Optimization of Software Testing for Discrete Testsuite using Genetic Algorithm and Sampling Technique

Siba Prasada Tripathy
National Institute of Science & Technology
Berhampur, India

Debananda Kanhar
National Institute of Science & Technology
Berhampur, India

ABSTRACT

Though Software Testing plays a vital role to produce better quality products, but it is time consuming and incurs expenditure. The more test will be conducted, products will be finer. Hence, testing is must for software development. Another side of testing is spending much money on it because people will work rigorously to generate the test suite and executing it. As we know that, no software is bug free software; we cannot assure that the testing which has been done for particular software is sufficient. To get a cost effective testing strategy, one should go for optimization of testsuite. This paper uses heuristic algorithm with sampling techniques used to optimize the test suite. Genetic algorithm may play a major role to have a sound weight on optimization of testsuite. If we go for sampling techniques then it usually gives more optimum result.

Keywords

Genetic Algorithm, selection, crossover, mutation, sampling

1. INTRODUCTION

Software Testing is a method to discover errors by executing the software [4]. There are two method of testing, that is black box testing or functional testing and white box testing or structural testing. A huge amount of money is spent for testing. Hence, testing is done in sophistical manner. Software testing remains the primary technique used to gain consumers' confidence in the software and categorizing the test cases using stratified sampling. Genetic Algorithm (GA) provides a general-purpose search methodology, which uses principles of natural evolution. In this paper, genetic algorithm is used for generating test suite from a discrete set of test cases. Hence, this is a major challenge that implementing a genetic algorithm in software testing generates an optimized test suite. The development of techniques that will also support the automation of software testing will result in significant cost savings.

2. DEFINITIONS

2.1 Test Case:

A test case is a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [IEEE, do178b] [7].

2.2 Test Adequacy Criteria:

To ensure the testing process, an empirical technical investigation is conducted to provide the adequacy of the test cases in testing the SUT. This may be statement coverage, branch coverage, condition coverage, mutation score etc. [12, 10].

2.3 Test Optimization:

To maximize the profit of finding more bugs (mutation score) and coverage and to minimize the total number of test cases needed [9].

2.4 Mutation Testing:

Mutation Testing is done by mutating certain statements in the source code and checking if the test case is able to find the errors.

2.5 Stratified Sampling

Stratified sampling[2] and category-partition are the same things but category-partition is used in categorizing all the functions according to their cohesiveness. Stratified sampling demonstrates the samples that populate certain functional domain. The principle of counting is used to estimate total number of optimal test suites.

2.6 Testing Categories:

Testing information flow is said to be as a testing technique which specifies the strategy to select input test cases and analyze test results [11]. Different testing techniques reveal different quality aspects of a software system, and there are two major approaches of testing techniques such as functional testing and structural testing.

2.6.1 Functional Testing:

The software program or system under test (SUT) is considered as a "black box". The selection of test cases for functional testing is based on the requirements or design specifications generated from the requirements of the software entity which is under test[15]. There are some examples of expected results which are collectively called test oracles. These include requirement or design specifications with hand calculated values, and simulated results. The main attraction of the software testing is the external behavior of the software entity.

2.6.2 Structural Testing:

The software program or system under test (SUT) is considered as a "black box". The selection of test cases is based on the implementation of the software entity. The main focus of such test cases is to cause the execution of specific spots in the program or software entity, which might include specific statements, program branches or paths. Any one of coverage criteria like path coverage, branch coverage, statement coverage or condition coverage may be used to evaluate the expected result which is the main focus of structural testing.

3. OPTIMIZATION PROBLEM FORMULATION

3.1 Optimization

Utilizing the available resources as much as possible is called optimization. Optimization process is an incremental problem. At each and every evolution the solution leads to the target function [4].

3.2 Test Case Optimization

This means generating test cases that have the ability to reveal as many errors as possible from the Software Under Test (SUT) and to cover the SUT within less time and cost by selecting an effective set of few test cases from the universe of test cases. Here both mutation score (total number of seeded errors) and fitness function have to be maximized for each test case during test case generation. Selection of test cases is then done based on mutation score and coverage criterion [12, 10].

4. APPROACH USED FOR GENETIC ALGORITHMS

GA is a search technique used to find exact or approximate solutions to optimization and search problem [3]. Genetic Algorithm gives more number of options for selection of most appropriate gene which justifies Darwin's theory of survival of the fittest. It is seen that randomized exchange of structured information is there in GA among an artificial chromosome population. Using GA, surprising results have been found. A problem is defined as maximization of a function of the kind f(x1, x2, ... xm) where (x1, x2, ...,xm) are variables which have to be adjusted towards a global optimum. Three basic operators responsible for GA are
(a) Selection (b) crossover (c) mutation
Crossover performs A chromosome can be a binary string or a more elaborate data structure. Firstly, there will be a pool of chromosomes which can be randomly produced or manually created. The suitability of a chromosome depends on fitness function measured to meet a specified objective: for coverage based testing, a chromosome is fitter in case of greater coverage. Participant chromosomes which can be taken part in the evolution stage of the genetic algorithm made up by the crossover and mutation operators. Exchanging genes from two chromosomes are done by crossover operator and creates two new chromosomes. Changing of genes in a chromosome is done by the mutation operator and creates one new chromosome. [13]
A basic algorithm for a GA is as follows:
The pseudo code for GA is:

```
Initialize (population)
Evaluate (population)
While (stopping condition not satisfied) do
{
Selection (population)
Crossover (population)
Mutate (population)
Evaluate (population)
}
```

The algorithm has some iteration which will be performed until getting a solution to the problem, or it will reach to maximum number of iterations.

4.1 Encoding

Direct value encoding can be used in problems where some more complicated values such as real numbers are used. In the value encoding, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.

4.2 Selection

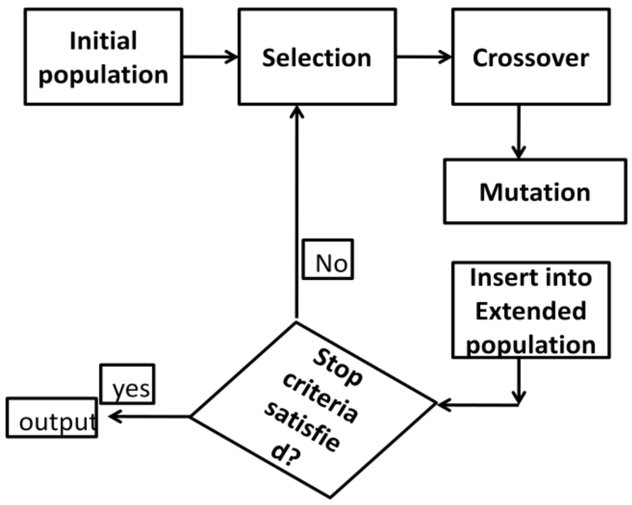
In selection process filter chromosomes are taken for better chance to survive to the next generation. The selection of better individuals is done by having a trade off and the unfit chromosomes become recessive. Rank selection criteria are used to select the individuals. Population ranking is done and then every chromosome is assigned a fitness value determined by this ranking. The worst individual will get the least fitness value and the best individual gets the best fitness value [2].

4.3 Crossover

Two crossover points are selected in two-point crossover, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again[1].

4.4 Mutation

Randomly change one or more digits in the string representing an individual.



(Figure-1: Information flow for the Genetic Algorithm process)

5. PROPOSED APPROACH

In this paper, quadratic equation is taken as the Unit Under Test (UUT). The equation defines the testsuite ranges from 0 to 100. A fitness function is generated to handle the test suite and to refine the test suite as it is known that there are four major conditions are supposed to be checked during testing. Hence, the fitness function is very important here to solve the problem. We can generate the test suite by taking 101*101*101 number of test cases.

6. PROCEDURE

Input: Quadratic equation: ax2+bx+c = 0

The theme behind the quadratic equation is to get the discrete testsuite which will help for comparing the expected output with the generated optimal output. Hence quadratic equation with four conditions is considered.

We have three variables, a, b and c
ax2+bx+c = 0

Roots are real if (b2-4ac) > 0

Roots are imaginary if (b2 – 4ac) < 0

Roots are equal if (b2 – 4ac) = 0

Roots are not equal if a = 0

Using three variables a, b, and c, an initial pool of individuals are generated which ranges from 0 to 100.
The table (see Table-1) indicates the test suite for the above problem. By taking all kind of possibilities, the testsuite is generated where each variable ranges from 0 to 100. Like this the total number of test cases is **100 × 100 ×100.**

(Table - 1: Testsuite for quadratic equation ranges 0-100)

Test case	A	B	c	output	Expected output
1	0	0	0	0	Not quadratic
--	--	--	--	--	--
100	99	0	0	0	Equal roots
101	100	0	0	0	Equal roots
102	0	1	0	1	Not quadratic
103	1	1	0	1	real
--	--	--	--	--	--
201	99	1	0	1	real
202	100	1	0	1	Real
203	0	50	0	2500	Not quadratic
302	99	50	0	2500	Real
303	100	50	0	2500	Real
304	0	0	0	0	Not quadratic
305	1	1	0	1	Real
404	100	100	0	10000	Real
405	0	0	0	0	Not quadratic
406	1	1	1	-3	Imaginary roots
505	100	100	100	-30000	Imaginary roots
506	1	1	0	1	Real
507	1	1	1	-3	Imaginary roots
606	1	1	100	-399	Imaginary roots
707	50	0	50	-10000	Imaginary roots
708	50	50	0	2500	real
709	50	50	1	2300	real
808	50	50	100	-17500	Imaginary roots
809	50	50	0	2500	real
810	50	50	1	2300	real
811	50	50	2	2100	real
808	50	50	100	-17500	Imaginary roots
--	--	--	--	--	--

6.1 Reproduction (crossover):

One point cross over or single point cross over discusses two input data are selected as potential parents by selection process exchange substring information at a random position in the data to produce two new data. According to a crossover probability pc crossover happens, with an adjustable parameter. Generate a random real number r in the range [0, 1]; for each parent. If number of fit gene is less than four then select the parent for crossover. Each pair of parents generates two new paths, called offspring. The crossover technique used is one point crossover done at the midpoint of the input bit string. In this technique, right half of the bits of one parent are swapped with the corresponding right half of the other parent.

6.2 Mutation

Bit by bit mutation is performed where every bit of chromosome has an equal chance to mutate which later changes from ‘0’ to ‘1’ or from ‘1’ to ‘0’. According to mutation probability, the mutation occurs which can be used as an adjustment parameter. To perform mutation, for each chromosome in the offspring and for each bit within the chromosome, alter the last bit of the gene, if number of fit gene is less than four.

6.3 Postmortem of GA in the problem :

Before sending the test suite to genetic algorithm, we applied the stratified sampling [2] which filtered the integer type testsuite from all positive and negative testsuite. This sampling technique helped a lot and reduced the half of the time of testing. When each combination of test case has been considered for checking the fitness, it was sent through a program which represents the fitness function. Then, it checks the fitness value and dumps into the appropriate category, either selected or rejected. After generation of testsuite, we stored these test suite in an excel sheet and then fetched to cross through GA. The testsuite becomes filtered by checking the fitness function and applying crossover and mutation if necessary. When the number of fit gene becomes four then there we stop the iteration where as the default maximum iteration is 100 in our approach. If there will be no fit gene found it will stop generating after 100th iteration. After implementation of genetic algorithm, we found in one experiment we got the fit genes in six iterations, the output is given below:

```
------------Optimization Starts--------
Iteration number :::0
The fit testcase is a = 2  b = 1  c = 1
The roots are imaginary
The fit gene is 000010,000001,000001
2 , 1 , 1 is an unfit gene due to redundancy
Iteration number :::1
2 , 0 , 2 is an unfit gene due to redundancy
2 , 3 , 2 is an unfit gene due to redundancy
Iteration number :::2
The fit testcase is a = 2  b = 3  c = 0
The roots are real
The fit gene is 000010,000011,000000
2 , 0 , 3 is an unfit gene due to redundancy
2 , 3 , 0 is an unfit gene due to redundancy
2 , 0 , 3 is an unfit gene due to redundancy
Iteration number :::3
3 , 3 , 2 is an unfit gene due to redundancy
The fit testcase is a = 0b = 3c = 2
The roots are Not Quadratic
The fit gene is 000000,000011,000010
3 , 3 , 2 is an unfit gene due to redundancy
0 , 3 , 2 is an unfit gene due to redundancy

Iteration number :::4
3 , 2 , 3 is an unfit gene due to redundancy
2 , 3 , 0 is an unfit gene due to redundancy
3 , 2 , 3 is an unfit gene due to redundancy
2 , 3 , 0 is an unfit gene due to redundancy
Iteration number :::5
3 , 2 , 0 is an unfit gene due to redundancy
2 , 3 , 3 is an unfit gene due to redundancy
3 , 2 , 0 is an unfit gene due to redundancy
2 , 3 , 3 is an unfit gene due to redundancy
Iteration number :::6
The fit testcase is a = 3  b = 0  c = 0
The roots are equal
The fit gene is 000011,000000,000000
Finished executing
```

In the above experiment, it was found that the selection of fit genes got evolved after six iterations and there were a number of genes got rejected due to redundancy. As the testing is functional testing, we have a discrete set of outputs and that are : real roots, equal roots, imaginary roots and not quadratic.

7. Results and Analysis

Total Four experiments are taken while implementation GA for a comparison and evaluating benefits of mutation.
In experiment 1, it is seen that the number of iteration is 30. The detail of implementation is given below. (See table-2)

(Table-2: GA implementation with 30 iterations)

Iteration no	Fit gene	Test Category
1	000001,000010,000011 (1,2,3)	Imaginary
2	unfit	--
3	unfit	--
4	Unfit	--
5	Unfit	--
---	--	--
17	000000,000001,000011 (0,1,3)	Not Equal
--	unfit	--
28	000001,000000,000000 (1,0,0)	Equal
29	Unfit	
30	000001,000011,000001 (1,3,1)	Real

There are 12 number of iterations generated all four fit genes in the second experiment (see Table-3). After applying more mutation the number of iteration got reduced from 30 to 12.

(Table-3: GA implementation with 12 iterations)

Iteration no	Fit gene	Test Category
1	000001,000010,000010 (1,2,2)	Imaginary
2	000001,000011,000010 (1,3,2)	Real
3	000000,000000,000011 (0,0,3)	Not Equal
4	Unfit	--
5	Unfit	--
6	Unfit	--
7	Unfit	--
8	Unfit	--
9	Unfit	--
10	Unfit	--
11	Unfit	--
12	000010,000000,000000 (2,0,0)	Equal

.
The comparison between experiment 1 and 2, it is found that after applying mutation it reduces the number iterations. The fit genes were found in iteration number 1,17,28,30 in experiment number-1 where as the fit genes we found in next experiment iteration number 1,2,3,12.
The table (see table-4) contains the fit genes after 10 numbers of iterations. Iteration number 5 to iteration number 9 generated the unfit genes.
The table (see table-5) contains the fit genes after 5 numbers of iterations. Iteration number 4 generated the unfit gene.

The comparison among experiment 3 and experiment 4 indicates that after applying more mutation it reduces the number of iterations. The fit genes were found in iteration number 1,2,3,12 in experiment number-3 where as the fit genes we found in next experiment are iteration number 1,2,4,10.

In this way the implementation of GA is done to get the discrete set of testcases for the above case study and to reduce the redundant testcases.

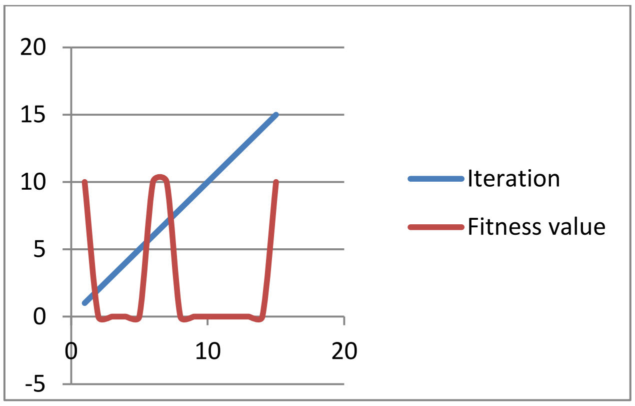
Iteration No.	Fit gene	Test Category
1	000001,000001,000011 (1,1,3)	Imaginary
2	000010,000010,000000 (2,3,1)	Real
3	unfit	--
4	000010,000000,000000 (2,0,0)	Equal
5	unfit	--
6	unfit	--
7	unfit	--
8	unfit	--
9	unfit	--
10	000000,000001,000010 (0,1,2)	Not Equal

(Table-4: GA implementation with 10 iterations)

Iteration No.	Fit gene	Test Category
1	000001,000010,000001 (1,2,1)	Equal
2	000001,000011,000011 (1,3,3)	Imaginary
3	000011,000010,000000 (3,2,0)	Real
4	000010,000010,000000 (2,2,0)	Redundant due to Real
5	000000,000010,000001 (0,2,1)	Not Equal

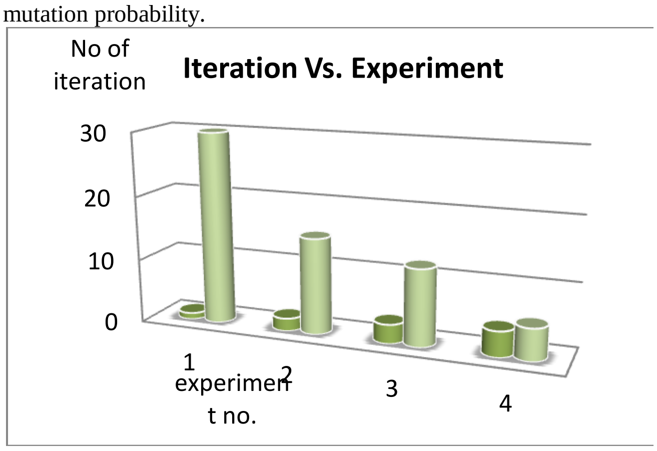
(Table-5: GA implementation with 5 iterations)

Figure (See figure-2) indicated the number of iteration with fitness value in one experiment where iteration number 1, 6,7 and 15 are having the fitness value 10 as there are two fitness value have been considered that are 0 for unfit and 10 for fit.



(Figure - 2: iterations with fitness value of the testcases used for selection of fit gene)

Figure (See Figure-3) denotes the number of experiments with number of iterations which has been decreased by applying more mutation probability.



(Figure - 3: Experiment number with the iteration number that generated the fit genes)

8. CONCLUSION

Though, Genetic algorithm is an effective optimization technique that looks for global optimal value where as the other optimization technique focuses on local optimal values, this algorithm alone is not sufficient for software testing. In some specific cases sampling techniques must be used for filtration of testcases. It is found that Genetic algorithm can generate an optimal testsuite and it is proved by taking the discrete testsuite where the optimal test suite is known to the tester. So, as GA can be used in any field, it has a vast scope to draw optimum result.

9. ACKNOWLEDGEMENT

My hearty gratitude to my guide Asst Prof. Mr. Devanand Kanhar, of NIST Brahmapur for his continuous guidance. My sincere thanks to the management of NIST, Berhampur for their support. A major literature review is done at Roland Institute of Technology, hence my sincere thanks to principal and management of Roland Institute of Technology, Berhampur. Finally I wish to pay my profound thanks to my parents who supported me during writing the paper.

10. REFERENCES

[1] Kulvinder Singh and Rakesh Kumar "Optimization of Functional Testing using Genetic Algorithms", International Journal of Innovation, Management and Technology, Vol. 1, No. 1, April 2010 ISSN: 2010-0248.

[2] Debasis Mohapatra, Prachet Bhuyan, Durga P. Mohapatra "Automated Test Case Generation and Its Optimization for Path Testing Using Genetic Algorithm and Sampling "2009 WASE International Conference on Information Engineering.

[3] K. Deb, A. Pratap, S. Agarwal, and T.Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transaction of Evolutionary Computation., vol. 6, pp. 182–197, Apr. 2002

[4] R.Pressman "Software Engineering" TMG sixth Edition page 386p-460p

[5] Venkatraman,S. Yen,G.G. "A Generic Framework for Constrained Optimization Using Genetic Algorithms" IEEE Transactions on Evolutionary Computation.

[6] Christoph C. Michael, Gary E. McGraw, Michael A. Schatz, Curtis C. Walton in their paper titled "Genetic Algorithms for Dynamic Test Data Generation" in National Science Foundation under award number DMI-9661393

[7]. Kamde, P.M.—Nandavadekar, V.D.—Pawar, R.G.: Value of Test Cases in Software Testing. IEEE International Conference on Management of Innovation and Technology, 2006, pp. 668–672

[8] Pargas, R.P.—Harrold, M. J.—Perk, R.R.: Test Data Generation Using Genetic Algorithm. Journal of Software Testing, Verification And Reliability, 1999, pp. 1–19.

[9] Desikan, S.—Ramesh, G.: Software Testing Principles and Practices. Pearson, 2002.

[10] Mathur, A.P.: Foundations of Software Testing. Pearson Education, 2008.

[11] L. Luo, "Software testing techniques technology maturation and research strategy," Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA15232, USA, Tech. Rep. 17939, 2001.

[12] K.K. Aggarwal, and Y. Singh, "A book on software engineering", New Age International (P) Ltd.; Publishers, 4835/24, Ansari Road, Daryaganj, New Delhi, 2001

.[13]Praveen Ranjan Srivastava and Tai-hoon Kim, "Application of Genetic Algorithm in Software Testing", International Journal of Software Engineering and Its Applications Vol. 3, No.4, October 2009.

[14] Sangameswar Venkatraman and Gary G. Yen,"A Generic Framework for Constrained Optimization Using Genetic Algorithms" IEEE Transaction of Evolutionary Computation, vol. 9, no Aug2005 pp.424-434.

[15] Baikuntha Narayan Biswal, Soubhagya Sankar Barpanda and Durga Prasad Mohapatra, "A Novel Approach for Optimized Test Case Generation Using Activity and Collaboration Diagram", International Journal of Computer Application (IJCA), vol. 1, no. 14, pp. 67 – 71, 2010.