Volume 4, Issue 5, May 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Regression Test Suite Minimization using Set Theory

Isha Mangal
Vivekananda College
University of Delhi, India

Deepali Bajaj
Shaheed Rajguru College of Applied Sciences
University of Delhi, India

Priyanka Gupta
Shaheed Sukhdev College
University of Delhi, India

Abstract- Regression testing is a software maintenance activity which is performed to ensure the proper functionality of the software. The test suits developed during the development phase are too large and exorbitant that it is not possible to re-run the entire test suite in the given time and constraints. Thus it is advisable to choose certain test cases and prioritised them with the objective to undersize the whole test suite dimension. In this paper, we use the concept of Set theory for test case minimization from a larger test suite. The proposed technique can be proved to produce better results.

Keywords- Regression Testing, Test Case Selection, Set Theory, Regression Matrix

I. Introduction

Regression testing is a maintenance phase software testing activity that is done to ensure that there are no more bugs in the software.[1,2] The idea is to execute all the test cases that are engendered during the development phase of the software[3]. But due to the time and cost constraints, it is not possible to execute the entire test suite. Regression test selection is a process of reducing the test suite from the original test suite to minimal set of test cases that will cover all the faults in minimal time.

Set theory is the area of mathematical logic that deals with sets, which are collections of items or objects. Any type of object can be collected into a set. Set theory provides us some binary functions such as union (U), intersection (∩), Set difference (-), Cartesian product(X) that can be applied to sets. These functions increase the power of sets. The application of set theory varies from mathematical structures such as graphs, rings, vector spaces to the field of computer science such as trees, discrete mathematics etc. We use the strength of set theoretic functions in this paper for test case selection from a much larger test suite.

Several works has been done on this area. Rothermal [4] discussed on the issues of test case prioritization. Wong [5] addressed the issues of effective regression testing in practice. Many researchers implement many techniques for test case selection and prioritization such as Ant colony optimization [6-7], bee colony optimization [8] and genetic algorithm [8-9] etc.

II. Proposed Approach

In this paper, the proposed algorithm aims to reduce the cost of regression testing by test case suite minimization. The algorithm finds the minimal set of test cases from the given test suite that cover all the faults in minimum execution time.

The proposed approach uses the concept of set theory. The union function of set theory is used as a tool to generate a minimal set of test case suite. The suggested technique introduces the concept of regression matrix. Each cell of regression matrix, c_{ij} where $1 \le i \le n$, $1 \le j \le n$, indicates the union of test case/s t_i and t_j.(Fig 1)

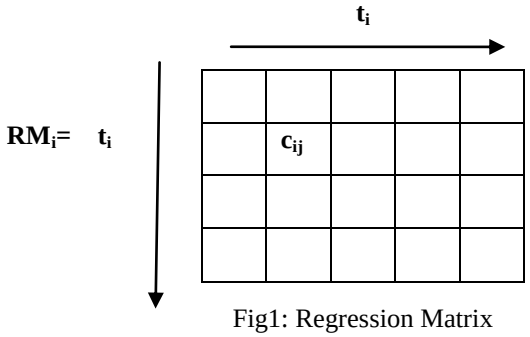

Fig1: Regression Matrix

The algorithm generates new regression matrix $\mathbf{RM_i}$ after each iteration i.

Algorithm:

Prerequisite:

The prerequisite for the proposed approach is a test suite 'T' of 'n' test cases with their given execution time. The value of execution time indicates the total time taken by the test case to find some set of faults. The outcome is subset 'S',

which consists of m test cases (m<=n), such that the test cases are selected on the basis of maximum fault coverage ability in minimum execution time.

Assumptions:

1) Original Test Suite, T={T1, T2,….T_n}
2) Set of faults, F={F1,F2,…….F_n}
3) Each test case T_i where 1<=i<=n, covers some or all faults from F.
4) α be the threshold value to eliminate the test cases covering lesser number of faults after each iteration .

Output: Minimal set S of test cases that covers all the faults in least amount of execution time.

Algorithm:

Initialization:

1. Set threshold value, α=5.
2. Set n = 8, where n is number of test cases.
3. Set i =2.
4. Set S= ϕ

(Each test case T_i is assigned a value which is equal to the number of faults covered by that test case.)

5. For (p=1 to n)
6. Tp=fp

(Where fp is the number of faults covered by Ti.)

7. Do{
8. If(α<10)

(The value of each c_{ij} of regression matrix is calculated by the union operation of row t_i and column t_j.)

9. for (p=1 to n)
10. for (q=1 to n)
11. $\mathbf{c_{pq} = f\{ t_p \}U f\{ t_q \}}$

(Numerically this value is the total number of faults covered by set $\{t_p\}$ and set $\{t_q\}$.)

12. if(c_{pq}>α)
13. S← { S U { t_p t_q },ET}

Where ET= Execution Time taken by test case$\{t_p\}$ and test case $\{t_q\}$.

(S_{red} is the reduced regression matrix generated after each iteration i.)

14. i++
15. α=α+2
16. } while (i<=n)

III. EXAMPLE OF PROPOSED APPROACH

Consider an example given in [6], a test suite with eight test cases in it that covers a total of ten faults. Test case selection from test suite selects a subset of test cases which will cover all the faults in minimum execution time. In this segment, we demonstrate the working of proposed approach using the given example.

Table 1: Algorithm's Input of test cases and faults

Test Case (T_i)	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
T1		X		X			X		X	
T2	X		X							
T3	X				X		X	X		
T4		X		X					X	
T5			X			X				X
T6	X						X			
T7			X			X		X		
T8		X								X

The test suite 'T' given in Table 1, has eight test cases {T1, T2, T3, T4, T5, T6, T7, T8} and ten faults {F1, F2,…..F10}.Each test case covers some faults. 'X' represents fault covered by test case T_i.

The prerequisite for this example assumes the number of faults detected by T as given in Table 2. Test case T1 can find four faults {F2,F4,F7,F9} in seven minutes, T2 finds two faults {F1,F3 } in four minutes, and T3 finds four faults {F1, F5,F7,F8}in 5 minutes. Test cases T4 and T5 find three faults in four minutes {F2,F4,F9} and {F3,F6,F10} respectively. Test cases T6 and T7 find two and three faults in five and four minutes {F1,F7} and {F3,F6,F8}.Test case T8 find two faults {F2,F10} in two minutes.

Table 2: Test cases, no. of faults covered and their execution time

Test Case T_i	No. of Faults Covered F_i	Execution Time (min) ET_i
T1	4	7
T2	2	4
T3	4	5
T4	3	4
T5	3	4
T6	2	5
T7	3	4
T8	2	2

According to our algorithm each test case covers some faults as shown in Table 2.

Iteration 1: Now combinations of test cases are made for i=2.

In first iteration regression matrix is calculated and shown in Table 3 indicating test suite of all possible combinations of two test cases. The value corresponding to each row and column i.e. cell c_{ij} is calculated by the union of faults of test cases $\{t_i\}$ and $\{t_j\}$. T1 and T1 combine can cover only four test cases. T1 and T2 together can cover six test faults. Similarly other value of each cell is calculated. The matrix formed is a symmetric matrix. The order of test cases does not matter as union operation performs on set is symmetric in nature so we have shown only upper diagonal matrix.

Table 3: Regression Matrix: Test Suite of Combination of two test cases

RM_2=

	T1	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**
T1	4	6	7	4	7	5	7	5
T2		2	5	5	4	3	4	4
T3			4	7	4	4	6	6
T4				3	6	5	6	4
T5					3	5	4	4
T6						2	5	4
T7							3	5
T8								2

The value of 10 in a cell represents that a test suite that has covered all the faults. From the table 3, we analyse that in this iteration no test case suite is generated which covers all the faults. So we have to iterate the procedure further and we further consider only those pairs of test cases that cover faults more than the given threshold value (α=5 faults) shown in Table 4. This is a critical step of an algorithm as it finds some sets that can catch more faults than other combinations (Step 8 of suggested algorithm).

Table 4: Result of Iteration 1

S=

	T2	**T3**	**T4**	**T5**	**T7**	**T8**
T1	6	7		7	7	
T3			7		6	6
T4				6	6	

Iteration 2: In this iteration, combinations of test cases are produced by applying union on selected test cases of iteration 1 and original test suite T. For i=3,

RM_3 =

Test Case	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**
T1 T2	X	X	8	6	8	6	8	7
T1T3	X	X	X	7	10	7	9	8
T1T5	X	8	10	7	X	7	8	7
T1T7	X	8	9	7	8	8	X	8
T3T4	7	8	X	X	10	7	9	8
T3T7	9	6	X	9	7	6	X	8
T3T8	8	7	X	8	8	6	8	X
T4T5	7	7	10	X	X	8	7	6
T4T7	7	7	9	X	7	8	X	7

X indicates that this value is not significant as R U S U R = R U S

And By associativity property we know,
(RUS) U T=R U (SUT)
Where R, S, T are sets and U represents union operator.
Those test cases combinations will be extracted whose cell value is 10. From Table 5, we evaluate there are two such test case suite combinations.
S= {{T1T3T5, 16}, {T3T4T5, 13}}

Table 6: Result of Iteration 2

Test Suite	No. of Faults covered	Total Execution Time
T1T3T5	10	16
T3T4T5	10	13

Now we have to iterate the procedure further till either $\alpha < 10$ or a test suite is generated with minimum execution time. Consider only those pairs of test cases that cover faults more than the given threshold (α=7 faults) shown in Table 6. We further emphasize the efficiency of algorithm as from this iteration onwards, optimal solution or near optimal solution are computed.

Iteration 3: For i=4

Table 7: Test Suite of Combination of four test cases

RM_4 =

Test Cases	T1	T2	T3	T4	T5	T6	T7	T8
T1T2T3	X	X	X	8	X	8	X	9
T1T2T5	X	X	X	8	X	8	9	8
T1T2T7	X	X	9	8	9	8	X	9
T1T3T5	X	10	X	10	X	10	10	10
T1T3T7	X	9	X	9	10	9	X	x
T1T3T8	X	9	X	8	10	8	10	X
T1T5T7	X	9	10	8	X	9	X	8
T1T7T8	X	9	10	8	8	9	X	X
T3T4T2	8	X	X	X	10	8	9	9
T3T4T5	10	10	X	X	X	10	10	10
T3T4T7	9	9	X	X	10	9	X	10
T3T4T8	8	9	X	X	10	9	10	X
T3T7T8	10	8	X	10	8	8	X	X
T3T5T8	10	8	X	10	X	8	8	X
T4T5T6	8	8	10	X	X	X	9	8
T4T7T6	8	8	9	X	9	X	X	9

Those test cases combinations will be taken into account whose cell value is 10. Table 7 indicates that the generation of large number of test case suite combinations is computed by this iteration. The entire test suite will be added to set S.
So, the set S becomes (Table 8),

Table 8: Result of Iteration 2

S=

Iteration	Test Suite	No. of Faults covered	Total Execution Time(unit)
Iteration 2	T1T3T5	10	16
	T3T4T5	10	13
Iteration 3	T1T2T3T5	10	20
	T1T3T4T5	10	20
	T1T3T5T6	10	21
	T1T3T5T7	10	20
	T1T3T5T8	10	18
	T1T3T7T8	10	18
	T2T3T4T5	10	17
Iteration 3	T1T3T4T5	10	20
	T3T4T5T6	10	18
	T3T4T5T7	10	17
	T3T4T5T8	10	15
	T3T4T7T8	10	15

As we have seen that the set S has covered all the faults in minimum execution time of 13 units. So, test case set {T3, T4, T5 } is the first minimal set. To spawn, more optimal and efficient results the above procedure should be iterated.

IV. Application of Proposed Approach

The proposed approach is intended for those software developers who wish to reduce their time and effort for the selection of minimal set of test cases from the larger assay of test cases. The technique will further entice software testers who are indulged in a monotonous work of testing. As stated earlier, suggested algorithm exploits the concept of regression matrix. Subsequently, the technique will find reduced reduction matrix which is the summarised result of the algorithm and may contain minimal set of test cases as per the numerical computation of the execution time. The algorithm is efficient in a way that it produces better results from the earliest iterative cycle. The result of the technique may prove to be optimal or near optimal to find the effective reduced set of test cases in a rigid resource constraint environment where maintenance phase testing activity is very time consuming. The outcome of the technique covers all the faults thus proves itself in close proximity with optimal solutions. Hence the proposed approach may prove to be useful in real life situation.

V. Conclusion and future work

We have proposed test case selection approach from a large test suite using the concept of set theory. This approach has been tested for several examples. One of these examples has been shown in this paper. The technique developed using this approach isolates and reduces the test data. The approach endows better results in the initial iteration of the whole process. It provides positive response and thus it leads to improved solutions in optimal time. Scope of future research may incorporate automation of the suggested technique and applying it on multifaceted software. We also intend to implement the technique and compare it with other test case reduction techniques like Ant Colony Optimization and Bee Colony Optimization.

References

[1] G.Duggal, B.Suri,"Understanding Regression Testing Techniques", COIT, 2008, India.

[2] W. E.Wong, J. R. Horgan, S. London and H.Agrawal, "A study of effective regression testing in practice," In Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE' 97), pages 264-274, November 1997.

[3] H. Leung and L. White, "Insights into regression testing," In Proceedings of the Conference on Software Maintenance, pages 60-69, Oct. 1989.

[4] Rothermel, Gregg, et al. "Prioritizing test cases for regression testing."Software Engineering, IEEE Transactions on 27.10 (2001): 929-948.

[5] s Wong, W. Eric, et al. "A study of effective regression testing in practice."Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on. IEEE , 1997.

[6] Singh, Yogesh, Arvinder Kaur, and Bharti Suri. "Test case prioritization using ant colony optimization." ACM SIGSOFT Software Engineering Notes 35.4 (2010): 1-7.

[7] Suri, Bharti, and Shweta Singhal. "Implementing Ant Colony Optimization for Test Case Selection and Prioritization." International Journal on Computer Science & Engineering 3.5 (2011).

[8] Suri, Bharti, Isha Mangal, and Varun Srivastava. "Regression Test Suite Reduction using an Hybrid Technique Based on BCO And Genetic Algorithm."Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (2011): 2231-5292.

[9] Pargas, Roy P., Mary Jean Harrold, and Robert R. Peck. "Test-data generation using genetic algorithms." Software testing verification and reliability 9.4 (1999): 263-282.

