ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 10(29), DOI: 10.17485/ijst/2017/v10i29/106374, August 2017* ISSN (Online) : 0974-5645

An Efficient Heuristic Based Test Suite Minimization Approach

Fayaz Ahmad Khan*, Dibya Jyoti Bora and Anil Kumar Gupta

Department of Computer Science and Applications, Barkatullah University, Bhopal – 462026, Madhya Pradesh, India; kfayaz1012@gmail.com, research4dibya@gmail.com, akgupta_bu@yahoo.co.in

Abstract

Objectives: Development of an efficient test suite minimization approach in order to reduce the size of a previously acquired test suite and produce a new representative suite which will guarantee the same requirement coverage that was achieved before minimization for an effective and efficient regression testing. **Method:** Test suite minimizations techniques try to reduce the size and redundancy of test suite by removing certain test cases since requirement covered by them are already covered by other test cases. But, it has been found that the acquired test cases after minimization severely lacks ability to achieve the desirable code coverage because the minimization was done based on a single test adequacy criteria. In this paper, we propose an efficient heuristic based test suite minimization algorithm which will reduce the size of the test suites with respective to multiple test adequacy criterions in order to preserve the fault detection effectiveness and code coverage characteristics of the final test suite. **Findings:** Our experimental results indicate that a significant percentage of reduction in the test suite size is achieved when the minimization is performed with respect to multiple test adequacy criterions. Our approach is unique compared to the existing approaches in the sense that, we carried out minimization based on multiple test adequacy criterions while most of the existing approaches usually take one or two criterions into consideration. The proposed approach is evaluated based on two well known software testing metrics; one indicate the percentage of reduction in test suite size and the second one indicate the percentage of code coverage achieved by the minimized test suite. Our experimental results indicate that a significant percentage of reduction in the size as well as significant code coverage characteristics is achieved when the minimization is done according to the proposed approach. **Improvements:** The important contribution of this study is that, it presents a novel and efficient test suite minimization technique that optimizes the test suite size based on multiple adequacy criterions.

Keywords: Regression Testing, Software Testing, Test Data Generation, Test Suite Minimization, Test Suite Selection and Data Clustering

1. Introduction

Software engineering is a well-defined approach to the analysis, design, implementation, testing, maintenance and re-engineering of a product. In software engineering, testing is an important activity used to identify the defects and problems associated with the product being developed. The testing process is usually expensive and may represent 50% of the software development budget. The key factor responsible is the size of test suite as it takes a very long time to execute the whole generated test suite. The test suite size goes on increasing when software undergoes maintenance, because new test cases are needed to test the exiting code and the code that is newly added. Thus changes to existing program lead to the expansion in size of the test suite. Therefore test suite management becomes an important research issue and in literature[1–7] it is termed as test suite minimization or test suite reduction. The aim of test suite minimization is to reduce the size of a previously acquired test suite and produce a new representative suite which will guarantee the same requirement coverage that was achieved before minimization.

The other related issue during software maintenance is regression test case selection. Regression testing is used to validate the modified software to detect whether

* *Author for correspondence*

new faults are introduced into a previously tested code. Regression selection techniques are also used to reduce the cost by selecting and running a subset of test cases from the previous test suite. Test suite minimization and test case selection techniques are similar because both of them try to acquire a subset from the previously existing test suite, but can be differentiated based on the criteria imposed. For the test suite minimization, the criteria are whether the minimized test set cover all the test requirements whereas as selection process focus on the modified parts of the system under test. A test case is called redundant when it covers the requirement that has already been covered by another test case.

2. Test Suite Minimization and the Intuition Behind our Approach

Test suite minimization is an optimization problem and its goal is to, select a minimized subset of test cases from an existing test suite that exercises the same set of requirements as exercised by the initial test suite. The definition according to [3] is:

2.1 Definition

Given T (a test suite), R (a set of test requirements r1, r2, r3 …, r_i) that must be satisfied to get the desired coverage of the program.

2.2 Problem

Find a subset T′ **of** T that satisfies all r_i^s**.**

Trying to find a representative set T′ that will cover the same requirements as covered by the initial test suite is the NP complete problem[8]. Due to NP completeness, test suite minimization encourages the use of heuristic approaches and in literature, many such approaches have been developed to produce a minimum hitting set[3,9,10]. In[9], proposes the usage of simple greedy heuristic in which each candidate set has a cost associated with it and chooses test cases which cover almost all requirements about to be covered, until are accomplished. But, a potential weakness of the approach is that the early selection made can eventually be rendered redundant by the test cases subsequently selected. Another greedy heuristic approach developed by authors in[3] chooses a minimal subset which covers the same set of requirements as covered by un-minimized test suite. The optimal test suite generated using the proposal of[3] is equally good or better than computed by[9]. The ping pong procedure developed by[10], declared that the technique presented in[3] is more expensive than their proposed approach. Another study proposed by[11] which is also known as double delayed greedy heuristic tried to overtake the weakness of the previous heuristic approaches by developing a concept lattice. The approach by[11] works in three phases: (1) apply reduction by removing test cases test cases whose test requirement are subsumed by other test cases; (2) remove test requirements that are not present in the minimal requirement set; (3) generate test suite using greedy method[12] from the remaining test cases. The empirical results showed that the minimized test suite minimized using double delayed greedy approach[11] were smaller or even smaller than minimized by traditional greedy approaches[13].

The previous studies[3,9–11] at one end focus alone on single standard minimization problems and have achieved high test suite reduction, but, at the other end have neglected the other important dimensions like fault detection effectiveness and code coverage characteristics of the minimized test suite. In a study reported in[16], it was observed that test suites achieving over 80% size reduction during minimization by different techniques suffer with less fault detection loss (around 50% loss on average). In another study proposed in [15] in which it was observed that only 7% to 16% fault detection effectiveness loss happened to a test suite which undergoes 82% to 94% size reduction on average. Authors in[14] carried an empirical investigation to deal with the limitations of single criterion minimization approaches by taking two testing demands (requirements) into account instead of one. The results reported in [14] showed that fault detection effectiveness was better preserved by returning the larger test suite compared to the test suite returned by single criteria version of the HGS heuristic[13].

Despite encouraging results by many studies like[14–16] however, there is still definitely much room for improving the existing techniques and developing new techniques for an effective and efficient testing.

3. The Proposed Test Suite Minimization Approach

To find a representative set that satisfies all the

requirements initially satisfied before reduction or minimization is a NP complete problem. Therefore, we are unaware of any approximate solution, hence a heuristic based approach is proposed as shown in Figure 1, which will find a representative set of test cases with minimum cardinality from the initial test suite based on different code coverage matrices. The different steps of the proposed test suite minimization approach are shown in Figure 1. The important characteristics of our proposed approach is that we have used a filter in the form of an array (step 2 to step 4) to throw away redundant test cases from the test suite according to different code coverage criteria's. A test case is redundant if the requirements satisfied by it have been previously satisfied by any other test case present in the test suite.

4. The Application of the Proposed Approach

The proposed approach is initially implemented on a test suite of a single sample program shown in Figure 2 and then extended to a large suite of sample programs. To test the program, a suite of test cases are generated and with an automated test data generation tool known as genratedata.com[19]. The Initially generated test suite is

input:
TS[i][j]…. all test cases present in the test suite
CM1.txt, *CM2.txt* and *CM3.txt*, Comma Separated Coverage Information in text format representing statement, branch and independent path coverage of each test case. 1 for covered and 0 for uncovered
output: *RS*: a reduced set of test cases from the test Initial Test Suite.
declare:
CM1 [*m*][*s*], *CM2* [*m*][*b*] and *CM3* [*m*][*ip*]: Matrices used to hold the data from *CM1.txt*, *CM2.txt* and *CM3.txt* files.
m[],m1[],m2[]: Index of test cases returned after Minimization Process.
r[],r1[],r2[]:array[1..n], Initially Empty, representing the requirements Covered by minimized test cases in *m[],m1[],m2[]*.

algorithm TestSuiteMinimization

begin

STEP 1: Initialize each *CM1* [*m*][*s*], *CM2* [*m*][*b*] and *CM3* [*m*][*ip*] Matrices by reading each text files using Java.io.BufferedReader Class.

STEP 2:
for-each *CM1[i][j]* **do** *Minimization with Respect to Statement Coverage Perspective*
if r(j) == 0 **and** *CM1[i][j]* **equals to** "1" **then**
r[j]= *CM1[i][j];* *Requirement Satisfied by Test cases*
m[i]=i+1; *Index of the Test Cases that have satisfied the Above Requirements*
endfor

STEP 3:
for-each *CM2[i][j]* **do** *Minimization with Respect to Branch Coverage Perspective*
if r1(j) == 0 **and** *CM2[i][j]* **equals to** "1" **then**
r1[j]= *CM2[i][j];*
m1[i]=i+1;
endfor

STEP 4:
for-each *CM3[i][j]* **do** *Minimization from Independent Path Coverage Perspective*
if r2(j) == 0 **and** *CM3[i][j]* **equals to** "1" **then**
r2[j]= *CM3[i][j];*
m2[i]=i+1;
endfor

STEP 5: $RS := \{\{m\} \text{Union } \{m1\} \text{ Union } \{m2\}\}$; To further Remove the Redundant Test Cases
return *RS*;

end TestSuiteMinimization

Figure 1. The proposed approach.

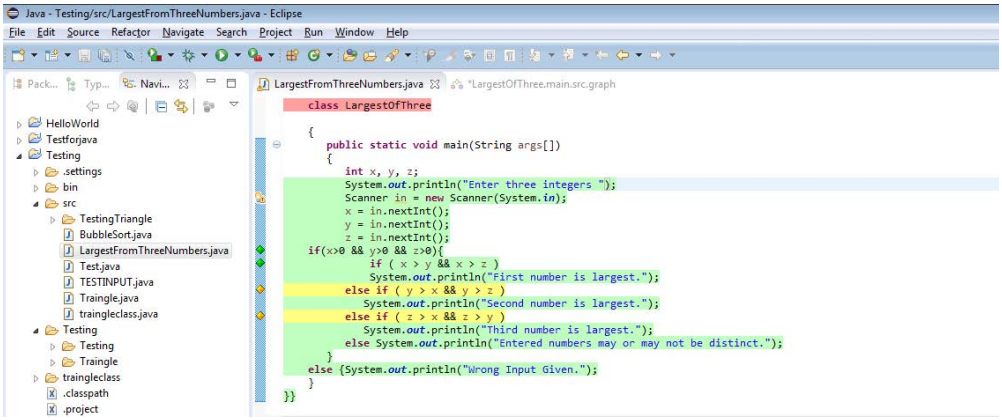

Figure 2. Statement coverage of RS1 (RS1 is statement adequate but not branch and independent path adequate).

presented in Figure 3. To remove the inconsistency of not generating fault revealing test cases, an efficient approach to test data generation is followed using our previous study[17]. Through the previous study reported in[17], few combination of test cases that are added to the initially generated test suite are (10,10,10), (0,0,0), (1,2,2), (2,2,1), (1,2,1), (-1,-2,-3) and many others. The tool generates a large volume of test cases and is not an effective choice for an initial and regression testing due to many reasons like, its size is huge, and it is redundant and hence will take time to execute.

To determine the adequacy or efficiency of a test suite, test case requirements play an important role. Test case requirements in case of black box testing are derived from program specifications while in case of white box testing; they are derived from program components. For the present study, we are employing program component based test adequacy criterion like statements, branches, and path coverage to determine the adequacy of the reduced test data set[18]. A test data set is adequate when all the test requirements are covered otherwise more test cases are added to achieve the desired coverage[18]. The experiments are carried in Eclipse with control flow graph factory, JUnit and EclEmma as plug-ins for knowing the structural components, test case execution and code coverage measurements.

4.1 Minimization with Respect to a Single Adequacy Criterion

The initial test suite when minimized using the proposed approach based on single test adequacy criteria (Statement coverage criteria) will result in formation of a sub-optimal representative subset RS1 shown in Table 1.

Test Case	X	Y	Z	t50	-1	102	88
t1	0	74	1	t51	35	110	53
t2	99	110	73	t52	67	11	0
t3	0	65	53	t53	104	23	98
t4	26	106	-10	t54	65	40	106
t5	56	101	2	t55	1	70	-6
t6	41	62	-2	t56	16	42	106
t7	46	45	46	t57	-4	71	43
t8	14	103	39	t58	44	96	102
t9	-9	55	1	t59	49	90	100
t10	24	17	90	t60	39	2	84
t11	78	67	47	t61	27	12	50
t12	68	9	26	t62	75	2	78
t13	71	-2	91	t63	11	108	16
t14	16	85	1	t64	84	108	3
t15	19	-5	87	t65	55	11	85
t16	107	-2	14	t66	103	55	55
t17	11	19	84	t67	89	-3	-5
t18	23	49	44	t68	54	23	52
t19	83	28	39	t69	92	78	94
t20	27	47	35	t70	77	79	107
t21	19	2	45	t71	81	47	27
t22	-4	109	106	t72	45	14	20
t23	105	-10	109	t73	78	47	94
t24	74	51	59	t74	63	15	27
t25	28	75	94	t75	57	73	87
t26	91	21	73	t76	26	33	53
t27	27	62	60	t77	105	54	64
t28	62	18	26	t78	109	27	79
t29	91	107	106	t79	-1	108	24
t30	77	-5	73	t80	28	14	72
t31	76	46	20	t81	14	58	4
t32	24	0	93	t82	36	9	67
t33	63	30	104	t83	23	90	10
t34	65	77	7	t84	32	103	101
t35	104	106	51	t85	32	-5	66
t36	19	85	22	t86	109	58	79
t37	77	60	96	t87	-6	71	43
t38	25	31	91	t88	53	63	52
t39	-7	107	49	t89	12	35	41
t40	38	33	73	t90	103	-3	50
t41	29	97	70	t91	16	-10	26
t42	65	94	93	t92	9	-2	91
t43	108	73	41	t93	91	71	61
t44	46	82	100	t94	87	20	88
t45	1	91	2	t95	45	48	76
t46	14	63	61	t96	86	90	48
t47	10	22	75	t97	110	27	84
t48	93	60	74	t98	93	91	0
t49	50	89	16	t99	48	49	-8
t50	-1	102	88	t100	46	-6	46

Figure 3. Automated test generated test suite.

Table 1. Sub-optimal test suite (only Statement Adequate)

Test Case ID	X	Y	Z
t11	78	67	47
t2	99	110	73
t10	24	17	90
ts1	10	10	10
ts2	0	0	0

The minimized test suite or representative suite RS1= {t11, t2, t10, ts1, ts2} when executed achieves around 94.1% statement code coverage and is shown in Figure 3.

The code coverage characteristic of other components of the subject program achieved by the RS1 is shown in Figure 4.

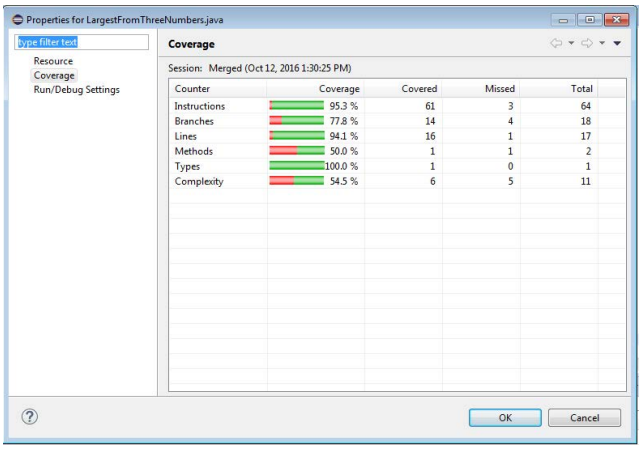

Figure 4. Requirement coverage of the reduced test suite RS1 (minimized with respect to a single objective test criteria).

The results in Figure 2 and in Figure 4 clearly indicate that when minimization is done with respect to single test adequacy criteria, a significant reduction in the size of test suite (from 110 to 5 test cases) is achieved. The code coverage (statement) coverage is also significant and is found to be (94.1%). But it also evident from the results that the minimized sub-optimal test suite RS1 does not achieve an efficient coverage in terms of the other components of the code and will also have less fault detection effectiveness.

4.2 Minimization with Respect to Multiple Test Criterions

In order to enhance the code coverage efficiency and fault detection effectiveness of a test suite, minimization should always be carried out using multiple adequacy criterions.

With the proposed test suite minimization, the initial test suite is further minimized with respect to branch and path coverage perspective in order to improve the code coverage and fault detection effectiveness. The minimized branch coverage and independent path coverage test suite are depicted in Table 2 and Table 3.

Table 2. Branch Adequate Test suite (RS2)

Test Case ID	X	Y	Z
t11	78	67	47
t2	99	110	73
t10	24	17	90
ts1	10	10	10
ts2	0	0	0
t32	24	0	93
t52	67	11	0
ts4	1	2	2
ts5	2	2	1
Ts7	-1	-2	-3
Ts6	2	1	2

Table 3. Independent Path Adequate Test Suite (RS3)

Test Case ID	X	Y	Z
t11	78	67	47
t2	99	110	73
t10	24	17	90
ts1	10	10	10
ts2	0	0	0
t32	24	0	93
t52	67	11	0
Ts3	1	1	1
ts4	1	2	2
ts5	2	2	1
Ts7	-1	-2	-3

The final outcome of the proposed technique is the RS, a representative subset of TS formed by the union of RS1, RS2 and RS3. The purpose of taking union between RS1, RS2 and RS3 is to further minimize the size and redundancy among test cases.

RS = [(RS1) U (RS2) U (RS3)]

RS= {t11, t2, t10, ts1, t32, t52, ts1, ts2, ts3, ts4, ts5, ts6, ts7}

With the implementing of the proposed approach, a considerable amount of reduction in the number of test cases (from 110 to 11) is also achieved and is given as:

$$\% \text{ Reduction } = \left(\frac{TS-RS}{110}\right) * 100 = \frac{110-13}{110} * 100$$

= 88 %.

The final representative suite RS after execution as shown in Figure 5 and Figure 6 has achieved above 95% statement coverage, 100% branch coverage and around 90.90% path coverage.

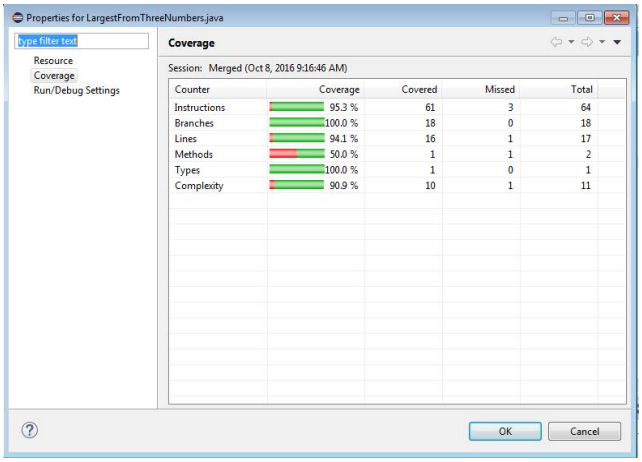

Figure 5. Code Coverage of the Reduced Test Suite (RS).

5. Application on Large Study

The proposed approach after its successful implementation on a single subject program is now carried on a suite of well known programs. The suite of programs and their corresponding test cases are listed in Table 4. The present study will evaluate the proposed approach with respect to the size and code coverage perspective. The other important parameter is the fault detection effectiveness measure. Fault detection effectiveness determines the fault detection ability of the reduced or minimized test suite. It is observed from some studies, one reported in[16] that test suite reduction can reduce the fault detection ability of the resulted minimized test suite significantly. But, on the other end, it is also reported in some studies like[20], that test reduction approaches achieve a substantial savings with little cost to fault detection effectiveness. In case of present study we assume that the fault detection ability of the minimized test suite is preserved due to the fact that the minimization is done with respect to multiple coverage criterions. In our previous studies[21], we have employed data clustering techniques for test suite minimization and have minimized test suite with respect single adequacy criteria, but this study presents an efficient heuristic based approach for the same reason with multiple test adequacy criterions.

Table 4. Suite of Test Programs and test data

S. No.	Subject Programs for Experimentations.	Total Number of Test Cases
P1	Triangle Classification	110 rows
P2	Roots of Quadratic Eq.	100 rows
P3	Largest of Three Numbers	110 rows
P4	Bubble Sort	100 rows

5.1 Evaluation with Respect to Size

The proposed approach is validated with respective to the size of the test suite and coverage of the structural components achieved for all programs. So, for an efficient testing and also for regression testing only a subset of test cases is required and this acquired subset should possess less number of test cases and should satisfy all the specified test requirements criterions.

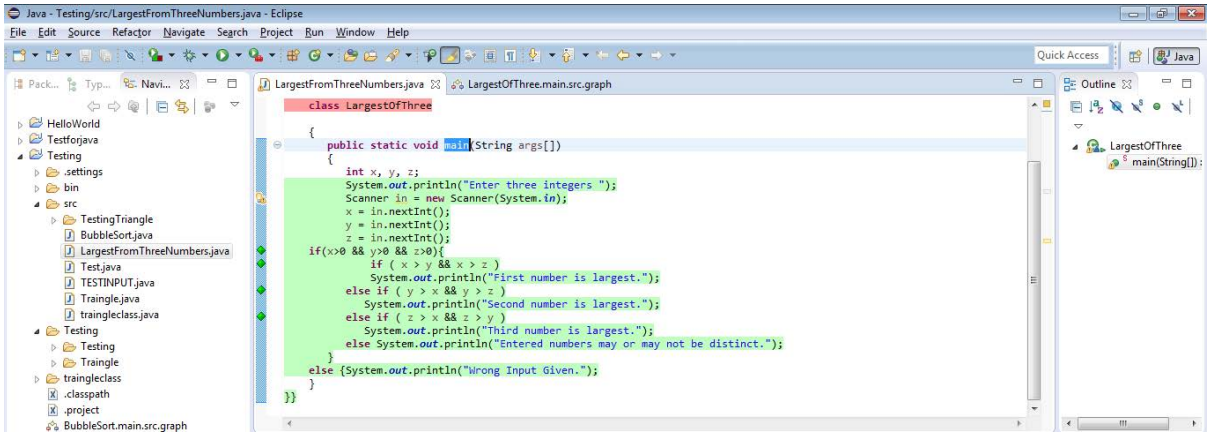

Figure 6. Statement and branch coverage.

The proposed approach after implementation achieved a considerable amount of reduction in the number of test cases for each program. The comparison of the size between initial test cases and the test cases acquired after implementation of proposed minimization approach is depicted in Figure 7.

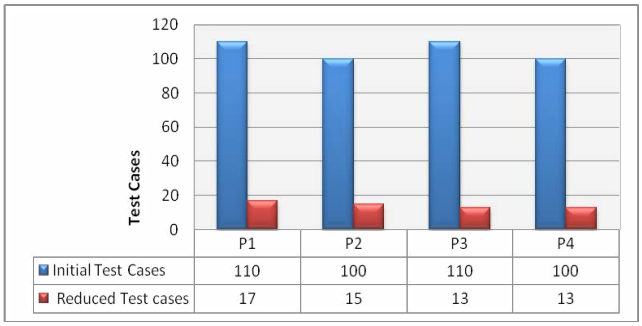

Figure 7. Comparison in terms of size between the original and minimized test cases.

The percentage of reduction in each test suite is also very effective and is calculated as:

$$\% \text{ of reduction for P1} = \frac{110-17}{110} * 100 = 85\%$$

$$\% \text{ of reduction for P2} = \frac{100-15}{100} * 100 = 85\%$$

$$\% \text{ of reduction for P3} = \frac{110-13}{110} * 100 = 88\%$$

$$\% \text{ of reduction for P4} = \frac{100-13}{100} * 100 = 87\%$$

5.2 Evaluation with Respect to Multiple Requirement Coverage Criterions

In this study multiple requirement coverage criteria's are used to determine the adequacy of each test suite. The proposed approach is also evaluated with respect to the following well known adequacy criteria's:

(1) The number of Instructions covered,
(2) The number of branches covered,
(3) The number of statements or lines exercised and
(4) The number of independent paths covered.

The specified requirement coverage resulted by the minimized test suite minimized using the proposed approach for each subject programs P1, P2, P3 and P4 are given in Figure 8, Figure 9, Figure 10, and Figure 11. The Figure 8–11 represents the total number components, the number of components covered and the number of components missed by the minimized test cases against experimental programs P1, P2, P3, and P4.

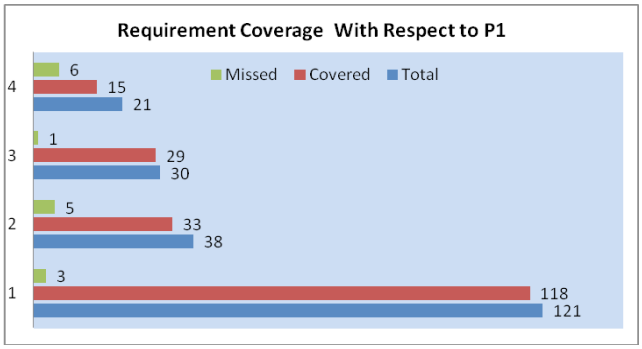

Figure 8. Requirement coverage of the specified components of P1.

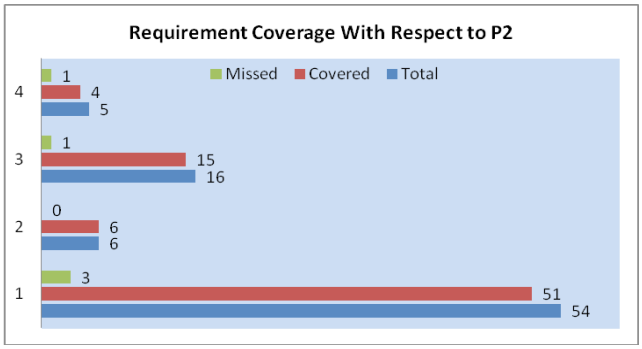

Figure 9. Requirement coverage of the specified components of P2.

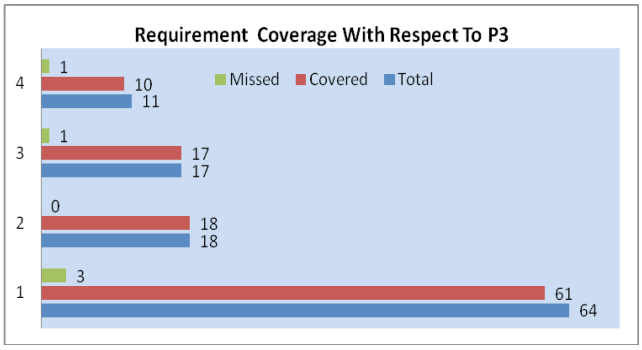

Figure 10. Requirement coverage of the specified components of P3.

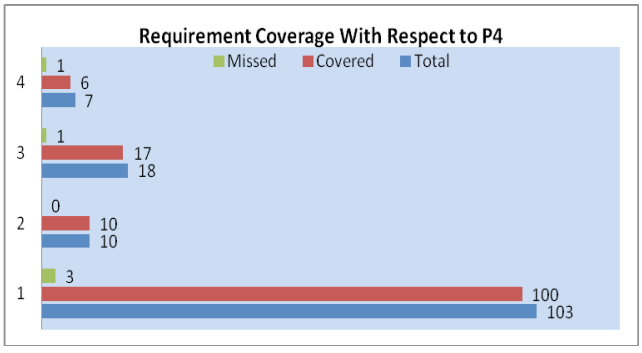

Figure 11. Requirement coverage of the specified components of P4.

6. Conclusion

In this study, we propose an efficient test suite minimization approach for unit testing. The proposed approach is evaluated with respect to two well known test metrics such as test suite size and test adequacy criteria. It has been observed by our experimentation that the size of the initial test suite is reduced to a great extent without compromising the test suite code coverage characteristics and its fault detection effectiveness. Although we have not measured the fault detection effectiveness of the minimized test suite but it assumed that it is preserved because the reduction is performed with respect to multiple code coverage criteria's. A test suite which is statement adequate, branch adequate and path adequate would also be effective in terms of fault detection effectiveness. The requirement coverage of the acquired test suites with our proposed technique is also very good. The future scope of this study would be to measure the fault detection effectiveness of each minimized test suite on a large study and its comparison with other proposed test suite minimization approaches.

7. References

1. Black J, Melachrinoudis E, Kaeli D. Bi-criteria models for all uses test suite reduction. Proceedings of the 26th International Conference on Software Engineering, 2004. p. 106–15. Crossref
2. Chen TY, Lau MF. A new heuristic for test suite reduction. Information and Software Technology. 1998 Jul; 40(5-6):347–54. Crossref
3. Harrold MJ, Gupta R, Soffa ML. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology. 1993 Jul; 2(3):270–85. Crossref
4. Hartmann J, Robson D J. Revalidation during the software maintenance phase. Proceedings Conference on Software Maintenance, 1989. p. 70–80. Crossref
5. Horgan JR, London S. A data flow coverage testing tool for C. Proceedings of Second Symposium on Assessment of Quality Software Development Tools, 1992. p. 2–10. Crossref
6. Mansour N, El-Fakih K. Simulated annealing and genetic algorithms for optimal regression testing. Journal of Software: Evolution and Process. 1999 Janl; 11(1):19–34.
7. Offutt AJ, Pan J, Voas JM. Procedures for reducing the size of coverage based test sets. Proceedings of 12th International Conference on Testing Computer Software, 1995 Jun. p. 111–23.
8. Garey MR, Johnson DS. Computers and Intractability. A Guide to the Theory of NP-Completeness. New York: W. H. Freeman & Company; 1979. PMCid:PMC1619045
9. Chvatal V. A Greedy Heuristic for the Set-Covering Problem. Mathematics of Operations Research. 1979 Aug; 4(3):233–5. Crossref
10. Tallam S, Gupta N. A concept analysis inspired greedy algorithm for test suite minimization. ACM SIGSOFT Software Engineering Notes. 2006 Jan; 31(1):35–42. Crossref
11. Hsu HY, Orso A. MINTS: A general framework and tool for supporting test-suite minimization. IEEE 31st International conference on Software Engineering, 2009. p. 419–29. Crossref
12. Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability. 2012 Mar; 22(2):67–120. Crossref
13. Jeffrey D, Gupta N. Test suite reduction with selective redundancy. Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. p. 549–58. Crossref
14. Heimdahl MPE, George D. Test-Suite Reduction for Model-Based Tests: Effects on Test Quality and Implications for Testing. Proceedings of the 19th IEEE International Conference on Automated Software Engineering, 2004. p. 176–85. Crossref
15. Rothermel G, Harrold MJ, Ostrin J, Hong C. An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites. International Conference of Software Maintenance, 1998 Nov. p. 34–43. Crossref
16. Gupta AK, Khan FA. An Efficient Test Data Generation Approach for Unit Testing. IOSR Journal of Computer Engineering (IOSR-JCE). 2016; 18(4):97–107. Crossref
17. Zhu H, Hall PAV, May JHR. Software unit test coverage and adequacy. ACM Computing Surveys. 1997 Dec; 29(4):366–427. Crossref
18. The random test data generation tool. Available from Crossref Accessed on 25/11/2016.
19. Wong WE, Horgan JR, London S, Mathur AP. Effect of test set Minimzation on fault detection effectiveness. Proceedings of the 17th international conference on Software engineering, 1995. p. 41–50.
20. Khan FA, Gupta AK, Bora DJ. An Efficient Technique to Test Suite Minimization using Hierarchical Clustering Approach. International Journal of Emerging Science and Engineering (IJESE). 2015 Sep; 3(11):1–9.
21. Khan FA, Gupta AK, Bora DJ. Profiling of Test Cases with Clustering Methodology. International Journal of Computer Applications. 2014 Nov; 106(14):32–7.