A GENETIC ALGORITHM FOR REGRESSION TEST CASE PRIORITIZATION USING CODE COVERAGE

Arvinder Kaur

Associate Professor,
University School of Information Technology
G.G.S. Indraprastha University, Delhi- 110043

Shubhra Goyal

Research Student
University School of Information Technology,
G.G.S. Indraprastha University, Delhi- 110043

Abstract— Regression testing is a testing technique which is used to validate the modified software. The regression test suite is typically large and needs an intelligent method to choose those test cases which will detect maximum or all faults at the earliest. Many existing prioritization techniques arrange the test cases on the basis of code coverage with respect to older version of the modified software. In this approach, a new Genetic Algorithm to prioritize the regression test suite is introduced that will prioritize test cases on the basis of complete code coverage. The genetic algorithm would also automate the process of test case prioritization. The results representing the effectiveness of algorithms are presented with the help of an Average Percentage of Code Covered (APCC) metric.

Keywords: Genetic Algorithm; Prioritization; Regression Testing; Automation Testing.

I. INTRODUCTION

Regression testing is retesting changed segments of application system. It is performed frequently to ensure the validity of the altered software. In most of the cases, time and cost constraint is prominent; hence the whole test suite cannot be run. Thus, prioritization of the test cases becomes essential. The priority criteria can be set accordingly e.g. to increase rate of fault detection, to achieve maximum code coverage, and so on.

One of the performance goals is to run those test cases that achieve total code coverage at the earliest [9]. Here, we propose a technique that achieves 100% code coverage. The three broad categories for prioritization are Greedy algorithms, non-evolutionary algorithms and evolutionary algorithms. Evolutionary algorithms (EA) have been chosen as they are global optimization methods and scale well to higher dimensional problems. They can be easily adjusted to the problem at hand and can be change and customized.

It is interactive and meta-heuristic process that operates on a set of population. Most of the implementations of evolutionary algorithms came from any of these three basic types: Genetic Algorithm (GA), Evolutionary Programming (EP) and Evolutionary Strategies (ES). All these are strongly related but independently developed. Among evolutionary techniques, the GA, invented by John Holland in the 1960s at the University of Michigan, study the phenomenon of evolution and adaptation as it occurs in nature. They depend on the use of selection, crossover (recombination) and mutation operators [9]. Automated software testing has been considered critical for big software development organizations but is often too expensive or difficult for smaller companies to implement. This algorithm automates the process of prioritize the test suites as per the criteria given to genetic algorithm.

II. RELATED WORK

Many researchers addressed prioritization problem and proposed various techniques for it. Many techniques are used for prioritization such as Greedy algorithms for test case prioritization [13], 2-optimal algorithms [7], non-evolutionary algorithms such as goal programming method [4], logarithmic least square method [5], weighted least square method [5] and evolutionary algorithms[3]. Most frequent among all is total fault-detection technique [15].

In the test case prioritization using genetic algorithms, the prioritization criterion is based on fitness function of population and genetic operators [11]. Further, Genetic algorithm is used for network security in cryptography [8]. GA is also used in Data Mining operations [12] and Robotics [14].

III. THE GENETIC ALGORITHM

Over several years, organisms are evolving on the basis of fundamental principle "survival of fittest" to accomplish noteworthy results. In 1975, Holland employed principle of natural evolution to optimization problems and built first GA.

In GA, a population P = (c_1… cm) is formed from a set of chromosomes and each chromosome is composed of genes. The GA populates the population of chromosomes by successively replacing one population with another based on fitness function assigned to each chromosome. The strong individual is included in next population and individuals with low-fitness are eliminated from each generation. [10]. There are two main concepts in genetic algorithm viz: crossover and mutation.

A. *Crossover*

The crossing over (key operator) is process of yielding recombination of alleles via exchange of segments between pairs of chromosomes.Crossover is applied on an individual by switching one of its allele with another allele from another individual in the population. The individuals resulting from crossover are very different from their initial parents. The code below suggests an implementation of individual using crossover:

Child1 = c*parent1 + (1-c)*parent2 (1)
Child2 = (1-c)*parent1 + c*parent2 (2)

B. *Mutation*

The mutation is a process wherein one allele of gene is randomly replaced by (or modified to) another to yield new structure .It alters an individual in the population. It can regenerate all or a single allele in the selected individual. To maintain integrity, operations must be secure or the type of information an allele holds should be taken into consideration. That is, mutation must be aware of binary operations, or it must be able to deal with missing values.
A simple piece of code:
child = generateNewChild(); (3)

The optimization problems are solved by GA's recombination and replacement operators, where recombination is key operator and frequently used, whereas, replacement is optional and applied for solving optimization of problem.

IV. GENETIC ALGORITHM FOR PRIORITIZATION OF TEST CASES

The initial population is automatically generated and the evaluation of the set of candidate solution has been done with the help of genetic algorithm. The stopping criteria used in this approach is total code coverage.

A. *Flowchart*

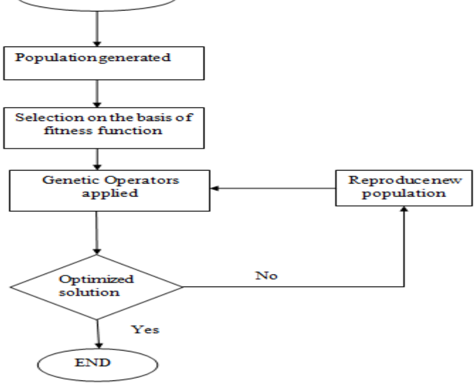

Figure 1. Flowchart of Genetic Algorithm.

B. *Algorithm*

STEP 1. Generation of initial population
Generate 'n' number of chromosomes $\{c_1, c_2 \ldots c_n\}$

STEP 2. Initialization of population
Set Test Suite= No. of chromosomes (n)

STEP 3. Fitness function criterion set
Set fitness function= total code coverage

STEP 4. Select suitable population on the basis of Fitness Function
SELECT (Best 2 chromosomes based on fitness function)

STEP 5. Genetic Operators Applied

```
Do for selected Chromosome(s)
    While (all conditions are covered)
        Do crossover
        Do mutation
        Remove Duplicacy
    EndWhile
EndFor
```

STEP 6. Optimization of solution cheked.

```
If (solution!= feasible)
    Goto STEP 5
    Else END.
```

C. Algorithm Explained:

In GA, the optimal solution is searched on the basis of desired population which further can be replaced with the new set of population. The generation and initialization of test cases (population) is done according to the problem. The two fitness criterion chosen are maximum fault covered in minimum execution time and total code coverage. Henceforth, this fitness function will help in selecting suitable population for problem. Further, the genetic operations are performed. Firstly, crossover, which recombines two individuals. Secondly, mutation, which randomly swaps the individuals. Thirdly, the redundant individuals are removed. Finally, the solution is checked for optimization. If solution is not optimized, then, the new population is reproduced and genetic operators are applied.

D. Problem Definition:

Prioritization based on total code coverage is done by structural testing. This is achieved through path testing which is a group of test techniques based on selecting a set of test paths through the program. Flow graph generation is the first step of path testing. Then decision to decision (DD) path graph is generated form flow graph. It is used to find independent paths. An independent path is any path through the DD path graph that introduces at least one new set of processing statements or new conditions. Therefore, we need to execute all independent paths at least once during path testing. The example is explained below:

The example taken for code coverage is the triangle problem which takes the three sides (a positive integer in the range of 0 to 100) of the triangle as input and gives the output as scalene, isosceles, equilateral, not a triangle and invalid inputs according to the input[11]. The test cases, conditions and independent paths covered by them are shown in the table 4.

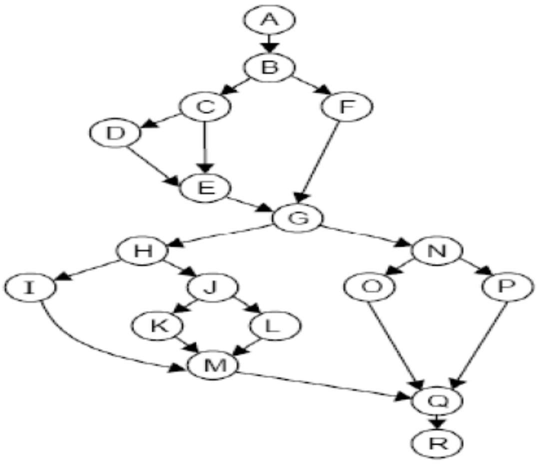

Figure2. DD Path graph of Triangle problem.

Following are the independent paths of the triangle problem:

i. ABFGNPQR
ii. ABCDEGHJKMQR
iii. ABCDEGHIMQR
iv. ABCDEGNOQR
v. ABCEGNPQR
vi. ABCDEGHJLMQR
vii. ABFGNOQR

INPUTS

Table I. Test cases with inputs and outputs

TEST CASE NO.	INDEPENDENT PATH	CONDITION COVERED
1	abfgnpqr	3
2	abcdeghjkmqr	5
3	abcdeghjkmqr	5
4	abcdeghimqr	4
5	abcdeghjkmqr	5
6	abcdegnoqr	4
7	abcegnpqr	4
8	abfgnpqr	3
9	abcdeghjlmqr	5
10	abcdeghjkmqr	5
11	abcdeghjkmqr	5
12	abcdeghjkmqr	5
13	abfgnoqr	3
14	abcegnpqr	4
15	abfgnpqr	3
16	abcdeghjkmqr	5
17	abcdeghjlmqr	5
18	abcdeghimqr	4
19	abcdegnoqr	4
20	abcegnpqr	4

Step 1: Test case Generation

We are applying the foremost step of our algorithm by selecting the randomized test suites. The number of test cases is the number of chromosomes generated. This is explained in the table 5 given below.

Table II. Execution of Example, G-Genes, C- Chromosome

CHROMOSOMES	**OBSERVATIONS FOR 1st ITERATION**											
C1	G	T1	T5	T6	T9	T4	T7	T11	T17	T20		
C2	G	T2	T4	T9	T12	T16	T18	T7	T8	T10	T6	T20
C3	G	T3	T15	T17	T19	T6	T20	T4	T13	T5	T14	
C4	G	T4	T6	T17	T9	T12	T10	T20	T1	T3	T12	T7
C5	G	T5	T8	T12	T15	T19	T20	T14	T6	T11	T10	T7
C6	G	T6	T12	T1	T20	T16	T2	T19	T4			
C7	G	T7	T9	T13	T15	T14	T18	T19	T20	T4		
C8	G	T8	T10	T14	T20	T12	T4	T9	T2	T6	T7	
C9	G	T9	T19	T12	T8	T1	T5	T4	T10	T17	T20	
C10	G	T10	T12	T14	T16	T18	T20	T6	T2	T4	T13	
C11	G	T11	T13	T15	T20	T19	T1	T18	T17	T3	T4	
C12	G	T12	T14	T16	T18	T4	T20	T9	T7			
C13	G	T13	T17	T20	T19	T6	T14	T12	T6	T7	T4	
C14	G	T14	T4	T19	T6	T8	T12	T5	T20	T3	T1	
C15	G	T15	T3	T7	T9	T4	T1	T18	T10	T14	T20	
C16	G	T16	T10	T19	T20	T5	T11	T8	T14	T4	T12	
C17	G	T17	T5	T1	T16	T7	T6	T17	T12	T20	T2	T4
C18	G	T18	T19	T15	T17	T5	T20	T8	T9	T2	T4	
C19	G	T19	T4	T13	T14	T3	T6	T1	T7	T3	T20	
C20	G	T20	T4	T1	T2	T8	T3	T18	T6	T9	T16	T19

Step 2: Select an input for GA algorithm based on the fitness function

The fitness function in this is selecting minimum test cases to cover all the independents paths with minimum test cases. Two test suites of eight test cases and two test suites of nine test cases are selected as per the fitness function. The crossover is applied on test suites of similar length. The 3-point crossover is applied on two sets of test suites.
Two offsprings are formed on applying crossover. One of the two offspring covers all the independent paths while the other does not cover all the independent paths and hence that offspring is discarded.
Thus based on this fitness function, we get two iterations with test suite {T6,T12,T1,T20,T16,T2, T19 and T4} and {T12,T14,T16,T18,T4,T20,T9,and T7}.

Step 3: Apply Genetic Algorithm on test suite of nine test cases does not yield optimized result. Thus, we apply on test suite of eight test cases to further prioritize.

Step 3.1. Do crossover

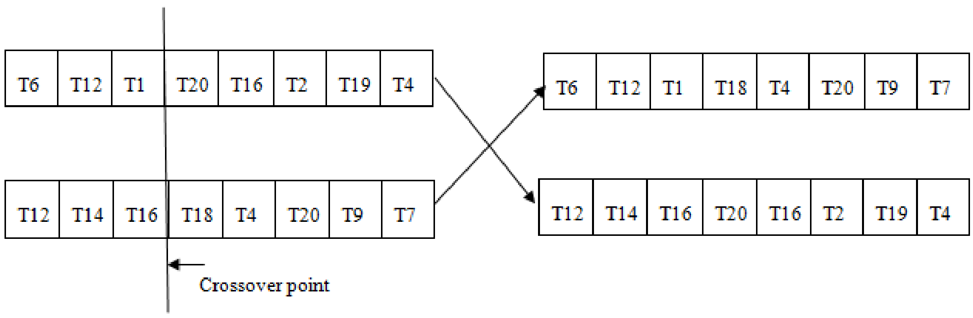

Figure 3. Applying crossover on the test suite

Thus the test suites we get after crossover as two offsprings are

T6	T12	T1	T18	T4	T20	T9	T7

And

T12	T14	T16	T20	T16	T2	T19	T4

The first offspring i.e. test suite obtained after crossover covers all the independent paths and that test suite is selected for mutation. The test suite selected is as:

T6	T12	T1	T18	T4	T20	T9	T7

Step 3.2 Do mutation on one of the best offspring and the process shown is as:

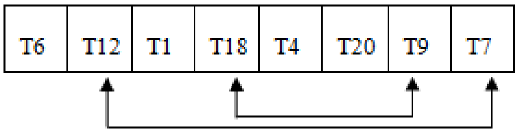

Figure 4. Applying mutation on the resulted test suite

The result obtained after applying mutation is

T6	T7	T1	T9	T4	T20	T18	T12

Step 3.3 removing the duplicates from the test suites

Thus, removing the duplicate test cases T18 and T12, we get the final test suite which covers all the seven independent paths as below and this is the final result.

T6	T7	T1	T9	T4	T20

E. Algorithm Analysis:

To analyze code coverage based testing effectively the Average Percentage of Condition Coverage (APCC)[16] approach has been used where average percentage of test suite to be executed with respect to average condition's covered.Table-9 shows various orders for Example-3 and corresponding APCC is plotted in figure 9. In this paper, Example-3has used APCC. The APCC is given as:

$$APCC = \frac{1-TC1+TC2+\dots\dots+TCm}{nm} + 1/2n \qquad (4)$$

Where, T = test suite been executed
n = number of test cases,
m = number of conditions to be covered,
TC_i = First test case covering ith condition.

Table III shows the final percentage calculated from APCC for example. Table IV shows proposed technique is comparable with optimum order for the examples.

Table III. Representing APFD and APCC values.

Technique	Example APCC %
No Order	90
Random Order	89.2
Reverse Order	89.2
Optimal Order	91.7
GA Order	88.3

Table IV. Order of test cases for various prioritization approaches for example of maximum code coverage.

No Order	Reverse Order	Random Order	Optimum Order	GA Order
T1	T20	T6	T9	T6
T2	T19	T8	T7	T8
T3	T18	T10	T4	T10
T4	T17	T5	T2	T2
T5	T16	T4	T18	T20
T6	T15	T2	T1	T15
T7	T14	T1	T13	T13
T8	T13	T9	T3	T9
T9	T12	T7	T5	T7
T10	T11	T3	T8	T1
T11	T10	T16	T6	T3
T12	T9	T13	T10	T12
T13	T8	T19	T19	T16
T14	T7	T20	T11	T14
T15	T6	T17	T15	T11
T16	T5	T19	T12	T17
T17	T4	T12	T16	T19
T18	T3	T18	T14	T5
T19	T2	T11	T17	T18
T20	T1	T14	T20	T4

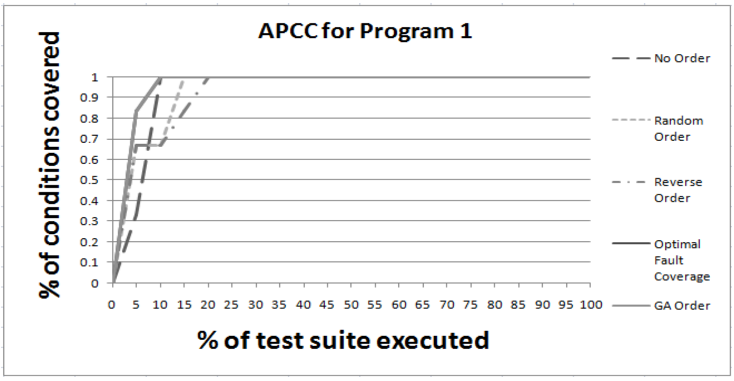

Figure 5. APCC chart for example 1 of maximum code coverage.

F. Threats to Validity:

The GA algorithm proposed here has been executed and following areas have been detected as threat to validity.

1. The optimal result depends on observing the final result.
2. The algorithm has been tested on less number of programs. More analysis is needed.

V. Application of the Proposed Approach

This approach may be used by the software practitioners to reduce the time and effort required for prioritization of test cases in the test suite. The proposed approach may lead to greater savings of time and effort in larger and complex projects as compared to smaller ones. Using GA approach, software practitioners can effectively select & prioritize test cases from a test suite, with minimum execution time. Hence, the proposed algorithm may prove to be useful in real-life situations.

VI. CONCLUSION

The algorithm has been proposed to prioritize test cases using Genetic Algorithm. Here, different prioritization approaches have been analyzed, namely: total fault coverage with in time constrained environment and amount of code coverage on different examples and their finite solution obtained, respectively. Through Genetic Algorithm technique, an approach has been identified to pull out suitable population, which was further formulated by GA operations to make it more flexible and efficient. The elaborations of results are shown with the help of APCC values. The APCC has been calculated for example for code coverage testing to evaluate the usefulness of the proposed algorithm.

The algorithm is solved manually and is a step towards Test Automation. In future an automation tool is to be developed to implement the proposed algorithm which can solve large number of test cases in efficient time.

REFERENCES

[1] 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.

[2] K.K. Aggarwal, Y. Singh, and A. Kaur, "Code coverage based technique for prioritizing test cases for regression testing", ACM SIGSOFT Software Engineering Notes, New York, NY, USA Volume 29 Issue 5. 2004.

[3] N.M.A. AL-Salami, "Evolutionary Algorithm Definition", American J. of Engineering and Applied Sciences 2 (4): Science Publications, pp.789-795, 2009.

[4] N. Byson, "A goal programming method for generating priorities vectors", Journal of Operational Research Society, Palgrave Macmillan Ltd.,Houndmills, Basingstoke, Hampshire, RG21 6XS, England,pp. 641-648,1995.

[5] A.T.W, Chu, R.E. Kalaba, and K. Springam, "A comparison of two methods for determining the weights of belonging to fuzzy sets", Journal of Optimization Theory and Applications, US,volume 27, pg. 531-541, 1979.

[6] G. Crawford, and C. William, "A note on the analysis of subjective judgment matrices", Journal of Mathematical Psychology, Elsevier Inc,volume 29,pg. 387-405, 1985.

[7] G. Duggal, and B. Suri, "Understanding Regression Testing Techniques",2008.

[8] Gorodilov, and V. Morozenko, "Genetic Algorithm for finding the key's length and cryptanalysis of the permutation cipher", International Journal information theories and application, Institute of Information Theories and Applications FOI ITHEA vol.15, 2008.

[9] J.H. Holland, "Adaptation in Natural and Artificial Systems", Journal of Computer and System Sciences, MIT Press, Cambridge, MA, USA, volume 64, 1992.

[10] R. W. Kristen, "Prioritizing Regression Test Suites for Time-Constrained Execution Using a Genetic Algorithm", Department of Computer Science, Allegheny College, 2005.

[11] R. Krishnamoorthi, S.A. Sahaaya , and A. Mary, "Regression Test Suite Prioritization using Genetic Algorithms", International Journal of Hybrid Information Technology, 2009.Kamble, "Incremental Clustering in Data Mining using Genetic Algorithm", International Journal of Computer Theory and engineering, vol. 2, no. 3, 2010.

[12] Z. Li, M. Harman, and R.M. Hierons, "Search algorithms for regression test case prioritization", IEEE Transactions On Software Engineering, San Francisco, CA, USA, volume 33, no.4, 2007.

[13] D.K. Pratihar, K. Deb, and A. Ghosh, "A genetic-fuzzy approach for mobile robot navigation among moving obstacles", International Journal of Approximate Reasoning, Elsevier Inc, Volume 20, Issue 2, pg. 145-172, 1999.

[14] G. Rothermel, R.H. Untch, C. Chu, and M. H Jean, "Test Case Prioritization: An Empirical Study, In Proceedings of the International Conference on Software Maintenance", IEEE Computer Society Washington, DC, USA, pg. 179-188, 1999.

[15] L. Shanmugapriya, A. Askarunisa, and N. Ramaraj, "Cost and Coverage Metrics for Measuring the Effectiveness of Test Case

[16] Prioritization Techniques", INFOCOMP Journal of Computer Science, pp. 1-10, 2009.

[17] K. R. Walcott, G. M. Kapfhammer, M. L. Soffa, and R. S. Roos, "Time- aware test suite prioritization, International Symposium on Software Testing and Analysis", Association in Computing Machinery, Portland, Maine USA, pp. 1-19, 20 July, 2006.

AUTHORS PROFILE

ArvinderKaur

Dr. ArvinderKaur is an Associate Professor with the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. She obtained her doctorate from Guru Gobind Singh Indraprastha University and her master's degree in computer science from Thapar Institute of Engineering and Technology. Prior to joining the school, she worked with B.R. Ambedkar Regional Engineering College, Jalandhar and Thapar Institute of Engineering and Technology. She is a recipient of the Career Award for Young Teachers from the All India Council of Technical Education, India. Her research interests include software engineering, object-oriented software engineering, software metrics, software quality, software project management, and software testing. She also is a lifetime member of ISTE and CSI. Kaur has published 60 research papers in national and international journals and conferences.

Shubhra Goyal

She received B.Tech from Guru Gobind Singh Indraprastha University in 2009. She is currently pursuing M.Tech from Guru Gobind Singh Indraprastha University. Her area of interest is Software Engineering.