Test Case Minimization By Generating Requirement Based Mathematical Equations

Mamta Santosh[1]

Rajvir Singh[2]

1. Research Scholar, CSE, DCRUST, Murthal, Haryana: 131039(India).

Abstract:- Test suite minimization techniques are used to remove redundant and obsolete test cases from test suite. Test case minimization approach can be considered as an optimization problem. This paper presents a practical approach towards test case minimization. In this paper in the proposed approach, the test case-requirement matrix is mapped to form mathematical equation(s) which in turn are optimized subjecting to some constraints using the genetic algorithm. The best individuals are selected for generating reduced test suite.

1. INTRODUCTION

Software testing is an essential phase in software engineering to determine the quality of software product and hence software reliability. Test cases are run on the software to find errors. A test case is a set of variables or conditions under which a tester determines whether an application or program is working correctly or not. A test case is defined in IEEE standard[2] as: "A set of test inputs, execution, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement". A test suite is a set of test cases that can satisfy all the requirements. It becomes inexpensive to run all the test cases as this may lead to high testing cost in terms of execution time, resources etc. So there is need to do test case minimization to reduce testing effort.

The quality of a test suite is measured –through following four factors: 1) its fault coverage, 2) its code coverage, 3) its size, and 4) the number of faults detected by the most effective test contained in it [2]. Before generating test cases, test requirements should be specifically determined i.e. whether the statement, functional, branch coverage testing need to be performed. The problem of test case minimization is to derive a representative set that covers all the requirements as the original suite and redundant test cases are removed.

The problem of selecting a representative set of test cases that provides the desired testing coverage of a program or part of a program is stated as follows[1]:

Given: A test suite TS, a list of testing requirements $r_1, r_2, \ldots r_n$, that must be tested to provide the desired testing coverage of the program, and a list of subsets of TS, T_1, $T_{2,\ldots}$ T_n,, one associated with each of the r_i's such that any one of the test cases t_j belonging to T_i can be used to test the requirement r_i.

Problem: Find a representative set of test cases t_j that will satisfy all of the r_i's.

Several techniques have been proposed for minimizing test suites. Heuristic based approach selects test cases based upon the strategies of

essential, redundant and 1-to-1 redundant strategies. A test case is said to be essential if the requirement satisfied by the test case cannot be satisfied by any other test case. A test case is said to be redundant if the requirement satisfied can be satisfied by another test case.

This paper has proposed an approach which formulates test case minimization problem as optimization problem. An optimization problem can be stated as:

Finding value of variables that minimize or maximize the objective function while satisfying the constraints [15].

Quality of software testing can be determined by checking whether all the requirements are satisfied or not.

2. MOTIVATION

To reduce testing cost, test case minimization is necessary. Genetic algorithm can be used for minimization as it is robust and provide optimized result. For applying genetic algorithm, the test case – requirements relationships need to be transformed in mathematical model expressed in form of functions and parameters that optimize the model. In literature no such technique has been presented which can formulate test case minimization problem as mathematical expression. This paper presents an approach to make such formulation. Later genetic algorithm is applied for optimization.

3. RELATED WORK

Harrold defined heuristic H in which he selected test cases on the basis of their essentialness. Chen and Lau [3] introduced GRE approach which firstly selects essential test cases, then discards 1-to-1 redundant test cases and then apply greedy approach on remaining test cases until all the requirements are satisfied. Tallam and Gupta [4] proposed inspired greedy algorithm for test suite reduction which is based on Formal Concept Analysis of the relation between test cases and testing requirements. Jeffrey and Gupta [5] proposed test suite reduction with selective redundancy to decrease the loss of fault detection effectiveness. For this, they modified HGS heuristic with selective redundancy in which they selectively added those test cases that provide additional def-use coverage at the time they become redundant with respect to branch coverage.

Chen, Zhang and Xu [6] proposed degraded ILP Approach for Test Suite Reduction. They produced a lower bound of minimum test suite and feasible solution near lower bound was searched. . Lin and Huang [7] analyzed test suite reduction with enhanced tie-breaking technique. They used some additional coverage criteria for breaking the tie among two test cases which was different than traditional approaches where a random decision was made. Yoo and Harman [8] proposed multi-objective test suite reduction. They utilized a hybrid, multi-objective genetic algorithm. This algorithm combined the efficient approximation of the greedy approach with the genetic algorithm to produce high quality Pareto fronts. Nachiyappan, Vimaladevi and SelvaLakshmi [9] proposed genetic algorithm for test suite reduction. The model built the initial population based on test history. The fitness value of test cases was calculated based on the block based coverage value and execution time of the test cases. The test cases with optimum fitness were selected. Test cases which violate fitness constraint were rejected. Huang, Liu et al. [10] proposed improved quantum genetic algorithm for reducing test suites. In their approach

chromosome is encoded with quantum bit as information bit. Their technique reduced testing costs greatly and improved test efficiency. You and Lu [11] proposed Genetic Algorithm for the Time-Aware Regression Testing Reduction Problem. Time criteria were added with the genetic algorithm. The algorithm removes all redundant test cases and also decreases running time. All these techniques performed well in some factors but it is harder to know which one is best among all these. In proposed approach requirements and test cases are handled by using control dependency graph and FEP respectively. A common prioritization approach groups requirements into three categories – high, medium and low priority. Other prioritization methods [13] include Analytical Hierarchy Process (AHP), hierarchy AHP, Minimal Spanning Tree, Bubblesort, Binary Search Tree, Priority Groups etc. Test cases can be handled by their essentialness, rate of code coverage, rate of fault detection, cost efficiency.

4. PROPOSED WORK

In proposed work, test case- requirement matrix is mapped to mathematical model (optimization function) subjecting to some constraints. Then genetic algorithm is applied for optimizing the function. The best individuals are selected to form reduced test suite. This can be summarized as (Figure 1) :

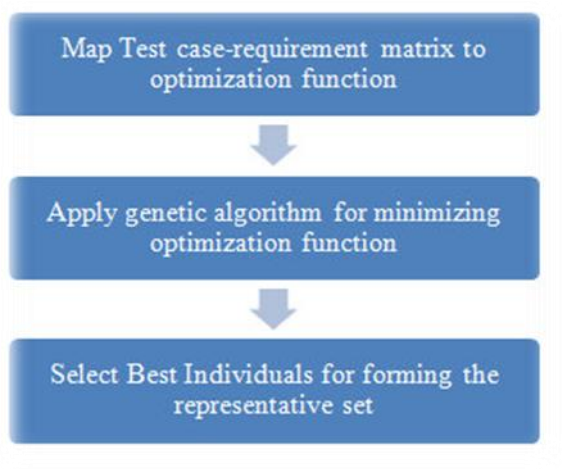

Figure 1

1. Test case – Requirement matrix to Optimization function mapping

Test cases and requirements need to be assigned some values as coefficients to form optimization function. Variables corresponding to test cases can be used for forming the minimization function. For optimal minimization requirements and test cases need to be prioritized. Let us consider an example program given below:

1. if(a>o and b>0 and c>0) {
2. If (a <(b+c) and b<(a+c) and c <(a+b)) {
3. Print triangle.
4. If(a==b and a==c)
5. Print equilateral.
6. Else if (a==b or b==c or a==c)
7. Print isosceles.
8. Else print scalene }
9. Print not a triangle}
10. Exit.

Here the numbers represent the statement number and testing goal is to cover all the statements except exit. The test cases considered for above program are:

Test Case	Input	Output
TC1	(4,1,1)	Not a triangle
TC2	(3,4,5)	Scalene
TC3	(3,3,3)	Equilateral
TC4	(2,2,3)	Isosceles
TC5	(0,0,0)	Not a triangle

The test case - requirement matrix has been formed according to requirements. If the test case satisfies any requirement then the corresponding cell contains 'X'.

Test case – Requirement Matrix is shown below in Figure 2 :

TC/REQ	TC1	TC2	TC3	TC4	TC5
r1	X	X	X	X	X
r2	X	X	X	X	
r3		X	X	X	
r4		X	X	X	
r5		X	X		
r6		X		X	
r7				X	
r8		X			
r9	X				X

Figure 2

a) Prioritizing Requirements

In proposed approach control flow graph is used. In control flow graph, statements are represented by nodes and edges with direction shows the direction of control flow in the program.

CFG of above program can be drawn as (Figure 3) :

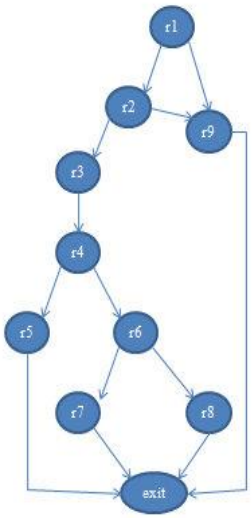

Figure 3

At each level of graph, priority value is assigned to requirements. Higher level requirements possess higher priorities. Requirements can be prioritized as (Figure 4) :

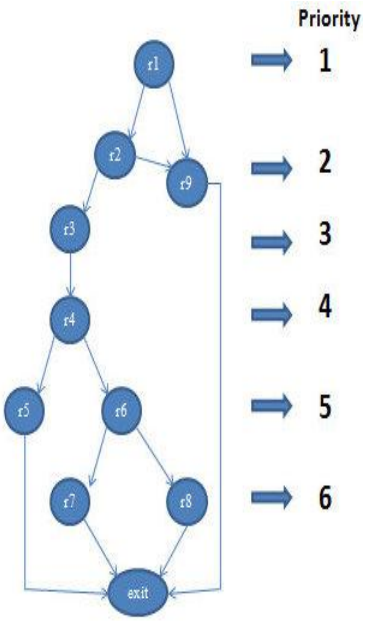

Figure 4

Priorities can be assigned to requirements on the basis of level of dependence:

Requirement	Priority
R1- highest –	1
R2, R9…….	2
R3……..…..	3
R4……..……..	4
R5,R6……..…….	5
R7,R8……..…….	6

b) Handling Test Cases

Test cases can be handled by their Fault Exposing Potential (FEP). FEP can be calculated by injecting faults in the program. A mutant can be generated by randomly changing operator in the statement. For e..g. increment operator(++) changed to decrement operator(--) or ">" changed to "<". Fault Exposing Potential of test case can be determined by the capability of test case to reveal mutants. So FEP value of test case is the ratio of number of mutant detected by test case to the total number of mutants injected.

$$FEP(t_i) = \frac{|\text{mutants exposed by } t_i|}{|\text{Total mutants}|}$$

Here t_i is the text case whose FEP is calculated.

A mutant is exposed by test case if mutated program give different output than original program.

Calculating FEP :

We introduce mutants in the program. In given example mutant (shown by red fonts) introduced are –

1. if(a>o and b>0 and c>0) {
2. If (a" >"(b+c) and b<(a+c) and c <(a+b)) {
3. Print triangle.
4. If(a"!="b and a==c)
5. Print equilateral.
6. Else if (a"!="b or b==c or a==c)
7. Print isosceles.
8. Else print scalene }
9. Print not a triangle}

Total mutants inserted are =3

Mutants exposed by test cases are (2,1,2,2,0) respectively .

FEP of test case = (mutant exposed)/ (total mutants)

FEP of TC1 = 2/3 = 0.66

FEP of TC2 = 1/3 = 0.33

FEP of TC3 = 2/3 = 0.66

FEP of TC4 = 2/3 = 0.66

FEP of TC5 = 0/3 = 0.00

These FEP values can be used in corresponding columns of test cases in the matrix.

c) Testing Cost

Software testing Cost can be calculated in terms of number of resources used execution time etc. For an efficient developing, the testing cost should be minimum. If we can have upper limit on testing cost,

then we can have an approximation on testing effort. In proposed approach we are considering execution time only. The constraints can be evaluated in respect to execution time.

Assumption : A test case takes one unit time to satisfy one requirement.

d) Test case –Requirement matrix formation

Test Case - Requirement matrix can be represented by a two-dimensional table in which test can be taken on one dimension and requirements at another. In this approach each test case and requirement has been assigned coefficients. FEP as coefficient has been assigned to test cases and requirement has been assigned their priority as coefficients. Each cell of matrix corresponds to the multiplication of "Priority of requirement" and "Fault Exposing Potential of test case" of the corresponding requirement and test case as in Table 1 .

	FEP	0.66	0.33	0.66	0.66	0
TC/REQ	Priority	TC1	TC2	TC3	TC4	TC5
r1	1	1*0.66	1*0.33	1*0.66	1*0.66	1*0
r2	2	2*0.66	2*0.33	2*0.66	2*0.66	
r3	3		3*0.33	3*0.66	3*0.66	
r4	4		4*0.33	4*0.66	4*0.66	
r5	5		5*0.33	5*0.66		
r6	5		5*0.33		5*0.66	
r7	6				6*0.66	
r8	6		6*0.33			
r9	2	2*0.66				2*0
Time		3	7	5	6	2

Table 1

e) Fitness Function Formation

Variables corresponding to test cases have been considered. Fitness function can be formed by summing up the test case – requirement matrix as shown in Table 2. The constraints can be determined by using random number of requirement variables.

	FEP	0.66	0.33	0.66	0.66	0
	Variable	t_1	t_2	t_3	t_4	t_5
TC/REQ	Priority	TC1	TC2	TC3	TC4	TC5
r1	1	0.66	0.33	0.66	0.66	0
r2	2	1.32	0.66	1.32	1.32	
r3	3		0.99	1.98	1.98	
r4	4		1.32	2.64	2.64	
r5	5		1.65	3.30		
r6	5		1.65		3.30	
r7	6				3.96	
r8	6		1.98			
r9	2	1.32				0
	Sum	3.3(t_1)	8.58 (t_2)	9.9 (t_3)	13.86 (t_4)	0 (t_5)

Table 2

Fitness function by summing up above table:

$$\mathbf{Z = 3.3\ (t_1) + 8.58\ (t_2) + 9.9\ (t_3) + 13.86\ (t_4) + 0\ (t_5)}$$

Constraints can be formed by grouping the test case variables. Test case variables can be grouped on the basis of number of requirements they satisfy. In above example test case TC1 and TC5 satisfy 2 to 3 requirements. Test cases TC2, TC3 and TC5 satisfy 5 to 7 requirements.

From t_1 and t_5 :

1. $3.3(t_1) + 0\ (t_5) <= 5$

$$\mathbf{3.3(t_1) + 0\ (t_5) - 5 <= 0}$$

$3.3(t_1) - 0\ (t_5) <= 1$

$$\mathbf{3.3(t_1) - 0\ (t_5) - 1 <= 0}$$

2. From t_2, t_3 and t_4 :-

$8.58\ (t_2) + 9.9\ (t_3) <= 12$

$8.58\ (t_2) + 9.9\ (t_3) - 12 <= 0$

$8.58\ (t_2) + 13.86\ (t_4) <= 13$

$8.58\ (t_2) + 13.86\ (t_4) - 13 <= 0$

$9.9\ (t_3) + 13.86\ (t_4) <= 11$

$9.9\ (t_3) + 13.86\ (t_4) - 11 <= 0$

5. USE OF GENETIC ALGORITHM

Genetic algorithm are based upon the Darwin's theory of survival of fittest. Genetic algorithm are adaptive heuristic search algorithm that provide global optimum results. Three factors, on which genetic algorithm focus are objective function, set of variables that affect objective function and set of constraints. Genetic algorithm optimizes based upon the three basic principles: **Selection, Crossover, Mutation**. In selection, we select parents on the basis of their fitness. Then cross over the parents to form new offsprings. then offspring are mutated at some random points. Genetic algorithm works as follows :

a) Generate initial population randomly.

b) Initialize each individual with their fitness value.

c) Select two individuals and cross over them to generate new offspring.

d) Mutate the offspring at random point

e) Repeat until maximum number of iteration are completed or function is optimized.

On the above optimization function, genetic algorithm is applied using MATLAB. It returns the individual with their best fitness value for the above given fitness function as shown in Fig.5 and Fig. 6.

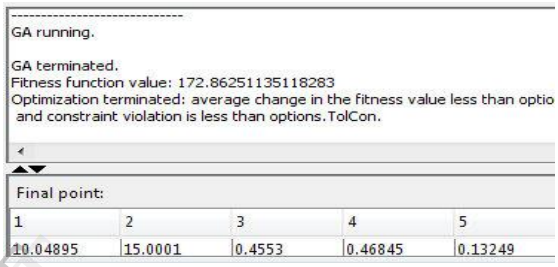

Figure 5.

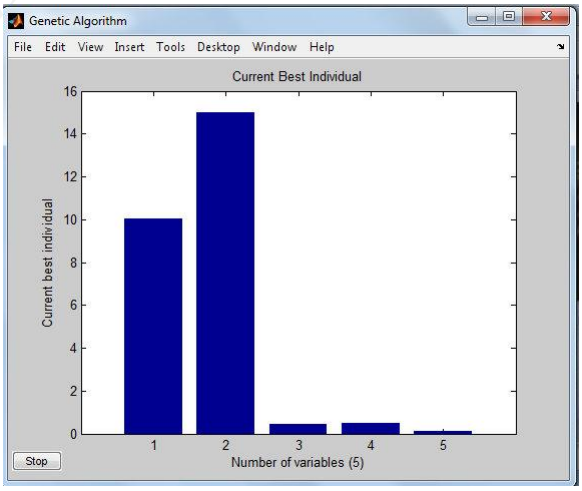

Figure 6.

Forming Representative set

The best individuals determined by the genetic algorithm can be used for forming the representative set. We select the test cases until all the requirements are satisfied, and add test cases to representative set.

In above example the test cases are best in the order TC2, TC1, TC4, TC3 and TC4 (decreasing order). Firstly we select the requirements satisfied by test case TC2, requirements satisfied are : (r1, r2, r3, r4, r5, r6 and r8). Next we select TC1, now requirements that have been satisfied are: (r1, r2 and r9). Next best individual is TC4. The requirements satisfied yet are (r1, r2, r3, r4, r5, r6, r7, r8 and r9) . All the the requirements have been satisfied and representative set is (TC1, TC2 , TC3).

6. CONCLUSION AND FUTURE WORK

This paper have discussed an approach for formulating the test case minimization problem as mathematical model subjecting to constraints. For this requirements are assigned priorities and test cases are assigned with their Fault Exposing Potential. The fitness function is calculated by summing up the test case- -requirement matrix. Execution time has been considered for testing cost. Genetic algorithm has been applied for optimization. Execution time has been considered for testing cost. This approach reduced the test suite size and covered all the requirements.

Future work may include considering the penalty function for overlapping in test case – requirement matrix.

8. REFERENCES

[1] Mary Jean, Harrold Rajiv Gupta ,Mary Lou Soffa "A Methodology for Controlling the Size of a Test Suite" CH2921-5/90/0000/0302 © 1990 IEEE.

[2] Saif-ur-Rehman Khan, Aamer Nadeem, "TestFilter: A Statement-Coverage Based Test Case Reduction Technique", 1-4244-0794-X/06/$20.00 ©2006 IEEE

[3] T.Y. Chen, M.F. Lau, "A new heuristic for test suite reduction," Information and Software Technology Conference, IEEE,40, (1998), 347-354.

[4] Sriraman Tallam, Neelam Gupta, "Concept Analysis Inspired Greedy Algorithm for Test Suite minimization," 2005, ACM :1595932399/05/0009.

[5] Dennis Jeffrey, Neelam Gupta, "Test Suite Reduction with Selective Redundancy", Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), 1063-6773/05 $20.00 © 2005 IEEE.

[6] Zhenyu Chen, Xiaofang Zhang and Baowen Xu, "Degraded ILP Approach for Test Suite Reduction", National Natural Science Foundation of China (60425206, 60773104, 60403016, 60633010).

[7] Jun-Wei Lin, Chin-Yu Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques", Information and Software Technology Journal, 0950-5849/$, 2008 Elsevier

[8] Shin Yoo , Mark Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimization", The Journal of Systems and Software 83 (2010) 689–701.

[9] S.Nachiyappan,A.Vimaladevi, C.B. Selva Lakshmi, "An Evolutionary Algorithm for Regression Test Suite Reduction", Proceedings of the International Conference on Communication and Computational Intelligence – 2010, pp.503-508.

[10] Yi-kun ZHANG, Ji -ceng LIU, Ying-an CUI, Xin-hong EI, Ming-hui ZHANG, "An Improved Quantum Genetic Algorithm for Test Suite Reduction", IEEE proceedings 978-1-4244-8728-8/11/$26.00 ©2011 IEEE.

[11] Liang You Yansheng Lu, "A Genetic Algorithm For The Time-Aware Regression Testing Reduction Problem", 2012, 8th International Conference on Natural Computation (ICNC 2012), ©2012 IEEE.

[12] Karl E. Wiegers, " Software Requirements", MicroSoft Press.

[13] Joachim Karlsson, Claes Wohlin, Bjorn Regnellc, An evaluation of methods for prioritizing software requirements, "Information and Software Technology 39", (1998) 939–947.

[14] MJ Harrold," Dependency Directed Graphs", 1993, doi=10.1.1.126.2554.

[15] R.C. Chakarborty, "Fundamentals of Genetic Algorithms", June 01, 2010, AI course, Lecture 39-40 notes.

ABOUT THE AUTHORS

1. **Mamta Santosh** is Research Scholar in the area of Software Testing at Deenbandhu Chhotu Ram University of Science & Technology (DCRUST) (A State Govt. University), Murthal: 131039 (India).

2.

Rajvir Singh is an Assistant Professor in Computer Science & Engineering Department , Deenbandhu Chhotu Ram University of Science & Technology (DCRUST) (A State Govt. University), Murthal: 131039 (India) since 5th Feb., 2010 to till date. Before joining DCRUST he has served in Delhi College of Engineering / Delhi Technological University, Delhi: 110042 (India) from 19th July, 2007 to 5th Feb., 2010 as Contractual Lecturer (Computer Engineering). He has completed his B.E.(CSE) degree in 2003 from MDU, Rohtak, Haryana:124001(India) and M.Tech. in Computer Engineering in 2007 from YMCA Institute of Engineering / YMCA University of Science & Technology, Faridabad, Haryana, India. He is pursuing Ph.D.(CSE)(Part-Time) from DCRUST, Murthal:131039(India). He have qualified various national level tests like GATE-2003, UGC-NET-2009(June), UGC-NET-2012(June), Research Aptitude Test (RAT)-2010 conducted by Jamia Millia Islamia, New Delhi-110025(India). His areas of interest are Software Testing, Software Engineering, Data Base Management Systems, Computer Graphics, Theory of Automata Computation and Artificial Intelligence.