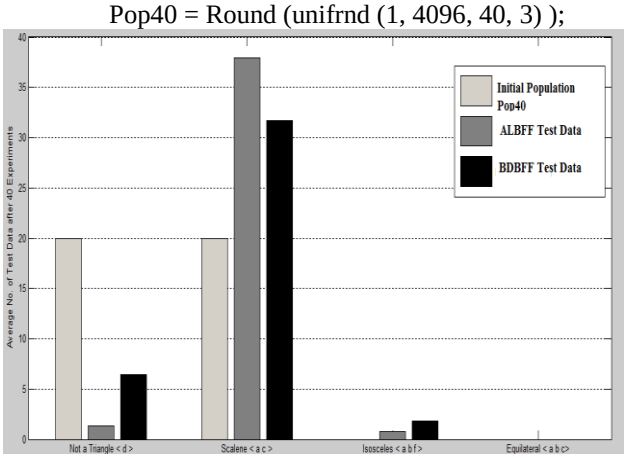

Fig. 7: Average number of test data (covering four paths) with identical initial population after 40 experiments

- **Experimental Results with Random Initial Population**

The initial population in this section was randomly generated in each of the 40 experiments and averaged initial population after 40 experiments was named as PopRnd. Other settings were identical with above experiment. After forty experiments with random initial population, average no. of test data (covering four paths) were generated by ALBFF and BDBFF. This result is represented in bar chart form in fig. 8.

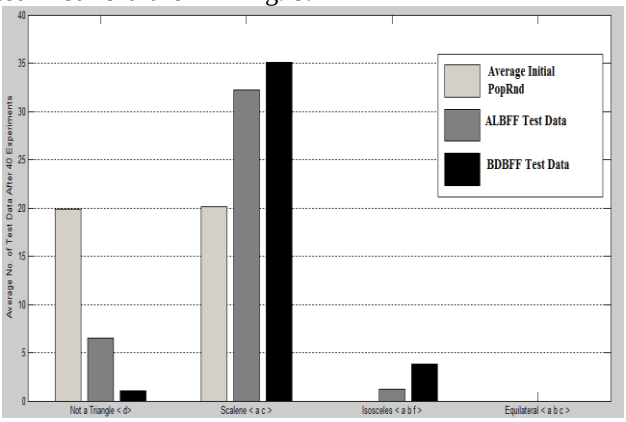

Fig. 8: Average number of test data graphically (covering four paths) with random initial population after 40 experiments

Since the path <a b f> is close to the target path <a b c>, so if no any approach generates any test data corresponding to <a b c> then an approach who will generates more test data corresponding to < a b f> can reach the target path <a b c> more quickly and can be said best approach. So, on the basis of experimental results with same initial population and random initial populations in forty experiments, two conclusions have been made. First conclusion is both approaches (BDBFF and ALBFF) are better than random search; second conclusion is that Basis path testing using BDBFF is better than basis path testing using ALBFF.

V. CONCLUSION

This paper applied standard genetic algorithm for test data generation under basis path coverage criteria and make a comparison of two fitness functions. After depicting basic process flow of path oriented test data generation using GA, these two fitness functions ALBFF and BDBFF were described. Using a triangle classification program as an example, experiment results show that BDBFF based approach can generate path oriented test data more effectively and efficiently than ALBFF based approach does. Future work will include investigating target path selection strategy, comparing branch distance based fitness function with others and doing experiments on larger and more complex programs. Furthermore, we intend to build an automated evolutinary structural testign tool based on these work.

REFERENCES

[1] B. Antonia, "Software Testing Research: Achievements, Challenges, Dreams", in Future of Software Engineering: IEEE Computer Society, 2007.

[2] G. J. Myers, "The Art of Software Testing", 2nd ed.: John Wiley & Sons Inc, 2004.

[3] B. W. Kernighan and P. J. Plauger, "The Elements of Programming Style", McGraw-Hill, Inc.New York, NY, USA, 1982.

[4] Tom Mc Cabe, "Structural Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-99, NIST, Washington, D.C., 1982.

[5] T. K. Wijayasiriwardhane, P. G. Wijayarathna and D. D. Karunarathna, " An Automated Tool to Generate Test Cases for Performing Basis Path Testing", Proc. International Conference on Advances in ICT for Emerging Regions (ICTer2011), IEEE Computer Society, August 2011, doi:10.1109/ICTer.2011.26.

[6] G. L. Latiu, O. A. Cret and L. Vacariu , "Automatic Test Data Generation for Software Path Testing using Evolutionary Algorithms", Proc. Third International Conference on Emerging Intelligent Data and Web Technologies, IEEE Computer Society, August 2012, doi: 10.1109/EIDWT.2012.25.

[7] G. M. C. Michael and M. Schatz, "Generating software test data by evolution", IEEE Transactions on Software Engineering, vol. 27, 2001, pp. 1085-1110.

[8] W. Joachim and S. Harmen,"Suitability of Evolutionary Algorithms for Evolutionary Testing", in Proceedings of the 26th conference on Prolonging Software Life: Development and Redevelopment: IEEE Computer Society, 2002.

[9] M. Alzabidi, A. Kumar and A. D. Shaligram, "Automatic Software Structure Testing by Using Evolutionary Algorithms for Test Data Generations", IJCSNS International Journal of Computer Science and Network Security, VOL. 9 No.4, April 2009.

[10] S. N. Sivanandam, S. N. Deepa, "Introduction to Genetic Algorithms", Springer, 2008.

[11] J. C. Lin and P. L. Yeh, "Automatic test data generation for path testing using Ga", Information Sciences, vol. 131, 2001, pp.47-64.

[12] C. Yong, Z. Yong, S. Tingting and L. Jingyong, "Comparison of two Fitness Function for GA-based Path-Oriented Test Data Generation", Proc. Fifth International Conference on Natural Computation (ICNC09z), IEEE ComputerSociety,August2009,doi:10.1109/ICNC.2009.235.

[13] B. Korel, "Automated software test data generation", IEEE Transactions on Software Engineering, vol. 16, 1990, pp. 870-879.

