RESEARCH ARTICLE OPEN ACCESS

Generation of Test Data Using Genetic Algorithm

Rupinder Kaur*, Sandeep Kaur Dhanda*
*A.P., CSE/IT Department, Baba Banda Singh Bahadur Engineering college, Fatehgarh Sahib (Punjab), India

ABSTRACT
Genetic Algorithm has been implemented to automate the generation of test data. The test data was derived from the program's structure with the aim to traverse every line of code in the software. This work uses fitness function and variables are represented in binary code. The power of using Genetic Algorithm lies in its ability to handle input data which may be of complex structure. Thus, the problem of test data generation is treated entirely as an optimization problem. The advantage of Genetic Algorithm is that through the search and optimization process, test sets are improved. The work has been tested with the help of Triangle Type Classification program.
Keywords- Genetic Algorithm, Crossover, Mutation, Test Data, Fitness Function

I. INTRODUCTION

Software testing is one of the main realistic methods to increase the confidence of the programmers in the correctness and reliability of software. Generating adequate test data manually is labour intensive and time-consuming process. This problem has motivated researchers to create test data generators that can examine a program's structure and generate adequate test data automatically. It is not a small task to judge whether a finite set of input test data is adequate or not. The goal is to uncover as many faults as possible with a potent set of a constrained number of tests. Obviously, a test series that has the potential to uncover many faults is better than one that can only uncover few. Random test data generation consists of generating test inputs at random, in the hope that they will exercise the desired software features. Often, the desired inputs must satisfy some constraints, and this makes a random approach likely to fail [2]. Software testing automation is not a straightforward forward process. For years, many researchers have proposed different methods to generate test data automatically, i.e. different methods for developing test data generators [1, 5]. Genetic algorithms that can automatically generate test cases to test selected path. This algorithm takes a selected path as a target and executes sequences of operators iteratively for test cases to evolve [3]. A method for optimizing software testing efficiency by identifying the most critical path clusters in a program [4].This paper has been organized in four sections. Section II describes the introduction to genetic algorithm, section III introduces methodology used to achieve the objectives and section IV shows the final results and their discussion.

II. GENETIC ALGORITHM

Genetic Algorithms were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan .Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. The Genetic Algorithm (GA) starts by creating an initial population of individuals, each represented by randomly generated genotype. The fitness of individuals is evaluated in some problem-dependent way, and the GA tries to evolve highly fit individuals from the initial population.

In translating the concepts of genetic algorithms to the problem of test-data generation, the following tasks are performed:

- First of all consider the population to be a set of test.
- Find the set of test data that represents the initial population. This set is randomly generated according to the format and type of data used by the program under test.
- Determining the fitness of each individual which is based on a fitness function that is problem-dependent.
- Select two individuals that will be mated to contribute to the next generation.
- Apply the crossover and mutation processes.

Various operators of genetic algorithm have been used. The selection operator chooses two individuals from a generation to become parents for the recombination process (crossover and mutation). There are different methods of selecting individuals, e.g. randomly or with regard to their fitness value. The crossover operation is used to produce the descendants that make up the next generation. Mutation operation picks a gene at random and changing its state according to the mutation probability. In a binary code, this simply means changing the state of a gene from a 0 to a 1 or vice

versa. A fitness value of an individual is the measure of its strength to survive in the next generation. It reflects the chance an individual has to be present directly in the next generation or to be selected for mating with other individuals in the current generation to produce children for next generation.

III. METHODOLOGY USED

Single point crossover and bit inversion mutation methods are used in genetic algorithm. Selection of values is based on their fitness function by using roulette wheel selection method.

The following code fragment is used to check the flow of program. All the statements are given weights. The assigned weights will be used to calculate the fitness function of each test case input given to genetic algorithm.

```
0. int a,b,c;
1. int match = 0;
2. if (a = b)
3. match = match + 1;
4. if (a == c)
5. match = match + 2;
6. if (b == c)
7. match = match + 3;
8. if (match == 0) /* if a, b and c are not equals to each other*/
9. if (a + b <= c)
10. printf("Not a triangle"); return 0.0; }
11. else if (b + c <= a) {
12. printf ("Not a triangle"); return 0.0; }
13. else if (a + c <= b) {
14. printf("Not a triangle"); Return 0.0; } else {
15. double p = (a + b + c)/2.0; printf ("Scalene");
16. return sqrt (p*(p-a)*(p-b)*(p-c)); /* compute square */ }
17. else if (match == 1) /* if (a = b ≠c) */
18. if (a + b <= c) {
19. printf ("Not a triangle"); return 0.0; }
20. double h = sqrt (pow (a, 2) - pow(c/2.0, 2)); printf ("Isosceles");
21. return (c*h)/2.0; /* compute square */ }
22. else if (match == 2) /* if (a = c ≠b) */
23. if (a + c <= b) { 24. printf ("Not a triangle"); return 0.0; } else {
25. double h = sqrt (pow (a, 2) - pow (b/2.0, 2)); printf ("Isosceles");
26. return (b*h)/2.0; /* compute square */ }
27. else if (match == 3) /* if (b = c ≠a) */
28. if (b + c <= a) {
29. printf ("Not a triangle."); return 0.0; } else {
30. double h = sqrt (pow (b, 2) - pow (a/2.0, 2)); printf ("Isosceles");
31. return (a*h)/2.0; /* compute square */ } else { /* if (a = b= c) */ printf( "Equilateral");
32. return (sqrt (3.0)*a*a)/4.0; /* compute square*/
}
```

Fig 1 Triangle Type Determination Program

IV. RESULTS

In order to compare the effectiveness of using GA, it has been compared with random testing. The following graph shows that the random testing is good enough for small size of input but as the size increases, the execution time for the program increases. So, genetic algorithm works good in that case

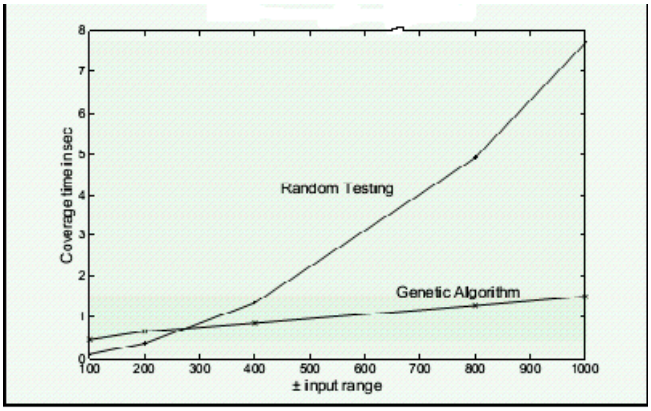

Fig 2 CPU Time taken to run genetic algorithm and random numbers

REFERENCES

[1] Mansour N, Miran Salame," Data Generation for PathTesting", Software Quality Journal, 12, 121–136, 2004,Kluwer Academic Publishers.

[2] Michael, C.C., McGraw, G.E., Schatz, M.A., Walton, C.C., " Genetic algorithm for dynamic test data generation" , Technical Report, RSTR-003-97-11

[3] Nirpal Premal B ,Kale K V," Using Genetic Algorithm for Automated Efficient Software Test Case Generation for Path Testing", International Journal in Advanced Networking and Applications Volume: 02, Issue: 06, Pages: 911-915 (2011)

[4] Srivastava, P.R., Kim, T.H." Application of genetic algorithm in software testing", International Journal of Software Engineering and Its Applications 3(4) (October 2009)

[5] Srivastava P. R. et al, "Generation of test data using Meta heuristic approach" IEEE TENCON (19-21 NOV 2008), India available in IEEEXPLORE.

.