AUTOMATIC EFFICIENT TEST DATA GENERATION BASED ON GENETIC ALGORITHM FOR PATH TESTING

Bindhyachal Kumar Singh *

ABSTRACT

Software system should be reliable and quality product. To achieve this objective we need to test the software thoroughly with adequate test data. But the automatic generation of test suite and its adequacy is the key issues during testing of a software product. This paper presents automatic generation of test data for path coverage testing using genetic algorithm. We select a target path and apply a sequence of operators iteratively to generate the test data. This test data is executed for achieve the selected target path. In this paper, we have presented an algorithm for automatic generation of test data to satisfy path coverage and a basic process flow for generation of test cases for path testing using genetic algorithm. The experimental result shows the efficiency of test data in terms of execution time and generates more effective test cases. This paper is organized into four parts: part I discuss the usages of genetic algorithm in software testing, part II describes the functionality of genetic algorithm and its operations, part III discuss the previous related works, part IV present the algorithm for generation of test cases and basic process flow of test data generation for path oriented testing, part V shows implementation of proposed algorithm and its results.

Keywords: *Software Testing, Path Coverage, Genetic Algorithm, Test-data generation.*

* School of Information and Communication Technology, Gautam Buddha University, Greater Noida.

I. INTRODUCTION

Software testing is a very tiring, tedious and time consuming process. According to Myers "Software Testing is the process of executing a program with the intent of finding errors" [1]. It is treated as a parameter for software quality assurance and represents the final review of specification, design, and coding. The importance of testing can be understood by the fact that "around 35% of the elapsed time and over 50% of the total cost are expending in testing programs" [2]. Software is expected to work, meeting customer's changing demands, first time and every time, consistently and predictably [2]. The aim of path testing is that every possible logical execution path in a program must be exercised at least once. In path testing every possible path in the program is tested. A path is nothing but set of conditions through which input travels from the starting point to the exit point. Path testing is the major test way of the software structure testing, and the key point to resolve the path testing is to find specific input data and make it possible to cover a path in the testes program.

Previous attempts to automate the test generation process have been limited and also test data generation (test cases) is a difficult process. A classical question in software development is "How much testing is enough?" [3]. Genetic algorithms have been used to find automatically a program's longest or shortest execution times. The critical point of the problem involved in the automation of software testing is the automation of software test-data generation. Test-data generation in software testing is the process of identifying a set of program input data, which satisfies a given testing criterion. A test-data generation technique must be accompanied by an application of a test data adequacy criterion, which is a predicate that determines whether the testing process is finished [4].

II. GENETIC ALGORITHM

Genetic algorithm is a computational model simulating the biological evolution process of the genetic selection theory of Darwin. It was first introduced by the professor j. Holland in 1975 in the university of Michigan [5].Genetic Algorithms begins with a set of initial individuals as the first generation, which are sampled at random from the problem domain. The algorithms are developed to perform a series of operations that transform the present generation into a new, fitter generation [6]. The algorithms are developed to perform a series of operations that transform the present generation into a new, fitter generation [4]. The use of GA techniques for the automatic generation of test data has been of great interest among researchers in recent years.

A. Operations

There are three primary operations are used for Genetic Algorithm:

i. **Selection:** This operation assigns the reproduction probability to each individual based on the output of the fitness function. As a result, the fitter individuals are allowed a better survival chance from one generation to the next.

ii. **Crossover:** Crossover results in two chromosomes being selected as parents and then crossed to produce two offspring. This results in some of the features of one parent being combined with those of the other.

iii. **Mutation:** Mutation is used to alter one or more elements in the chromosome. The purpose of the mutation operation is to maintain the diversity in a generation to prevent premature convergence to a local optimal solution. The mutation probability is given intuitively since there is no definite way to determine the mutation probability [6].

III. RELATED WORKS

There are a lot of research works done in this area but the automatic generation of test data is very big problem yet. According to *Premal I., B. Nirpal*, "Genetic algorithms that can automatically generate test cases to selected path. This algorithm takes a selected path as a target and executes sequences of operators iteratively for test cases to evolve [4]. According *to Alsmadi I.* "Using a concept of genetic algorithms to optimize the generation of test cases is the main key point for effectiveness of a Software testing techniques. This is accomplished through encoding the location of each control in the GUI graph to be uniquely represented and forming the GUI controls' graph. After generating a test case, the binary sequence of its controls is saved to be compared with new generated sequences. This paper ensures that genetic algorithm will generate a unique test case or path through the GUI flow graph every time" [7]. According to *Dhawan S., Kulvinder S. Handa, Rakesh Kumar* "A new categories of genetic codes can be applied in optimizations for the generation of software test cases, In this paper, some key definitions of genetic transformation have been used viz. crossover, mutation and selection. Some of our research shows that genetic encoding techniques have very important influence on the performance of software test cases. This paper presents the functionality of GAs, usage of GAs in software testing to the alternatives of existing software testing techniques, the implementation of GAs using MATLAB for the generation of optimized test cases" [8]. The techniques of genetic algorithm as the key algorithm to automatically generating the test data, and elaborates some specific problems need to solve in realization process: such as coding, the selection of fitness function and the improvement of hereditary operator. According to *Gary McGraw, Member, IEEE, and Michael A. Schatz* "The use of genetic algorithms (GAs) for automatic software test data generation is present in this paper. In this paper, the function is minimized by using one of two genetic algorithms in place of the local minimization techniques used in earlier research. Paper presents the implementation of our GA-based system and examines the effectiveness of this approach on a number of programs, one of which is significantly larger than those for which results have previously been reported in the literature" [9]. According to *Korel* "A dynamic test data generation approach which is based on a path wise test data generator to locate automatically the values of input variables for which a selected path is traversed. The path selection stage is not used because if unfeasible paths are selected, the result is a waste of computational effort examining these paths" [10]. This method is based on data flow analysis and a function minimization approach to generate test data which is based on the execution of the software under test. According to *Jones B. F., Sthamer H.H.* "Genetic algorithms can be used to generate test sets automatically to satisfy a predefined testing criterion. These criteria have been set by the requirements for test data set adequacy of white box testing, such as obtaining branch coverage and controlling the number of iterations of a conditional loop. The goal of the testing is clearly defined, and a fitness function which relates to this goal can be devised to give single numeric value for the fitness. The quality of the test data is enhanced by designing the fitness function to generate data close to a sub domain boundary where the likelihood of revealing an error is higher" [11].

IV. ALGORITHM AND BASIC PROCESS FLOW GRAPH OF TEST DATA GENERATION FOR PATH TESTING

To make use of genetic algorithm, a path-oriented test data generation problem requires being transformed into an optimization problem**.**

Firstly We need to generate a control flow graph A CFG is a directed graph which can be denoted as G = (N, A, s, e) where N is a set of nodes, A is a set of edges; s and e are unique entry and unique exit node respectively. Each decision node is associated with a branch predicate, which is a logical expression [12].

A. Test Data Generation Algorithm

```
TestDataGenetic(m,Ff,pop_size,Iter_max, Cp, Mp, l)
{
Maximum iteration number Iter_max
```

Input: The fitness function Ff

Population size pop_size

Crossover probability Cp

Mutation probability Mp

Number of input variables m

Total leaf nodes present l

Functions and Variables:

Initial random test case generation

Test Harness(ps)

No. of test cases in a particular population

Choosing the search node Choosesrv()

Search node srv

Changing the search node Changesrv(srv)

Samples of populations Popx[y] (array of structure)

Searching node in the Converged graph graphequi(srv)

Optimal collection of test cases evaluation Optimal(popx[])

Checking uniqueness unique(srv)

Output:

optimal test suites.

```
j=0,count=0;
Boolean y;
Popx[j]<-Test Harness(ps);
while(j!=Imax || c!=t )
Choosesrv();
Boolean X=graphequi(srv);
If(X==true)
{
If(y==unique(srv))
count++;
Else
Changesrv(srv);
}
```

```
Else
{
Population<-select(populationx,fitness);
popx[j]<-Crosssover(population,fitness,Cp);
popx[j]<-mutate(population,fitness,Mp);
}
j++;
}
optimal(popx[j]);
```

B. Explaination of Algorithm

By using the Control Flow Graph (CFG), we have extracted a graph that we named converged graph. Converged graph is the graph that is constructed by taking only the predicate nodes, because the path splits only when the predicate is encountered. We have represented the converged graph by an equivalent program (graphequi(srv)) and the search node as input that is the one of the leaf node. GA optimizes the test cases to satisfy the search nodes one by one. The genetic algorithm then optimizes the test cases by using selection, crossover and mutation operators. The optimized test cases are then passed through random or stratified sampling (optimal(popx[i])) to derive the optimal test suites.

In MATLAB, a proposed test data generation procedure using genetic algorithm was implemented, which selects individuals for free migration based on their fitness values and generate test data.

C. A Process Flow Graph for Test Data Generation

To generate path-oriented test data for the program under test using GA, there are six steps need to follow which is given in Fig. 1. Selected target path is the goal for GA to achieve, and an input vector Y (a test data) is treated as an individual.

i. **Acceptance of Unit Under Test (UUT)**

   Firstly we take a program or unit module as an input which is to be tested using path coverage.

ii. **Program instrumentation**

   This means inserting probes at the beginning of every block of source code to monitor program execution and collect related information

iii. **Control flow graph construction**

   Control flow graph of the program under test may be constructed manually or automatically with related tools. It helps testers to select representative target paths.

iv. **Target path selection**

In general, a program under test has too many paths to test completely. Thus, testers have to select meaningful paths as target paths.

v. **Fitness function construction**

In order to evaluate distance between the executed path and the target path, fitness function has to be constructed. It can be used either Hamming distance approach or Brach distance function approach. Here we can use both approaches according to situations.

vi. **Instrumented program execution and Test data generation**

In this phase, Instrumented units or modules are executed and Original test data are chosen from their domain at randomly or using an approach.

GA generates new test data in order to achieve the target path. Finally, suitable test data that executes along the target paths may be generated or no suitable test data may be found because of exceeding max generation [6].

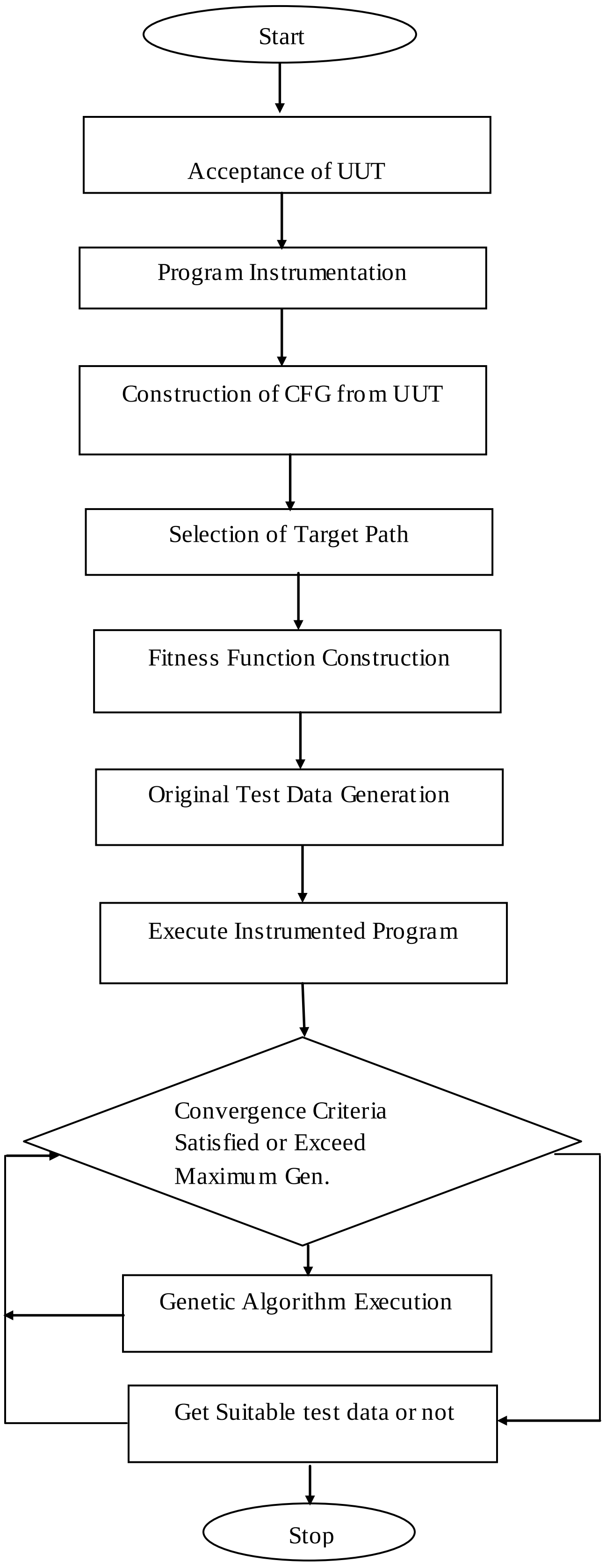
Start
Acceptance of UUT
Program Instrumentation
Construction of CFG from UUT
Selection of Target Path
Fitness Function Construction
Original Test Data Generation
Execute Instrumented Program
Convergence Criteria Satisfied or Exceed Maximum Gen.
Genetic Algorithm Execution
Get Suitable test data or not
Stop

Fig. 1: Basic Process Flow Graph

V. EXPERIMENTAL STUDY

A. Triangle Problem Code

```
TraversedPath= [];
TriangleType='Not a Triangle';
If ((X+Y>Z)&&(Y+Z>X)&&(Z+X>Y))
TraversedPath =[TraversedPath 'a'];
If ((X~=Y)&&(Y~=Z)&&(Z ~=X))
TraversedPath=[TraversedPath 'e'];
TriangleType='Scalene';
else
TraversedPath =[TraversedPath 'b'];
If(((X==Y)&&(Y~=Z))||((Y==Z)&& (Z~=X))||((Z==X)&&(X~=Y)))
TraversedPath =[TraversedPath 'f'];
TriangleType='Isosceles';
else
TraversedPath=[TraversedPath 'c'];
TriangleType='Equilateral';
end
end
else
TraversedPath =[TraversedPath 'd'];
End
```

Triangle Problem determines if three input sides can form a triangle and so what type of triangle can be formed by them.

B. Control Flow Graph of UUT and Target Path Selection

The tested program (Triangle problem) determines what kind of triangle can be formed by any three input lengths. Fig. 2 is the CFG, which consists of four paths :< d>, <ae>, <abf>, <abc>. They represent 'Not-a-triangle', 'Scalene', 'Isosceles' and 'Equilateral' respectively. According to probability theory, the path <abc> is the most difficult path to be covered in path testing. Therefore, the path <abc> is selected as the target path.

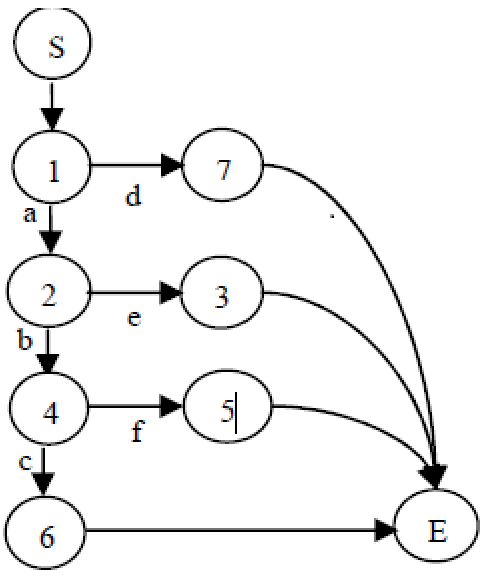

Fig. 2: Control Flow Graph of UUT

C. Parameter Setting

Generation of Test data may depend upon parameter of genetic algorithm.

Settings of SGA are as following:

- Coding: Binary string
- Length of Chromosome=15*3 = 45bits
- Population size=1500
- Rank-based fitness assignment
- Steady State Selection Strategy
- Single-point crossover probability= 0.9
- Mutation probability=0.09
- Max generation=500

Multi Population GA still requires other parameters below:

- No. of subpopulations=5
- Number of individuals per subpopulation=275
- Insertion rate=0.75
- Migration rate=0.63
- Migration period=10 generations
- Generation gap=0.9

D. Experimental Results

Table 1 show that SGA based approach may achieve the target path <abc> within 74689 test data by average in 18 seconds. However, according to probability theory, the probability of achieving the target path is 2^{-30} (that is (^{215} $*1*1$) / (^{215} $*2^{15} * 2^{15}$) where each input variable is 15bits), which means it will take random testing2 ^{30} tests to achieve the objective.

Type	Generations	Test data	Time (seconds)
MPGA	25	21472	3
SGA	79	74689	18
SGA / MPGA	3.16	3.47	6

Table 1: Average results with identical initial population

After one hundred fifty experiments respectively, average results of MPGA [12] and SGA in Table 2 are slightly higher than corresponding results in Table 1. However, proportional relation between the two algorithms is very close to corresponding results in Table 1. Moreover, average number of test data covering four paths with random initial populations.

Therefore, initial population has little influence on performance of MPGA and SGA in path oriented test data generation.

Type	Generations	Test data	Time (seconds)
MPGA	27	24421	4
SGA	87	85954	26
SGA / MPGA	3.22	3.52	6.5

Table 2: Average results with random initial population

VI. CONCLUSION

In software testing, the generation of testing data is one of the key steps which have a great effect on the automation of software testing. In this paper, we discuss an Algorithm and basic process flow that depends on the principles of genetic algorithms to generate test cases that provide good coverage in terms of the paths it tests or visits within the application. The greatest merit of genetic algorithm in program testing is its simplicity. The quality of test cases produces by genetic algorithms is higher than the quality of test cases produced by random way. The proposed algorithm gives a better result in terms of execution time and efficiency of test cases. Using a triangle problem under test, experiment results show that MPGA can generate path oriented test data more effectively and efficiently than SGA .This paper also shows that genetic algorithms are useful in reducing the time required for complex testing expressively by generating test cases for path testing.

Furthermore, we can build our Genetic Algorithm for path testing for reduce execution time & generate more suitable test cases. We can also use hybrid GA (Memetic Algorithm) for generation of test case that includes local search also.

REFERENCES

1. Myers G. J., "The Art of Software Testing", *1st Edition, John Wiley and Sons, NY, USA*, 1979.
2. Beizer B., "Software Testing Techniques", *2nd. Edition, Van Nostrand Reinhold, USA* 1990
3. Nagappan N., Williams L., Osborne J., Vouk M. and Pekka Abrahamson P., "Providing Test Quality Feedback Using Static Source Code and Automatic Test Suite Metrics", *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, Chicago*, 2005, pp85 - 94.
4. Premal I., Nirpal B., Kale K. V., "Using Genetic Algorithm for Automated Efficient Software Test Case Generation for Path Testing", *IEEE Int. J. Advanced Networking and Applications, vol. 2, pp. 911-915*, 2011.
5. Holland J.H., "Adaptation in natural and artificial system" *The University of Michigan Press*, 1975.
6. Chen1 Y., Zhong Y., Shi1 T. and Jingyong Liu, "Comparison of Two Fitness Functions for GA-based Path-Oriented Test Data Generation", *IEEE Fifth International Conference on Natural Computation*, 2009.
7. Hermadi I., Lokan C. and Sarker R., "Genetic Algorithm Based Path Testing: Challenges and Key Parameters", *Second WRI World Congress on Software Engineering*, 2010
8. Dhawan S., Kulvinder S. Handa, Kumar R., "Optimization of software testing using Genetic Algorithms" *International conference on mathematical and computational methods*, 2008
9. Michael C C, McGraw G E, Schatz M A. "Generating Software Test Data by Evolution", *IEEE Transactions on Software Engineering, vol.27, December* 2001.
10. Korel B., "Automated software test data generation" *IEEE Transactions on Software Engineering*, 16(8), August 1990.
11. Jones B. F., Sthamer H. H., Eyres D. E., "Automatic structural testing using genetic algorithms" *IEEE Software Engineering Journal, September*, 1996.

12. Yong Chen, Yong Zhong "Automatic Path-oriented Test Data Generation Using a Multi-population Genetic Algorithm" *IEEE Fourth International Conference on Natural Computation, (ICNC2008), pp. 566 – 570,* Oct. 2008