Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-4, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

Genetic Algorithm- A new approach to software testing

Kriti Jindal[1], Lokesh Garg[2] & Vibhor Sharma[3]
[1]student, GGSIPU, India
[2]student, GGSIPU, India and [3]Assistant Professor, GGSIPU, India.

Abstract*: There is always a need to test the software product for quality assurance, but it is a very tedious task. To increase efficiency and reduce time and cost, automated testing is done. In this paper, our focus is on path testing using the principles of genetic algorithm. We are using the combination of two existing algorithms, branch distance based fitness function (BDBFF) and approximation level based fitness function (ALBFF) to determine the best value of the fitness function. To implement this fitness function we are using MATLAB optimization tool.*

1. Introduction

1.1 Software testing

Testing the software for quality assurance is one of the main tasks which ensure satisfaction of the client. However, the process, largely being manual and static takes up a lot of resources. This calls for a need for automated testing methods which can handle the changing dynamics of the software efficiently. The software testing process can be largely classified as static and dynamic. Static methods fail to deliver the results in case of infinite loops, arrays or procedural calls. Hence, the need to switch to dynamic methods arises. This is where the amalgamation of software testing and artificial intelligence techniques comes in. A lot of research is being done on this area and researchers have come to the conclusion that the cross connection of the two can be very helpful in generating effective test cases.

1.2 Path testing and Genetic Algorithm

Path testing can detect almost seventy percent of the errors in a program under test. This is because it covers the criteria of path coverage and branch coverage as well. Hence, we have chosen path coverage as the main criteria to help generate effective test cases. Genetic algorithms are adaptive heuristic search techniques that are based on the principles of natural selection and the survival of the fittest ideology. Using genetic algorithm with software testing will lead to the production of optimum test cases which can then be tested to give the best results.

2. Procedure

For the implementation purpose, we are taking the example of a triangle classifier problem which on taking the input from the user determines whether the triangle is equilateral, isosceles, and scalene or a triangle at all. The code of the program is then used to draw a control flow graph (CFG) which will be used to determine all the possible paths from the start node to the target node.

The algorithm of triangle classifier is as follows:

```
Function[]=triangleclassifier(x,y,z)
 if(x+y>z)&(y+z>x)&(z+x>y)&(x>0)&(y>0)&(z>0)
    if(x~=y)&(y~=z)&(z~=x)
          disp('Scalene');
    else  If(x==y)&(y~=z)||(y==z)&(z~=x)||(z++x)&(
    x~=y)
          disp('Isosceles');
    else
          disp('Equilateral');
    end
 end
 else
    disp(' Not a triangle');
 end
end
```

2.1 Control flow graph and Independent paths

From the above program, the corresponding control flow graph is made and the various possible paths are named a, b, c, d, e and f. Basis set is a finite set of linearly independent paths through a standard flow graph. According to the control flow graph drawn, there are four independent paths as follows:
Path 1: <d> //not a triangle
Path 2 :< a e>// scalene
Path 3: <a b f>//isosceles
Path 4: <a b c>//equilateral

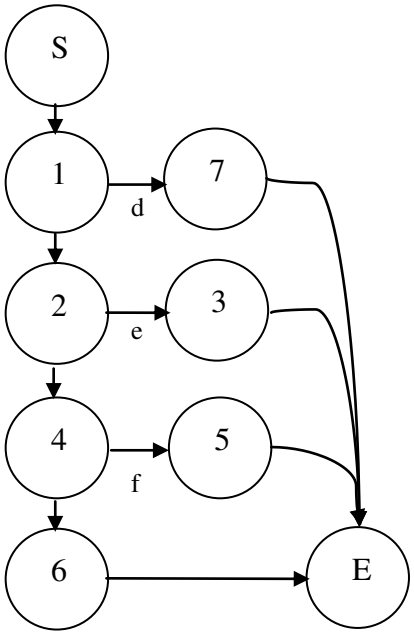

Figure 1. CFG of triangle classifier.

3. Fitness functions

Here, our fitness function comprises of two other already existing fitness functions which are branch distance based fitness function (BDBFF) and approximation level based fitness function (ALBFF).

3.1 Branch distance based fitness function (BDBFF)

It is used to distinguish between different individuals who execute the same program target path. 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 and aim is to minimize it. The branching conditions are evaluated based on a Table 1 as shown.

3.2 Approximation level based fitness function (ALBFF)

It is used to distinguish between different tests data individual's executed path from the target path by counting the number of branching nodes not traversed by current executed paths, so aim is to minimize approximation level.

Table 1. Korel's branch distance function.

Logical expression in branch C	F(C)= Branch Distance If branch output=0=false	F(C)= Branch Distance If brach 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))

For example, the below figure 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.

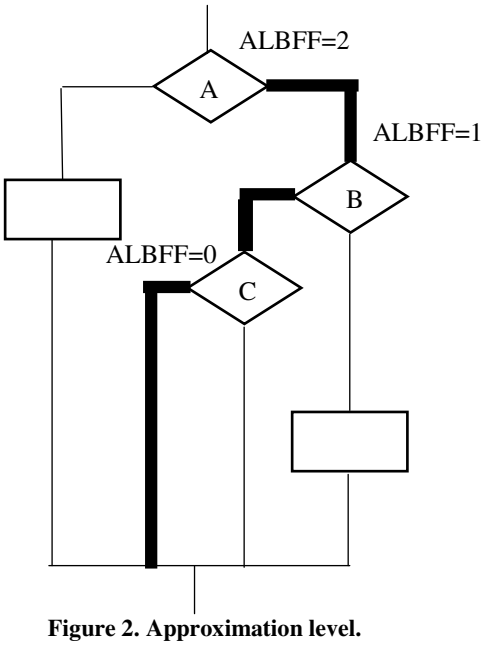

Figure 2. Approximation level.

3.3 Extended level branch fitness function (ExLBB)

In this fitness function, branch distance based fitness function and approximation level based fitness functions are combined. Here two changes have been done.

First improvement is, magnitude of ALBFF is made higher by multiplying value of ALBFF to a large constant value, so that ALBFF can show its significance in final fitness function to distinguish between different test data individual's executed path from the target path.

Second improvement in this is when all branching nodes are covered then also ALBFF value is not taken as zero for last branching node but it is taken as half of the ALBFF of the parent branching node of the last node.

Let ALBFF for second last branching node is 1, and then ALBFF of last node is 0.5. For all child paths originating from that last branching node, ALBFF is put zero only for that child path which fulfills target path completely and for all rest child paths ALBFF is put half of the ALBFF corresponding to last branching node i.e. 0.25.

The fitness function obtained is the summation of fitness function of both ALBFF and BDBFF.

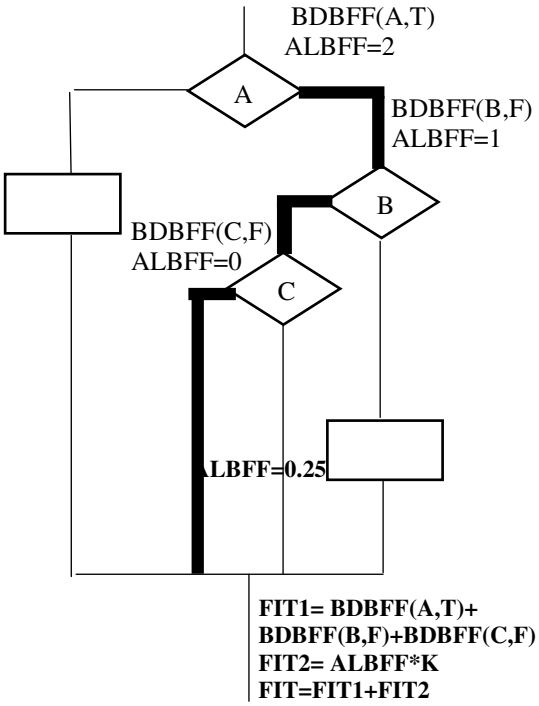

Figure 3. ExLB fitness function (Extended level branch)

4. Experimental Understanding

This section involves implementation of triangle classifier for generation of test cases using genetic algorithm optimization tool.

4.1 Optimization tool settings

Settings of SGA under matlab are as follows:

1. Coding: Standard binary string
2. Bounds of variable:1 to 4096
3. Population size: 40
4. Selection method: Tournament selection
5. Two point crossover: 0.8
6. Mutation probability: 0.03
7. Replacement: Steady state replacement
8. K-Large constant: 50000

The first generation for optimization of fitness function is created by using random function of matlab i.e.:(round(unifrnd(1,4096,40,3));). It generated a matrix of 40X3, each column representing a variable for value of side length of triangle.

4.2 Experimental results

This section consist of the resulting values of fitness function after number of generations. A graphical representation shows the value of fitness function after every generation. Then the optimization tool calculates the best fitness value and mean fitness value. The best value represents the value of fitness function on which we will get optimum test case for the path under testing. Another result shows the effect of increasing the size of population from 30 to 50 in the intervals of 10.

4.2.1 Best and mean fitness value

Using the experimental settings stated above, following graph is obtained to evaluate best and mean fitness value.

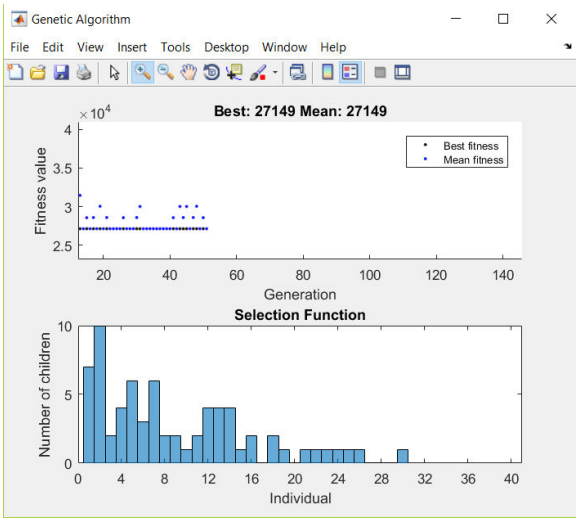

Figure 4. Best and mean value obtained

It is being seen that for population size 40, the best value obtained is: 27149 and mean value

obtained is: 27149. It also shows the number of children generated for each individual by the process of crossover and mutation.

4.2.2 Comparison of best fitness value

By varying the population size in the intervals of 10, it is being observed that the value of fitness function reaches to more optimum value. Firstly, the population size is set to 30.

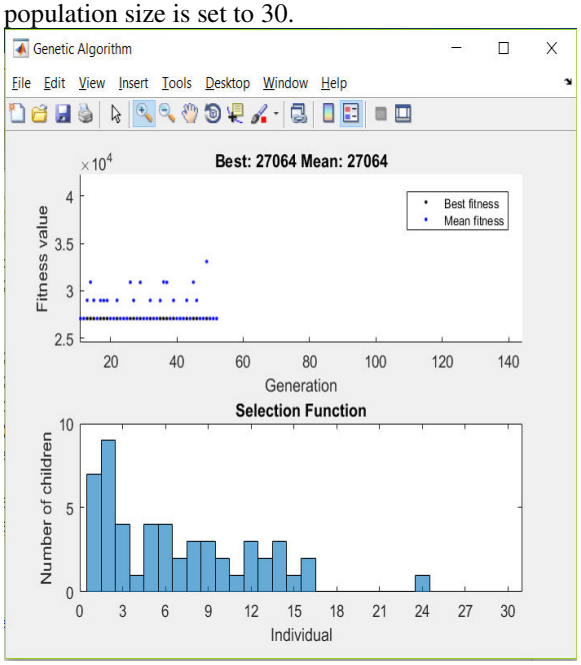

Figure 5. best and mean value obtained at pop. size 30.

Then it is incremented by the interval of 10 up to 50.

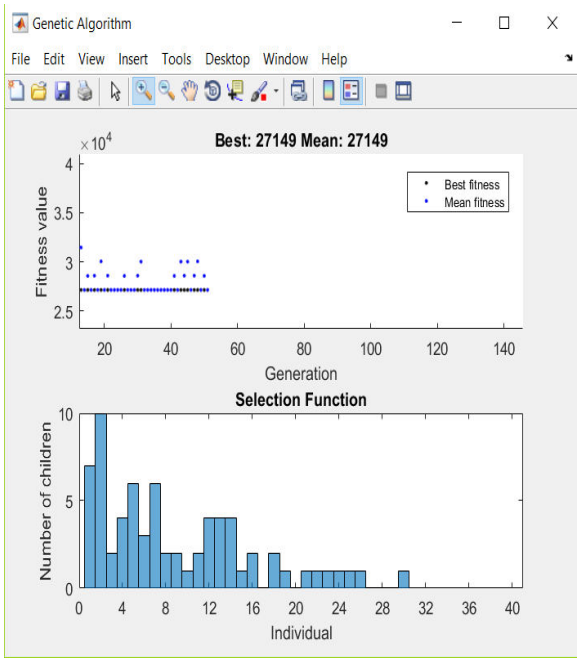

Figure 6. Best and mean value obtained at pop. size 40.

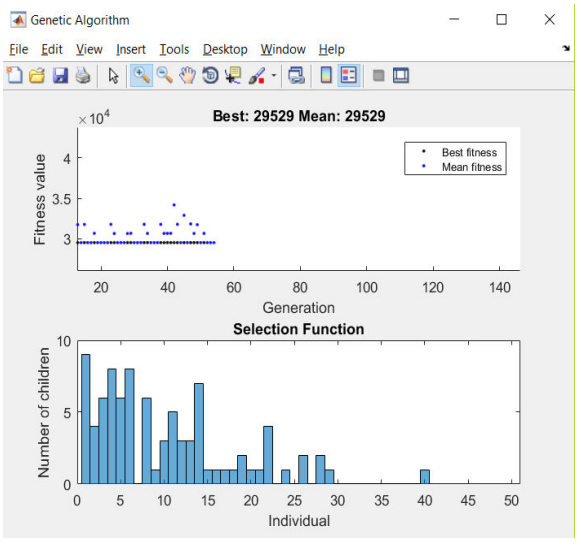

Figure 7. Best and mean value at pop. size 50.

5. Conclusion

In this paper, we have implemented the extended level fitness function which is a combination of approximation level based fitness function and branch distance based fitness function to generate test cases for our subjected program of triangle classifier. The type of software testing being implemented is basic path testing. With the help of genetic algorithm and optimization tool of Matlab, it is much easier to generate test cases which are optimum to particular path under testing. Also we have concluded that by increasing the size of population, the value of fitness function approach towards more optimum value. Future work will include to apply this fitness function in the testing of real world problems to generate optimum test cases with optimum time and cost.

6. References

[1] Briand, L. C., Daly, J., and Wüst, J., "A unified framework for coupling measurement in object oriented systems", *IEEE Transactions on Software Engineering*, 25, 1, January 1999, pp. 91-121.

[2] Maletic, J. I., Collard, M. L., and Marcus, A., "Source Code Files as Structured Documents", in *Proceedings 10th IEEE International Workshop on Program Comprehension (IWPC'02)*, Paris, France, June 27-29 2002, pp. 289-292.

[3] Marcus, A., *Semantic Driven Program Analysis*, Kent State University, Kent, OH, USA, Doctoral Thesis, 2003.

[4] Marcus, A. and Maletic, J. I., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", in *Proceedings 25th IEEE/ACM International Conference on Software Engineering (ICSE'03)*, Portland, OR, May 3-10 2003, pp. 125-137.

[5] Salton, G., *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989.