International Journal of Advanced Technology & Engineering Research (IJATER)
www.ijater.com

TEST SUIT REDUCTION BY FINDING COST OPTIMAL REPRESENTATIVE SET

Sudhir Kumar Mohapatra, Research Scholar, SOA University, Bhubaneswar, Odisha , India
Srinivas Prasad, Dept. of Computer Science &Engineering, Gandhi Institute for Technological Advancement, Bhubaneswar, Odisha, India
Bimal Prasad kar, Dept. of Computer Science &Engineering, GIET,Ghangapatna, Bhubaneswar, Odisha, India

Abstract

Software testing is one of the important stages of software development. In software development, developers always depend on testing to reveal bugs. In the maintenance stage test suite size grow because of integration of new technique. Addition of new technique force to create new test case which increase the size of test suite. In regression testing new test case may be added to the test suite during the whole testing process. These additions of test cases create possibility of presence of redundant test cases. Due to limitation of time and resource, reduction techniques should be used to identify and remove them. Research shows that a subset of the test case in a suit may still satisfy all the test objectives which is called as representative set. Redundant test case increase the execution cost of the test suite, in spite of NP-completeness of the problem there are few good reduction techniques have been available. In this paper the previous technique proposed [17] is improved to find out cost optimal representative set.

Keywords: Genetic Algorithm; Software testing; Test suite reduction; Representative set;

1. INTRODUCTION

Software testing and retesting is done frequently during the software development lifecycle and in particular in regression testing. In regression testing software grows and evolves, that create new test cases and added them to a test suite to exercise the latest changes to the software[18]. Over many versions of the development of the software, test cases in the test suite can be redundant .The redundant test case may in respect to the testing requirements for which they were generated, because these requirements are now also satisfied by new test cases in the test suite that were newly added to cover changes in the later versions of software. Due to limitation of time and resource for retesting the software every time before a new version is release, it is really important to search for techniques that ensure manageable test suits size by periodically removing redundant test cases. This process is called *test suite minimization*. The test suite minimization problem [1] can be formally stated as follows:

Given. A test suite T of test cases $\{t_1,t2,t3,\ldots..,tm\}$, a set of testing requirements {r1,r2,r3….,rn} that must be satisfied to provide the desired test coverage of the program, and subsets {T1,T2,..,Tn} of T, one associated with each of the ris such that any one of the tests tj belonging to Ti satisfies ri.

Problem. Find a minimal cardinality subset of T that exercises all ris exercised by the unminimized test suite T.

The r_i's can represent either all of the program's test case requirements or those requirements related to program modifications. A representative set of test cases that satisfies the r_i's must contain at least one test case from each T_i. Such a set is called a hitting set of the group of sets Tl, T2, . . . , T. A maximum reduction is achieved by finding the smallest representative set of test cases. However, this subset of the test suite is the minimum cardinality hitting set of the T,'s and the problem of finding the minimum cardinality hitting set is NP-complete [2]. Therefore, since we are unaware of any approximate solution to the problem, we develop a heuristic [3,4] to find a representative set that approximates the minimum cardinality hitting set.

The development team if able to find out redundant test case and eliminate them from the test case then the test suite size can be reduced. while finding the representative set the team must ensure that all test requirements are satisfied by the reduced test suite, to make testing more efficient. That is, given the original test suite T={t1, t2, t3, ..., tn} and a set of test requirements R={r1, r2, r3, ..., rm}, the goal is to find a subset of the test suite T, denoted by a representative set RS, to satisfy all the test requirements satisfied by T. The process of finding the representative set is called test suite reduction [5], [6].

The organization of this paper is as follows. In section 2 we have specified the Existing Test Case Reduction Techniques. In section 3 an algorithm based on the genetic algorithm for test case reduction is proposed and discuss. The proposed algorithm is discussed in details followed by its implementation in section 4. In last section the findings of the paper are summarized.

2.RELATED WORK

The Greedy algorithm [9,10] removes the test case continuously. The algorithm stop when a representative set i.e RS which covers the entire requirement is derived. In Chen and Lau [11] algorithm choose all important test case first then apply greedy algorithm over the remaining test case for rest of test case selection from that. In [5] Jeffrey and Gupta produce representative set for test suite reduction using selective redundancy. Harrold, Gupta and Soffa [1] find representative test cases for each subset and include them in the representative set. In [14] the authors use irreplaceability to evaluate the importance of tests and present an algorithm that ultimately produces reduced test suites with a substantially decrease in the execution cost. Using genetic algorithm in paper [13, 15] the authors are able to minimize test case which cover the entire requirement that can be covered by all the test cases. In [17] Prasad and Mohapatra has proposed a genetic algorithm technique to find representative set.

3. MODIFIED ALGORITHM TO FIND COST OPTIMAL REPRESENTATIVE SET

In our previous paper a genetic algorithm based algorithm to find representative set is proposed. The algorithm is further modified to find all the representative set and among all possible representative set we choose that set whose cost is minimum. Like our previous algorithm the algorithm needs a test requirement matrix. Test requirement matrix (TR) is a two dimensional 0-1 matrix of size (m * n). The test suite T={ t1, t2, t3 …..,tm} is represented in row and the requirement R={r1, r2,…..,rn} is represented in the column. That is each row of the matrix represent requirements fulfill by a particular test case. Entry into the TR matrix is determined by

$$TR(i,j) = \begin{cases} 0 & if\ ti\ \ cannot\ satisfy\ \ rj \\ 1 & if\ ti\ satisfies\ rj \end{cases}$$

In table no1 a test suite of four test case and their six requirements are given. Each test case is representing in row where as the requirement fulfilled by the test case are marked as 1 in the requirement column otherwise 0. The test case cost in terms of execution time is given on the last column.

Table no 1: An example of test case, requirements and it cost

Test case	Requirements to be satisfied						Cost
No	r1	r2	r3	r4	r5	r6	
t1	1	1	1	0	0	0	2
t2	0	1	1	1	1	0	5
t3	1	0	0	0	0	1	2
t4	0	0	0	1	1	0	2
t5	1	0	1	0	0	0	1

From Table no 1 the following TR matrix is derived

$$TR = \begin{pmatrix} 1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,0 \\ 1\,0\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1 \end{pmatrix}$$

As for the 0-1 matrix with m rows and n columns, it is essential to select a subset of rows to cover all of the columns in the matrix with minimal cost. Suppose the vector element represents the row i in the vector x is selected and xi=0 means not, therefore, the set coverage problem can be represented as standard optimization problem:

$$Min\ z(x)=\sum_{i=0}^{n} C_i X_i$$

$$s.t \sum_{i=0}^{n} a_{ij}\, x_i \quad i \geq 1\ ,i=1,2,3,4,\ldots\ldots$$

(Ensure that every column is covered by at least one row)

$$xj \in \{0,1\}\ , j=1,\ 2,\ 3,\ldots\ldots$$

The test suite reduction problem is converted to set coverage problem, and then converted to standard optimization problem. It is an optimization algorithm that can use genetic algorithm to solve this reduction problem. The GA based algorithm is presented in figure1. After generating the entire representative set by the genetic algorithm process, next job is to find the one with minimum cost. From the set of RS choose the RS whose cost is minimum.

Algorithm Optimal Representative Set
Input *T*: the set of test cases
R: the set of requirements
S: the relation between *T* and *R*, *S*={(*t*, *r*)| *t* satisfies *r*, *t* ϵ *T*, and *r* ϵ *R*}
rs_i: representative set i rs_i ϵ RS
RS: set of representative set
Output : Optimal representative set of *T*
Begin
RS = { };
i 1
while (*no new rs is generated*)
{
Using GA generate a rs_i.
RS=RS U rs_i
i← i+1**}**
Calculate cost of each rs
return optimal *RS*;
end

FIGURE 1: Optimal representative set generation algorithm process.

3.1 Initial Population

Each chromosome of the initial population represents a set of test case i.e a test suite. The initial population is built up randomly using the test case pool. We use permutation encoding for encoding the chromosomes. Each chromosome contains a set of test case as given in fig 1. The initial population also store the cost of each chromosome.

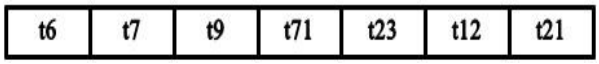

FIGURE 2: Chromosome using permutation encoding.

3.2 Selection

We use rank selection to select the chromosome to go to the next epoch. Elitism is used as test show that best population are selected.

3.3 Crossover

After the chromosomes are selected we applied single point crossover with crossover probability of 0.5 to generate new child from the selected parent.

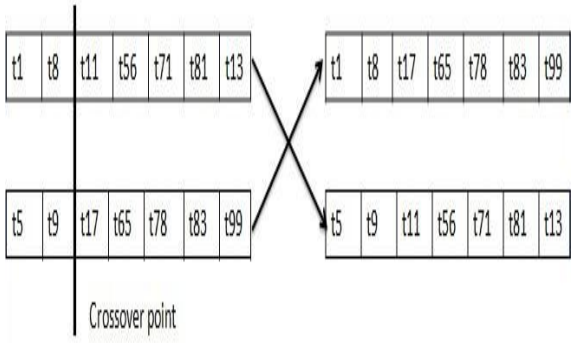

FIGURE 3: Single Point Crossover.

Let's take this example, where P1 and P2 are two individuals represented as:
P1 = <T1; T3; T6; T4> and P2 = <T2; T3; T5; T9; T4>.
If 1 is chosen, P1 and P2 could be crossed over after the first locus in each to produce two off springs as P1 =<T1; T3; T5; T9; T4> and P2=<T2; T3; T6; T4>. A crossover selection process is depicted in Fig 2.

3.4 Mutation

Mutation is used to replace the duplicate test case present in the test suite. For duplicate test case the algorithm randomly select a test case from the existing set that are not included in the chromosome with a mutation a probability of 0.1.

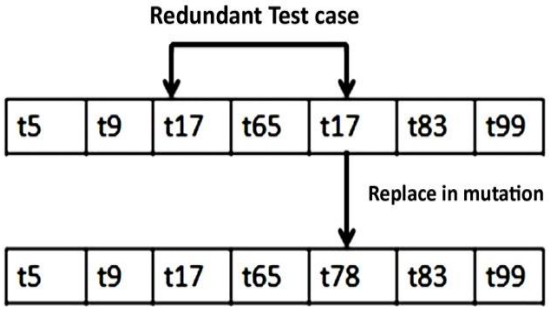

FIGURE 3: Mutation Operation.

3.5 Fitness value

The fitness value of each chromosome is calculated by performing and operation among all the requirement sets of individual test case.

Then fitness the result is converted into a percentage which denotes how much percentage of requirements is covered by the chromosome. This percentage is calculated using equation no 1.

$$F(x)=\frac{\text{No of requirment fullfill}}{\text{Total no of requirment}} \text{ X } 100 \quad (1)$$

F(x) is fitness of chromosome x. The following example gives a clear picture about how it works.

Using the TR matrix, initial population of the algorithm is generated. The algorithm first generate test suite of size 2,3,4… . The fitness is calculated for these test suite by performing OR operation of the requirements. For test suite T={t2,t4}, fitness value will be

t2={0 1 1 1 1 0}
t4={0 0 1 0 0 1}
OR {0 1 1 1 1 1}

So for the said test suite no of requirement not fulfilled is=1. Total no of requirement =6. Its fitness is
=(1/6*100) =83.33%

4. EMPIRICAL STUDIES

In order to verify our test suite reduction we take the TR matrix derived from TABLE1.

$$TR = \begin{pmatrix} 1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,0 \\ 1\,0\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1 \end{pmatrix}$$

From this we select initial population with chromosome length $l \geq 2, 3, 4\ldots$ m where m is the total no of test case present. In our TR matrix no of test case is 5. The algorithm in each iteration chooses population of size n X l where n is the population length. In every iteration GA is applied over the population. In any iteration if the fitness of one or more chromosome is 100% our algorithm stops. Out of all the chromosome produced by the algorithm we choose that chromosome whose cost is minimum as representative set.

For example by taking population size=5, Pc=0.6, Pm=0.2 from the above TR matrix, we get the following result.

Iteration # 1

l=2

Randomly choose 5 chromosomes of length 2 and calculate their fitness.

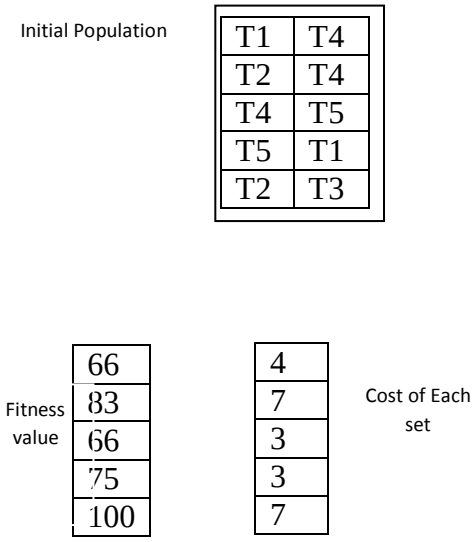

FIGURE 4: Initial population with fitness value of our example.

The above example derived representative set {T2,T3} and its cost is 7 . The algorithm further find RS={T1,T3,T4} with cost 6. The RS {T1,T3,T4} is chosen as RS in spite of its length. The test suite T={T2,T3} gives 100% fitness value that's why it is the representative set(RS) of T={T1,T2,T3,T4,T5}. Hence our algorithm stops after 1st iteration. For the above example in iteration#1 no cross over or mutation operation of GA needed. In this case the representative set is derived in 1 epoch. Otherwise we have to go for a fixed no of epoch in iteration#1. In the next iteration chromosome length 2 will be increased to 3 and again GA will be applied. This process will continue till RS is produce.

The algorithm is implemented in the working platform MetLab. After getting RS, the test case are run using an environment of JUnit ,Ant and Eclipse Emma using IDE Eclipse.

We use three JAVA program for our study: one is a 'STACK' program, and the other is a 'LIB' program. STACK consists of 91 and LIB consists of 123 blocks, and either of them are divided and instrumented by our test tool JUnit. For STACK, we have 71 test cases in the pool , and for LIB, we have 38 test cases in the pool. From these test pools, 19 randomly sized, randomly generated test suites, for each subject program, are extracted. The test suites for STACK range in size from 5 to 40 test cases, in coverage 65% to 95% and in test execution cost (mainly considering the runtime) from 149 to 2398 seconds. The average coverage is 90%. The test suites from LIB range in size from 5 to 21 test cases, in coverage 60% to 98% and in test execution cost from 87 to 1984 seconds. The average coverage is 89%. We also execute the algorithm for Jdepend which show promising result.

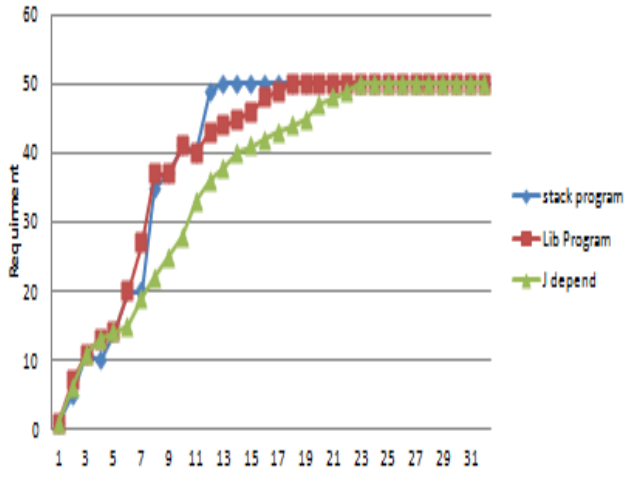

FIGURE 5: Comparison of 3 JAVA programs for deriving their representative set

In FIGURE 5 it can be clearly visualize that after certain no of test case all new addition to the test suite never increase the requirement already covered. For this three programs are selected with each having 50 requirements. Using these

program the GA algorithm give promising result. The test case size almost reduces to half.

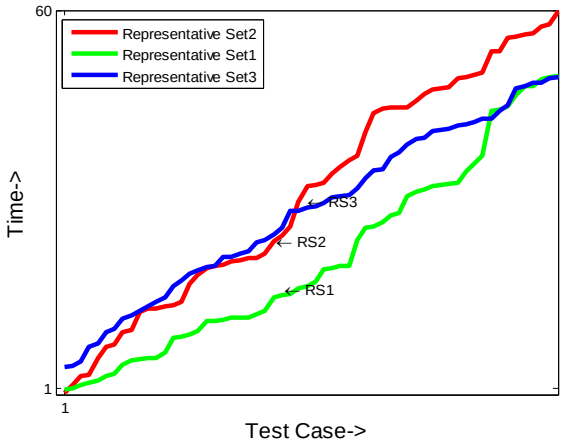

FIGURE 6: Execution time comparison of different RS

In FIGURE 6 represents execution cost comparison of different RS derived by our algorithm. It is clearly show the difference between execution time of all the representative set. We will choose the one with minimum cost.

5. CONCLUSION

In this paper our previous algorithm is modified for finding a represented set whose execution time is minimum. It finds out representative set of the test case from the given set of test case. It uses a simple GA method to reduce the test case in regression testing. Moreover, the generated test suite is minimized greatly. Therefore it can reduce test cost of regression testing and improve the efficiency of the software with the optimized test suite.

REFERENCES

[1] M.J. Harrold, R. Gupta, and M.L. Soffa, "A Methodology for Controlling the Size of a Test Suite," ACM Trans. Software Eng. And Methodology, vol. 2, no. 3, pp. 270-285, July 1993.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, second ed. MIT Press, Sept. 2001.

[3] GUPTA, R. A reconfigurable LIW architecture and its compiler. Tech. Rep. 87-3. Dept. Computer Science, Univ. Pittsburgh, Pittsburgh, Pa., 1987.

[4] GumA, R., AND SOFFA, M. L. Compile-time techniques for improving scalar access performance in parallel memories. IEEE Trans. Parallel and Distributed Systems 2, 2 (Apr.1991), 138-148.

[5] D. Jeffrey and N. Gupta, "Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction," IEEE Trans. on Software Engineering, Vol. 33, No. 2, pp. 108-123, February 2007.

[6] J. W. Lin and C. Y. Huang, "Analysis of Test Suite Reduction with Enhanced Tie-Breaking Techniques," Information and Software Technology, Vol. 51, No. 4, pp. 679-690, April 2009.

[7] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman and Company, 1979.

[8] R. M. Karp, "Reducibility among Combinatorial Problems," Complexity of Computer Computations, Plenum Press, pp. 85-103, 1972.

[9] V. Chvatal, "A Greedy Heuristic for the Set-Covering Problem," Mathematics Operations Research, Vol. 4, No. 3, pp. 233-235, August 1979.

[10] S. Yoo and M. Harman, "Regression Testing Minimization, Selection and Prioritization: a Survey," Software Testing, Verification and Reliability, Vol. 22, No. 2, March 2012.

[11] T. Y. Chen and M. F. Lau, "A New Heuristic for Test Suite Reduction," Information and Software Technology, Vol. 40, No. 5-6, pp. 347-354, July 1998.

[12] J. A. Jones and M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage," IEEE Trans. on Software Engineering, Vol. 29 No. 3, pp. 195-209, March 2003.

[13] Ma, X.y., He, Z.f., Sheng, B.k., Ye, C.q.: "A genetic algorithm for test-suite reduction". In: Proc. the International Conference on Systems, Man and Cybernetics, pp. 133–139, October 2005

[14] Chu-Ti Lin, Kai-Wei Tang, Cheng-Ding Chen, and Gregory M. Kapfhammer. "Reducing the Cost of Regression Testing by Identifying Irreplaceable Test Cases" . In Proc. Of the 6th ICGEC '12.

[15] Y Zhang, J Liu, Y Cui, X Hei , "An improved quantum genetic algorithm for test suite reduction ",IEEE International Conference on Computer Science and Automation Engineering (CSAE), 2011

[16] Dan Hao ,Tao Xie ,Lu Zhang ,XiaoyinWang , Jiasu Sun , Hong Mei, "Test input reduction for result inspection to facilitate fault localization" ,Automated Software Engineering Volume 17, Issue 1 , pp 5-31, 2010 – Springer

[17] S.K.Mohapatra, S Prasad , "Minimizing Test Cases to Reduce the Cost of Regression Testing" Proceedings of the 8th INDIACom; INDIACom-2014

[18] S.K.Mohapatra, S Prasad , "Evolutionary search algorithm for Test Case Prioritization " 2013 International Conference on Machine Intelligence Research and Advancement