Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 70 (2015) 593 – 602

Procedia Computer Science

4th International Conference on Eco-friendly Computing and Communication Systems

Basis Path Testing Using SGA & HGA with ExLB Fitness Function

Deepak Garg[a*], Pallvi Garg[a]

[a]*Department Of Computer Science and Applications, Kurukshetra University Kurukshetra 136119, Haryana,India.*

Abstract

Software testing is the most significant analytic quality assurance for software pr*oducts, but it is very expensive and time consuming process. This limitation is overcome by automatic testing to reduce high cost and to increase reliability & efficiency as compared to manual testing. Basis path testing is a coverage criterion of software testing that can detect almost sixty five percent of errors in program under test. In this paper a new fitness function has been proposed named as Extended Level Branch (ExLB) Fitness function for basis path testing using simple genetic algorithm (SGA) and hybrid genetic algorithm (hill climbing with selection operator). Using a triangle classifier as program under test, performance of SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function have been compared using MATLAB. Experimental results showed that SGA with ExLB Fitness Function (proposed approach) performs better than the SGA with Simply Combined Fitness Function and HGA using ExLB Fitness Function is better than all these approaches in terms of test data generation under basis path coverage criteria.

Peer-review under responsibility of the Organizing Committee of ICECCS 2015

Keywords: Approximation Level Based Fitness Function; Basis Path Testing; Branch Distance Based Fitness Function; Extended Level Branch Fitness Function;Simple Genetic Algorithms; Software Testing; Hybrid Genetic Algorithm.

*Corresponding author. Tel.: +919467210641
E-mail address: erdeepakgarg21@gmail.com

Peer-review under responsibility of the Organizing Committee of ICECCS 2015
doi:10.1016/j.procs.2015.10.044

1. Introduction

1.1 Background

In general, manual software testing accounts for approximately 50 percent of the elapsed time and more than 50 percent of the total cost in software development1 & automated software testing is a promising way to cut down time and cost2. Path oriented test data generation is an un-decidable problem. On the whole, test data generation methods can be classified into two types: Static Methods and Dynamic Methods. Static methods consist of domain reduction and symbolic execution etc. Static methods may get into trouble when they handle indefinite arrays, loops, pointer references and procedure calls3. Dynamic methods include random testing, local search approach, goal oriented approach, chaining approach and evolutionary approach4. As values of input variables are determined when programs execute, dynamic test data generation can avoid those problems that are confronted with the static methods. As a robust search method in complex spaces (robust to dynamic), genetic algorithm was applied to test data generation4 in 1992 and this evolutionary approach has been in interest since then.

1.2 Need of Basis Path Testing & Genetic Algorithms

Basis path testing strategy can detect almost 65 percent of errors in program under test5 and various structural test data generation problems can be represented into a path oriented test data generation problem6. Basis path coverage criteria cover 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. Furthermore related work3 indicates that genetic algorithms based test data generation outperforms other dynamic approaches and static approaches. So, Genetic Algorithm has been used in this paper for generating test data set from a pool of randomly generated test data automatically.

The rest of the paper has been organized as follows. In section 2, brief introduction to three basic existed fitness functions (Branch Distance Based Fitness Function (BDBFF), Approximation Level Based Fitness Function (ALBFF) and Simply Combined Fitness Function (BDBFF+ALBFF)) are given under basis path coverage criteria for test data generation. In section 3, ExLB Fitness Function (our proposed approach) and HGA (hill climbing with selection) has been given. In section 4, experimental settings have been given. In section 5, experimental results consisting of comparisons have been given. Finally, conclusions and direction for future work has been given in section 6.

2. Existed Fitness Function For Test Data Generation

2.1 Branch Distance Based Fitness Function

It is used to distinguish between different individuals who execute the same program target path7. 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 as here table I shows a branch distance function7.

Table I: Korel's Branch Distance Function

Logicalexpression in branchC	**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 ANDC2	Min (F(C1),F(C2))	F(C1)+F(C2)

2.2 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 path8, so aim is to minimize approximation level9. As example Fig. 1 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.

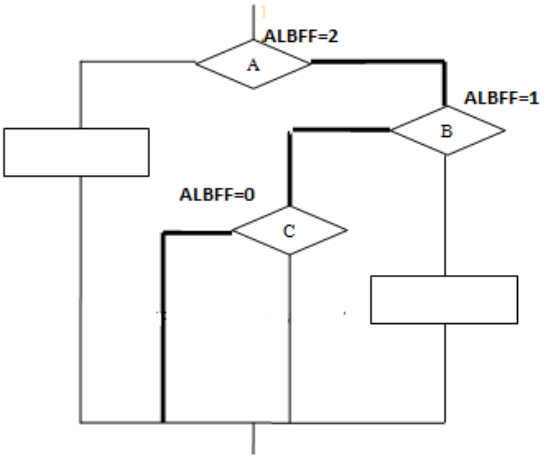

Fig. 1: Approximation Level

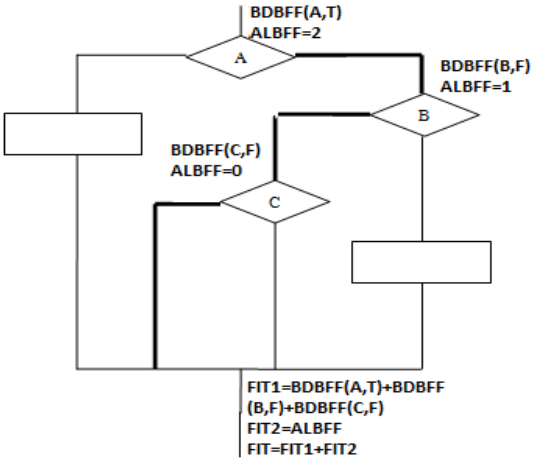

Fig. 2: Simply Combined fitness function (BDBFF+ALBFF)

2.3 Simply Combined Fitness Function (BDBFF+ALBFF)

In this fitness function BDBFF and ALBFF are both used in final fitness function value. But there are 2 limitations in this. First limitation is that ALBFF magnitude in every problem is very low than the BDBFF magnitude. So, value of ALBFF cannot show its significance in final fitness function, while ALBFF value's more impact is needed to distinguish between different test data individual's executed path from the target path and second limitation is ALBFF assigns level 0 towards last branching node but there are still at least 2 paths from last branching node. So, ALBFF can't fully identify target path. As example Fig. 2 illustrates this simply combined fitness function (BDBFF+ ALBFF), the target path Tp (in highly dark lines) which contains three decision nodes: A, B and C. Fit1 is the BDBFF value and Fit2 is the ALBFF value and FIT is the final fitness value. Aim is to minimize the value of final fitness i.e. FIT.

3. Methodology

3.1 ExLB Fitness Function (Extended Level Branch fitness function)

In this ours proposed fitness function both BDBFF and ALBFF are used, but 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. As example this improved Fitness function (BDBFF+ALBFF) for the target path Tp (in highly dark lines) which contains three decision nodes: A, B and C is represented in fig. 3.

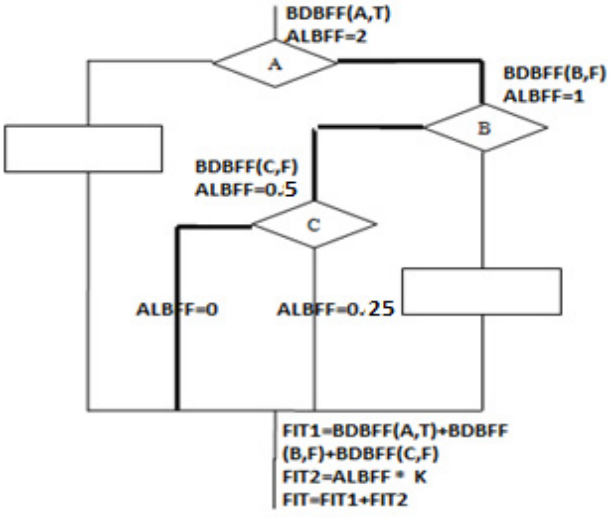

Fig. 3: ExLB Fitness Function (Extended Level Branch)

//Computing ALBFF for ExLB fitness function (Target path Tp={A, B, C, C-False})

1: Count the number of branching nodes in CFG
   Result: In fig 4.2 it is 3, TOTAL=3.
2: Set the initial value of ALBFF= (TOTAL-1)
   Result: ALBFF=3-1=2

3: Assign ALBFF in decrementing fashion towards branch nodes by traversing branch nodes from top to bottom until move on to last branch node.

```
While (node~=last node)
    ALBFF towards node=ALBFF;
    ALBFF=ALBFF-1;
End
Result: ALBFF towards node A=2;
        ALBFF towards node B=1;
```

4: Assign ALBFF towards last node=Half (ALBFF towards second last node)

Result: ALBFF towards last node C=Half (1)=0.5;

5: Find child paths of last node.

Result: In above figure there are 2 child paths of last node C (C-True and C-False)

6: Assign final ALBFF towards the child paths on the basis of match to target path Tp.

```
If (child path matched in Tp target path)
    Set ALBFF towards child path=0;
Else
    Set ALBFF towards unmatched child path= Half (ALBFF towards last node);
End
Result: ALBFF towards C-True=0.25;
        ALBFF towards C-False=0;
```

7: Finally, increase the magnitude of ALBFF by multiplying it with a large constant value.

FIT2=ALBFF * k;

3.2 Hybrid Genetic Algorithm

Memetic algorithm is often called as hybrid genetic algorithm. Incorporating problem specific information (like Hill climbing, Simulated Annealing etc) in a genetic algorithm at any level of genetic operation forms a hybrid genetic algorithm10. MA is motivated by Dawkins notation of a meme. Meme which is unlike genes can adapt them. A meme is a unit of information that reproduces itself as people exchange ideas. In this paper, hill climbing has been applied after selection operation as a search heuristic.

Outline of Hybrid Genetic Algorithm:

1. [START] Initially the population is randomly generated of N chromosomes as population size.
2. [FITNESS] Evaluation of fitness value F(X) for each chromosome X is done.
3. [NEW POPULATION] Create a new population by repeating the following steps until the new population is completed
   (i) [SELECTION] Select two parent chromosomes from a population according to their fitness, let named as mate1 and mate 2
   // Applying local search to each selected individual
   Optmate1=hill climbing (mate 1)
   Optmate2=hill climbing (mate 2)
   (ii) [CROSSOVER] With a crossover probability pc crossover the optmates to form new offspring as like exploitation.
   (iii) [MUTATION] With a mutation probability pm mutate new offspring as like exploration.
   (iv) [ACCEPT] Place new offspring in a new population.
4. [REPLACE] Old population is replaced by newly generated population using some scheme.
5. [TERMINATION TEST] If the end condition is satisfied then stop and return the best solution in current population.
6. [LOOP] Go to step 2 (as a failure of step 5).

4. Experimental Settings

This section involves triangle classifier as a benchmark problem, CFG for benchmark problem; basis set selection, program instrumentation to return fitness value, parameters settings.

4.1 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. 4 gives program's source code under MATLAB. Fig. 5 represents control flow graph of it.

4.2 Basis set of independent paths

Basis set is a finite set of linearly independent paths through a standard flow graph[11]. 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 (212 *1*1)/ (212*212*212), if each positive integer edge is 12 bits), which means it will take random testing 224 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.

```
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. 4: An example program

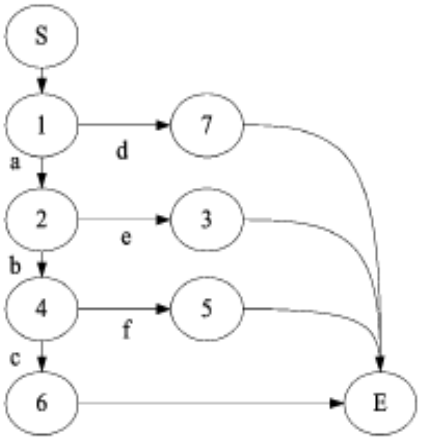

Fig. 5: Control flow graph of the triangle classifier

4.3 Instrumented Program to return fitness value

On the basis of ExLB fitness function (proposed approach) 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. 6, taking individual as input and returning its fitness value (Fit). Fit1 corresponds to BDBFF, Fit2 corresponds to ALBFF and Fit is the final fitness function value, and aim is to minimize value of Fit in GA.

```
function[ fit ]=triangleclassifier (x, y, z)
  BDBFF(1)= - 2*(x+y+z);
  ALBFF =2;
  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));
        ALBFF=1;
        if (x~=y)&(y~=z)&(z~=x)                                %2
             disp ('Scalane');                                 %3
        else
            BDBFF(3)= -abs(y-z)-abs(z-x)-abs(x-y);
            ALBFF=0.5;
           if (x==y) & (y~=z)||(y==z) & (z~=x) || (z==x) & (x~=y)   %4
              disp ('Isosceles');                              %5
              ALBFF=0.25;
           else
              disp ('Equilateral');                            %6
              ALBFF=0;
           end
         end
    else
      disp ('Not-a-Triangle');                                 %7
   end
fit1=sum(BDBFF);
fit2=k*ALBFF;
fit=fit1+fit2;
end
```

Fig. 6: Instrumented program

4.4 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
(9) K-Large Constant: 50000

The first generation of test data was generated from their domain at random. For example, by executing the command (Round(unifrnd(1,4096,40,3));) 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.

5. Experimental Results

In this section to compare the already existed fitness functions with our proposed ExLB fitness function, average number of test data was generated after 40 experiments (covering four paths) for basis path testing by already existed SGA with Simply Combined Fitness Function (BDBFF+ALBFF), by SGA with ExLB Fitness Function (proposed approach), by Hybrid Genetic Algorithm with ExLB 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.

5.1 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 (Pop40= Round(Unifrnd (1,4096, 40,3));) 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 SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function were generated and all this result is shown in table II in numerical form. This result is also represented in bar chart form in fig. 7.

Table II: Average number of test data numerically (covering four paths) after 40 experiments with identical initial population Pop40

	Not-a-Triangle <d>	**Scalene <ae>**	**Isosceles <abf>**	**Equilateral <abc>**
Initial Pop40	20.0000	20.0000	0	0
SGA with Simply combined Fitness function	1.9000	37.8999	0.2000	0
SGA with ExLB Fitness function	1.4000	36.3999	2.2000	0
HGA with ExLB fitness function	1.3500	30.8500	7.8000	0

After 40 experiments on same population pop40, SGA with Simply Combined Fitness Function generated 0.20 test data (or 0.50%=(0.20/40) *100) corresponding to <a b f>, SGA with ExLB Fitness Function generated 2.20 test data (or 5.50%=(2.20/40)*100) corresponding to <a b f> and HGA with ExLB fitness function generated 7.80 test data (or 19.50%=(7.8/40)*100) corresponding to <a b f>.

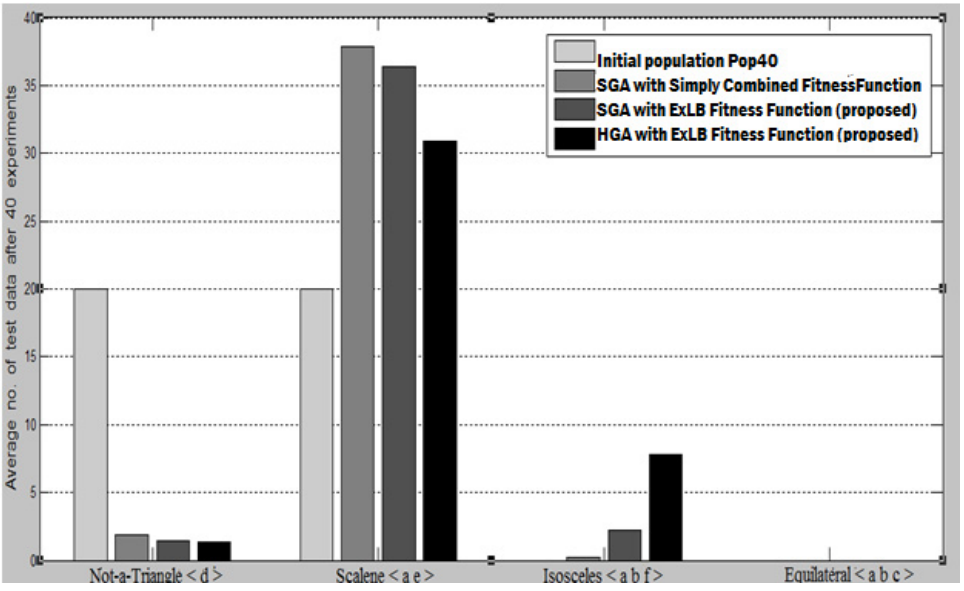

Fig. 7: Average number of test data graphically (covering four paths) with identical initial population after 40 experiments

5.2 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. In this population PopRnd (of 40 population size), 20.25 individuals formed not-a-triangle, 19.69 formed a scalene, 0.05 formed isosceles and no any individuals formed equilateral. After forty experiments with random initial population, average no. of test data (covering four paths) by SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function were generated and all this result is shown in table III in numeric form. This result is also represented in bar chart form in fig. 8.

Table III: Average number of test data numerically (covering four paths) after 40 experiments with random initial population.

	Not-a-Triangle < d >	**Scalene < a e >**	**Isosceles < a b f >**	**Equilateral < a b c>**
Averaged PopRnd	20.2500	19.6900	0.0500	0
SGA with Simply combined Fitness function	1.5500	35.9500	2.5000	0
SGA with ExLB Fitness function	2.3500	33.1000	4.5500	0
HGA with ExLB fitness function	0.8000	33.1000	6.1000	0

After 40 experiments on random initial population, SGA with Simply Combined Fitness Function generated 2.50 test data (or 6.25%=(2.50/40) *100) corresponding to <a b f>, SGA with ExLB Fitness Function generated 4.55 test data (or 11.37%=(4.55/40)*100) corresponding to <a b f> and HGA with ExLB fitness function generated 6.10 test data (or 15.25%=(6.10/40)*100) corresponding to <a b f>.

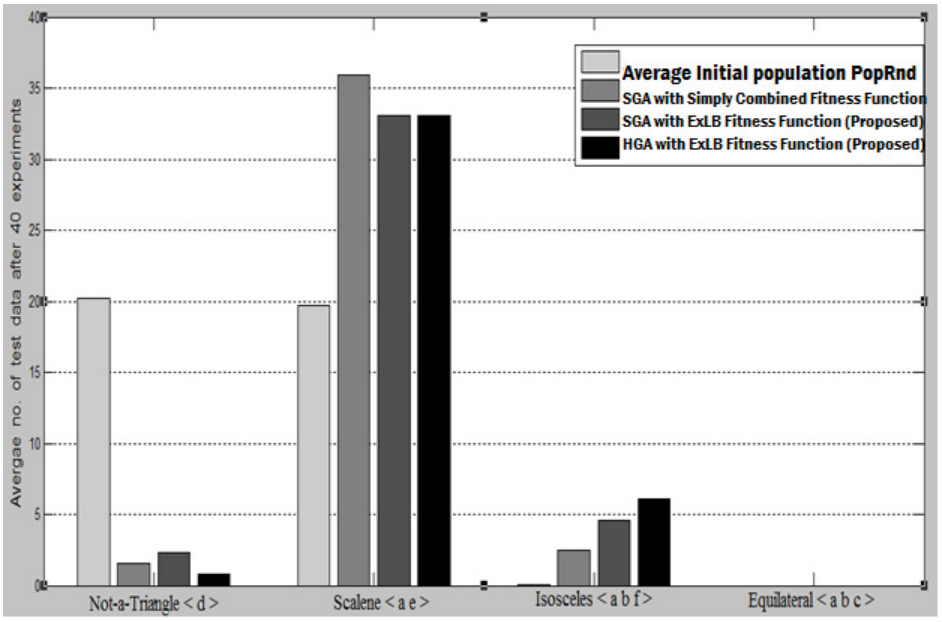

Fig. 8: Average number of test data graphically (covering four paths) with random initial population after 40 experiments

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, three conclusions have been made. First conclusion is all these three approaches (SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function) are better than random search; second conclusion is that SGA with ExLB Fitness function is better than SGA with Simply Combined Fitness Function and third conclusion is HGA with ExLB fitness function is better than all these approaches.

6. Conclusion

In this paper, a new fitness function (Extended Level Branch) has been proposed which is improvement of simply combined fitness function (BDBFF+ALBFF) to generate test data under basis path coverage criteria for a program. Comparison has been also made among Random Search, SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function by running forty experiments at first on same identical initial population and then on random initial population. On the basis of comparisons, three conclusions have been made. First conclusion is that all these three approaches (SGA with Simply Combined Fitness Function, SGA with ExLB Fitness Function and HGA with ExLB Fitness Function) are better than random search; second conclusion is that SGA with ExLB Fitness function is better than SGA with Simply Combined Fitness Function and third conclusion is HGA with ExLB fitness function is better than all these approaches. Future work will include to improve Proposed fitness function(ExLB) so that it can give more optimum test data and to compare performances of HGA with ExLB fitness function using different combinations of types of selection, crossover and replacement operators.

References

1. B. Antonia, “Software Testing Research: Achievements, Challenges, Dreams”, in Future of Software Engineering: IEEE Computer Society, 2007.
2. S. Desikan and G. Ramesh , "Software Testing: Principles and Practices”, 6th ed.: Pearson Education, 2008.
3. G. M. C. and M. Schatz, "Generating software test data by evolution", IEEE Transactions on Software Engineering, vol. 27, 2001,pp. 1085-1110.
4. S. Xanthakis, C. Ellis, C. Skourlas, A. Gall, S. Katsikasand K. Karapoulios, "Application of genetic algorithms to software testing (Application des algorithms genetics au test des logical)", in Proceedings of 5th International Conference on Software Engineering and its Applications Toulouse, France, 1992, pp. 625-636.
5. B. W. Kernighan and P. J. Plauger,“The Elements of Programming Style”, McGraw-Hill, Inc.New York, NY, USA, 1982.
6. 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.
7. B. Korel, "Automated software test data generation", IEEE Transactions on Software Engineering, vol. 16, 1990, pp. 870-879.
8. J. C. Lin and P. L. Yeh, “Automatic test data generation for path testing using Ga”, Information Sciences, vol. 131, 2001,pp.47-64.
9. 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 Computer Society, August2009,doi:10.1109/ICNC.2009.235.
10. P. Mascato and P. C. Cotta, “A gentle introduction to memetic algorithms”, handbook of Metahuristics, 2003, PP 105-144
11. T. M. Cabe, “Structural Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric”, NIST Special Publication 500-99, NIST, Washington, D.C., 1982.