Cost Aware Test Suite Reduction Algorithm for Regression Testing

C.P.Indumathi
Assistant Professor
Department of IT
Anna University - BIT Campus,
Trichy
inducp@gmail.com

S.Madhumathi
PG Scholar
M.E - Computer Science and Engineering
AnnaUniversity- BIT Campus,
Trichy
madhumathisekar94@gmail.com

***Abstract*—Regression testing is the process that a recent code change has not adversely affect the existing features. The re-running of all the test cases during regression testing is very expensive as it requires huge time and resources. Test case prioritization techniques are to schedule the test cases in accordance with some criteria such that important test cases are executed with that given period. This study presents test case prioritization using genetic algorithm and their effectiveness is measured using APFD. Then the prioritized test cases are reduced. Test suite reduction techniques aim at identifying and eliminating redundant test cases from test suites in order to reduce the total number of test cases to execute, thereby improving the efficiency of the software testing activity. Our aim is to reduce the cost by reducing the number of test suite after prioritization. MFTS algorithm is used to reduce the given test suite with maximum coverage and it improves the rate of fault detection effectiveness.**

Keywords— Regression testing, Genetic Algorithm, APFD, MFTS algorithm.

I. INTRODUCTION

Software testing is the item to detect the differences between existing and required conditions and to evaluate the features of the software item [1]. Testing is the process of executing a program or application with the intend of finding software bugs and to verify that the software product is fit for use. Software testing helps in completing the software application or product against business and user necessities. it's important to own good test coverage in order to test the software application completely [3].

Every time a new module is added or modified to a software program after integration testing, the behavior and the control logic of the software system changes. These changes may create problems for the existing quality tested earlier. Regression testing is the re running of some test cases to verify whether the existing functionalities are operating properly or not. The scope of regression testing will increase as there's any new module added to the software system and no modifications or bugs occurred [2]. Regression testing focuses on finding enhancement after a major code change has occurred. Common methods of regression testing include rerunning previous set of test cases and checking whether previously fixed faults have re-emerged.

The genetic algorithm is used to prioritize the test cases. The genetic algorithm is the process for prioritizing the given test cases by using the certain operations like selection, crossover, mutation. The basic techniques of GAs are designed to simulate processes in natural systems necessary for evolution [5]. The prioritization results in the fittest individuals dominating over the weaker once.

Test suite reduction based on MFTS (Maximum Frequent Test Set) algorithm [6]. The MFTS algorithm selects maximum frequent test cases in a test suite. This algorithm uses dynamic minimum support to select maximum frequent test cases that ensures requirement coverage of the subject program. This performance and effectiveness of the proposed reduction algorithm will be increased based on test suite size, requirement coverage, fault detection density and execution time.

II. RELATED WORK

This section presents background information and how this research work relates to software testing

Xue et al [19] proposes test suite reduction approaches are based on evolutionary computation. These approaches normally present a mathematical model for the test suite reduction problem and transform it into a linear integer-programming form. Also, there is no guarantee to find optimal test cases in finite amount of time.

Weighted Set Covering (WSC) techniques can be used to resolve the test suite reduction, a novel approach, called modified greedy algorithm, based on the Weighted Set Covering problem has been proposed by Shengwei et al [15]. This work focuses on reduction of the test suite for a student retrieval navigation model.

Wong et al [18] and Horgan et al [6] conducted empirical studies which showed that there is a higher correlation between the block coverage of a test suite and the fault detection effectiveness of the test suite. It further compares the correlation between test suite size and test effectiveness.

Rauf et al [13] proposed a genetic algorithm based technique for coverage analysis of GUI testing. This approach was subjected to extensive testing and the results showed enhanced coverage as the number of generations were increased.

978-1-5090-4257-9/17/$31.00 ©2017 IEEE

Dennis Jeffrey et al [8] proposed a new approach intended to use additional coverage information to particularly keep some test cases additionally in the reduced suite that are redundant regarding the testing criteria used for minimization.

Zuang et al [20] proposed a new strategy to select test cases and also adapt their idea to manage online test case prioritization to support more statistical significant execution information.

III. SYSTEM DESCRIPTION

This study presents prioritization of test cases based on genetic algorithm shown in Fig.3.1. This algorithm is mainly used for prioritization by using the given operators like selection, crossover, mutation. Every generation consists of a population that are analogous to the chromosomes. The best individuals are executed first than the weakest individuals.

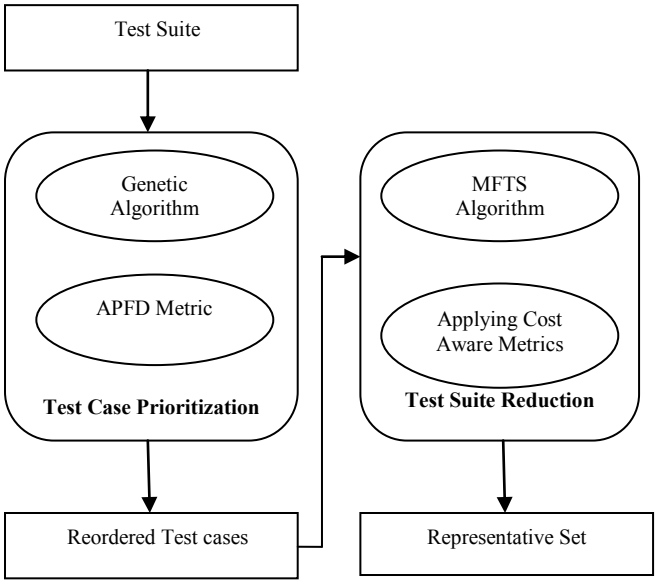

Fig 3.1 Architecture for Test case Prioritization and Reduction

A. Selection

This is the techniques of selecting possible solution to reproduce the next generation of solution. The selected best ones should endure and create new offspring.

B. Crossover(Reproduction)

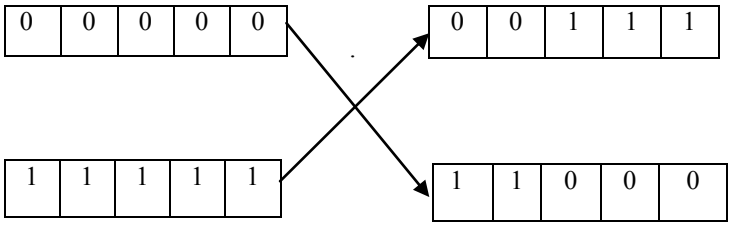

Fig 3.2.1 Crossover

Two individuals are chosen from the population using selection operator. Crossover which represents relationship between individuals. Two crossover point is selected and that the first crossover point is copied from one parent, the other part is copied from the first to the second crossover point. The two new offspring created from this recombining put into the next generation of population. By recombining portions of good individuals, this process is likely to create even better individuals as shown in Fig 3.2.

C. Mutation

Mutation which introduces random modifications. The order of the genes in a section of a chromosome is reversed. Selected bits are inverted.

A	B	C	D	E	F	G	H

Fig.3.3 Before Mutation

A	D	E	D	E	F	G	H

Fig 3.4 After Mutation

Algorithm TestCase Prioritization

Input:

T: Test Suite

P: Population size

g: Number of generation

cp: Crossover probability

mp: Mutation probability

Output:

$T_{greatest}$: A test order which has greatest fitness value in population in final generation

Acquire test cost list, *tc*, detected fault list, *fl*, and fault severity list, *fsl*, from Historical Information Repository

$P_i \leftarrow genPopulation(T, p, fl, fsl)$

For i=1 to g

$F_i \leftarrow evaFitness(P_i, tc, fl, fsl)$

$P_{i+1} \leftarrow addTwoBest(F_i, P_i)$

For j=3 to p

$Parent_1 \leftarrow randSelectParent(P_i)$

$Parent_2 \leftarrow randSelectParent(Pi)$

$Child_1, child_2 \leftarrow crossover(parent_1 parent_2, cp)$

$Child_1 \leftarrow mutation(child_1, mp)$

$Child_2 \leftarrow mutation(child_2, mp)$

$P_{i-1} \leftarrow addchildren(child_1, mp)$

$F_{g+1} \leftarrow evafitness(P_{g+1}, tc, fl, fsl)$

$T_{greatest} \leftarrow selectBestChild(F_{g+1}, P_{g+1})$

Return $T_{greatest}$

Fig 3.5 Genetic Algorithm

The genetic algorithm is applied to the example program and then prioritize the test cases using algorithm.

$T_1 = \{ t_1, t_2 \}$

$T_2 = \{ t_1, t_2, t_3, t_4 \}$

$T_3 = \{ t_3 \}$

$T_4 = \{ t_2, t_4 \}$

$T_5 = \{ t_3, t_4 \}$

$T_6 = \{ t_3, t_4 \}$

$T_7 = \{t_3 \}$

$T_8 = \{t_4\}$

This table which consists of test cases and the chromosomes with their respective values denoted by 0's and 1's. It is equated based on the genetic algorithm which is shown in Fig. 3.1.

	t_1	t_2	t_3	t_4
c[1]	1	1	0	0
c[2]	1	1	1	1
c[3]	0	0	1	0
c[4]	0	1	0	1
c[5]	0	0	1	1
c[6]	0	0	1	1
c[7]	0	0	1	0
c[8]	0	0	0	1

f.obj(x) = [10- (a+2b+3c+4d)] ---- (3.1)

f.obj(1) = 10 – [1+2(1)]

f.obj(1) = 7

f.obj(2) = 0

f.obj(3) = 7

f.obj(4) = 4

f.obj(5) = 3

f.obj(6) = 3

f.obj(7) = 7

f.obj(8) = 6

F[n] = 1/f.obj(n) ---- (3.2)

F[1] = 1/8 = 0.125

F[2] = 1

F[3] = 0.125

F[4] = 0.2

F[5] = 0.25

F[6] = 0.25

F[7] = 0.125

F[8] = 0.142

Total = 2.21

The prioritization based on fitness function is calculated by using the following equation

P[n] = F[n]/Total ---- (3.3)

P[1] = 0.125/2.21 = 0.056.

P[2] = 0.45

P[3] = 0.056

P[4] = 0.09

P[5] = 0.11

P[6] = 0.11

P[7] = 0.056

P[8] = 0.064

The prioritized order is T_2, T_5, T_6, T_4, T_8, T_1, T_3, T_7.Based on the fitness value we can prioritize the test cases. The greatest value will prioritize first and then lower values are to the next.

IV. PROPOSED MFTS ALGORITHM

The proposed algorithm uses terminologies in particular to determine test set size(cardinality), test case cardinality(Global test case array) and common test case pattern among test sets. Selecting frequent test sets based on support value to generate an optimal test suite is the focus of this proposed algorithm.

- MFTS Initialization
- MFTS Mining

A. MFTS Initialization

The test suite reduction using MFTS algorithm begins with the initialization of the associated tables and vectors. Each test set T_i represents a requirement r_i. The cardinality of each test set, Ti is determined as in equation. Similarity the cardinality of each individual test case tj in all the test sets Ti is placed in the array TCi and computed. The test sets Ti is sorted in descending order of cardinality of each individual test case tj in all test sets Ti are identified and the value of global test case array(GTC) is computed as in equation. Finally the min_Support representing the minimum cardinality is computed.

$$\sum_{m}^{j-1} T_j \qquad \text{---- (4.1)}$$

B. MFTS Mining

After initialization of MFTS variabls: tables and vectors, Algorithm begins by retrieving each test set, T_i from MFTS_Source table and placing it in the MFTS table for frequent test set mining. Then the support(T_i) for each test set

is set as '1' when it is placed in MFTS table. This is followed by recomputing the value of GTC . Within a loop the test sets in MFTS table are checked if they are not disjoint. If any test set is not disjoint the corresponding support value is incremented by '1'. Then the support value is determined if it is greater than or equal to min_support. If any test set is maximal frequent, the corresponding test set T_i is placed in the table T_{rs}. The occurrence of the test set T_i is placed in the table T_{rs} is removed from the table MFTS. After a test set T_i is added to the MFTS table if any elements in, GTC becomes '0' then the corresponding test case t_i is infrequent. The occurrence of that test case (s) is pruned from all the test sets in MFTS table.

$$GTC = GTC - TC_i \quad \text{----- (4.2)}$$

Algorithm: MFTS Initialization

Input: T_i

Output: MFTS_Source table, Global vector GTC, Vector TC_i

Begin

{

//construct the source table

Create a table MFTS_Source and place all T_is

//consider each test set

For each T_i do

//Determine the cardinality of each test set

Compute

$$\sum_{m}^{j-1} T_j$$

//Determine the count of the individual unique test cases for the test set considered

$$TC_i = \sum_{i=1}^{n} Card(t_1), \sum_{i=1}^{n} Card(t_2), \ldots \sum_{i=1}^{n} Card(t_m)$$

//place the test set in the source table along with its details in the descending order of cardinality

//compute the count of each individual test case in all test sets

$$GTC_i = \sum_{j=1}^{m} Card(t_1), \sum_{j=1}^{m} Card(t_2), \ldots \sum_{j=1}^{mn} Card(t_m)$$

endfor

//Finally determine the minimum support as the minimum test set size

$$minsupport = MIN(Card(T_i))$$

}

end

Fig 4.1 MFTS Initialization Algorithm

The above process iterates till all the elements of GTC become '0' or there are no more maximal frequent test sets. The resulting T_{rs} table contain the reduced test suite of maximal frequent test sets. The resulting T_{rs} table contains the reduced test suite of maximal frequent test sets. As m test sets are added to the MFTS source table in a linear manner the computational complexity becomes O(m) for test suite reduction.

Algorithm: MFTS Mining

Input: MFTS_Sorce table, min_Support, Global vector GTC, Vector TC_j

Output: T_{rs} table

Begin

{

//Retrieve each test set placed in the source table

From each T_i in MFTS_Source do

//initialize the support value of the test set as one

$$Support(Ti) := 1$$

//place the test set in a temporary table

$$MFTS := MFTS \cup T_i$$

//Recompute the value of GTC after including the test set

Compute

$$GTC = GTC - TCi$$

//Determine the common test cases in the temporary table

for each $T_i$in MFTS do

$$if(Ti \cup Ti + 1)arenotdisjoint$$

//Increment the support value of the subset

support(T_i)=support(T_i)+1

endif

endfor

for each T_i in MFTS do

//check for maximal frequent test set

If(support(T_i>=min_support)

//If a test set is maximal frequent then add the test set in the representative set

$$T_{rs} := T_{rs} \cup T_i$$

//Remove the occurrence of this test set from the temporary table

$$MFTS := MFTS - Ti$$

Endif

Endfor

for each $t_j$in GTC do

//Determine if the cardinality of any test case has become zero

if(t_j == 0)

//Prune the occurrence of the test case t_j that is infrequent from all the test sets in MFTS table

T_i: = remove t_j in T_i

Endif

Endfor

}

End

Fig 4.2 MFTS Mining Algorithm

V. EVALUATION USING APFD METRIC

A. APFD(Average Percentage of Fault Detection) Metric

The aim is to increasing a subset of the test suite's rate of fault detection using a metric called APFD developed by Elbaum et al [5]. The APFD is measured by taking the weighted average of the number of faults detected. So as the formula for APFD shows that scheming APFD is barely attainable when previous data of faults is accessible. APFD calculations therefore are only used for evaluation.

$$APFD = 1 - \frac{TF1+TF2+TF3+\cdots+TFm}{mn} + \frac{1}{2n} \quad \text{--- (5.1)}$$

Where,

T - Test suite

n - Total number of test cases

m - Total number of faults

TF_i - Position of the first test in T that expose the fault i.

Following table shows the number of faults detected by a test case in the test suite for the given example program and total time taken by each test case.

	T1	T2	T3	T4	T5	T6	T7
F1	X	X				X	
F2				X			
F3							X
F4					X	X	
F5			X		X		
F6				X		X	
F7				X			X
Number of Faults	1	1	1	3	2	3	2
Time	5	7	11	4	10	12	6

A. APFD VALUE FOR NON PRIORITIZED TEST SUITE

m= number of faults=7
n= number of test cases=7
Test sequence= T1,T2,T3,T4,T5,T6,T7
Putting values in formula:
APFD = 1 – (1+4+7+5+3+4+4)/7*7 + 1/(2*7) = 0.50

B. APFD VALUE FOR PRIORITIZED TEST SUITE

The prioritized test sequence obtained after applying Genetic Algorithm and its operators crossover and mutation T4 T7 T1 T5 T3 T2 T6. Calculation on putting values

APFD = 1 – (3+1+2+4+5+1+1)/7*7 + 1/(2*7) =0.72

C. Analysis of APFD

Results calculated by APFD proves that prioritized test sequence is more effective in finding out the faults in less time.

The example programs are measuring their efficiency based on the APFD value. This graph shows that the value after prioritization value increased so that their efficiency also increases.

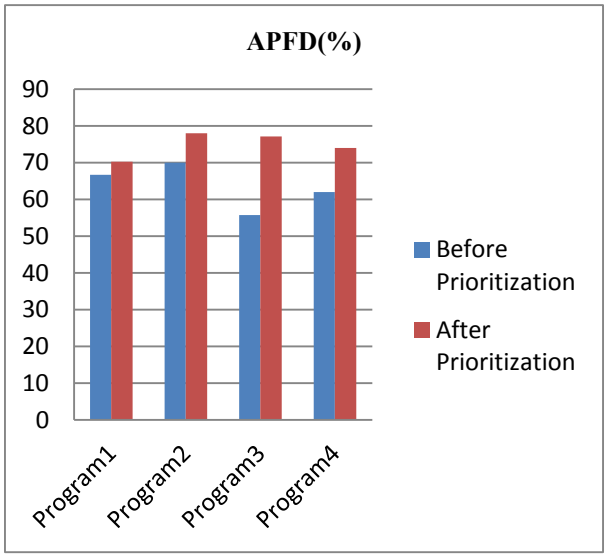

Fig 5.1 Comparison of APFD Metrics

VI. CONCLUSION

The method of prioritization provides an effective ordering of test cases for the given applications In this present work we find the effectiveness of genetic prioritization for regression testing. The obtained test sequence is less costly and less critical then other. The proposed MFTS algorithm for test suite reduction also addressed the test metrics: size, requirement coverage, fault detection and execution time. This algorithm provides efficient in reducing the size and which reveal that the algorithms offered very little or no fault loss while optimizing the test suites.

REFERENCES

[1] Ammann, P, Offutt, J 2008, 'Introduction to software testing', Cambridge University Press.

[2] Binkley, D 1997, 'Semantics guided regression test cost reduction', IEEE Trans. Softw. Eng, vol. 23, no. 8, pp. 498-516.

[3] Chen, TY& Lau, MF 1996,'Dividing Strategies for the Optimization of a Test Suite', Information Processing Letters, vol. 3, pp. 60.

[4] Dennis Jeffrey & Neelam Gupta 2005, 'Test Suite Reduction with Selective Redundancy', Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05).

[5] Elbaum, S, Malishevsky, AG, Rothermel, G 2002, 'Test case prioritization: a family of empirical studies, IEEE Trans. Softw. Eng, vol. 28, no. 22, pp. 159-182.

[6] Harrold, MJ,Gupta, R & Soffa, ML 1993,'A methodology for controlling the size of a test suite',ACM Trans. Software Engineering Methodology, vol. 2, no. 3,pp. 270-285.

[7] Horgan, JR & London, SA 1991, 'A Data flow Coverage Testing Tool for C', Proceedings of symposium on assessment of quality software development tools, pp. 282-290.

[8] Huang, YC, Peng, KL, Huang, CY 2012, 'A History-based cost-cognizant test case prioritization technique in regression testing', J. Sys. Softw, vol. 85, no. 3, pp. 626-637.

[9] Jeffrey, D & Gupta, N 2007,'Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction', IEEE Trans. Software Eng., vol. 33, no. 2, pp. 108-123.

[10] Jones, JA, Harrold, MJ 2003, 'Test-suite reduction and prioritization for modified condition/decision coverage', IEEE Trans. Softw. Eng, vol. 29, no. 3, pp. 195-209.

[11] Jun – Wei Lin & Chin – Yu Huang 2009, 'Analysis of test suite reduction with enhanced tie-breaking techniques', Journal of Information and Software Technology, vol. 51, pp. 679-690.

[12] Lin, CT, Tang, KW, Chen, CD, Kapfhammer, GM 2012, 'Reducing the cost of regression testing by identifying irreplaceable test cases', In the Proceedings of the 6th International Conference on Genetic algorithm and Evolutionary Computing, IEEE Computer Society, pp. 257-260.

[13] Park, H, Ryu, H, Baik, J 2008, 'Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing', In Proceedings of the 2nd IEEE International Conference on Secure System Integration.

[14] Rauf, AM, Arfan Jaffar, Arshad, A & Shahid, M 2011, 'Fully automated GUI testing & coverage analysis using genetic algorithms', International Journal of Innovative Computing, Information and Control, pp.3281-3294.

[15] Rothermel, G, Harrold, MJ, Ostrin, J & Hong, C 1998, 'An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites', In Proceedings of IEEE International Test Conference on Software Maintenance (ITCSM'98), Washington D.C., pp. 34-43.

[16] Shengwei Xu, Huaikou Miao & Honghao Gao 2012, 'Test suite reduction using weighted set covering techniques', Proceedings of the 13th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing(ACIS), pp. 307-312.

[17] Shifa-e-ZehraHaidry &Tim Miller 2013, Using Dependency Structures for Prioritization of Functional Test Suites', IEEE Transaction on Software Engineering, vol. 39, no. 2, pp.258-275.

[18] Scott McMaster & Atif M. Memon 2008, 'Call-Stack Coverage for GUI Test Suite Reduction', IEEE Transactions on Software Engineering, vol. 34, no. 1, pp. 99-115.

[19] Wong, WE, Horgan, JR, London, S & Mathur, AP 1998, 'Effect of test set minimization on fault detection effectiveness', Software Practice and Experience, vol. 28, no. 4, pp. 347-367.

[20] Xue- Ying, Zhen- Feng, H, Bin-kui, S & Chen-quing, Y 2005, 'A genetic algorithm for test suite reduction', Advanced Parallel Processing Technologies Lecture Notes in Computer Science, vol. 3756, pp. 253-262.