ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

AN ANALYSIS OF TEST SUITE MINIMIZATION TECHNIQUES

Shilpi Singh*, Raj Shree
Department of Information Technology, BBAU, Lucknow, India

DOI: 10.5281/zenodo.165632

ABSTRACT

In software development, testing is widely used by developers to reveal faults that cause failures and improve the quality of the software by omitting or removing the detected faults. Test suites for testing and validating software programs can contain large numbers of test cases to execute various parts of the software program code to check for defects and code compliance. Test suite sizes may grow significantly with subsequent modifications to the software over time that causes redundancy in test suites. Due to time and resource constraints retesting the modified software every time, it is advantageous to develop techniques that manage test suite sizes by periodically removing duplicate test cases, in turn, can improve the maintenance efforts required for regression testing. The primary objective of test suite reduction is to effectively improve the testing process while maintaining Fault Detection Effectiveness and fulfilling all the testing requirements as well. Researchers have proposed some TSR heuristics based on different variations of a greedy algorithm. This paper presents a comparative study of test suite reduction techniques based on some parameters.

KEYWORDS: Software Testing, Regression Testing, FDE, Test Suite, Test Cases.

INTRODUCTION

Software testing is the most commonly used but expensive method for demonstrating that a software program performs its intended function. The goal of software testing is to execute the software system, identify the faults that cause failures, and improve the software quality by removing the identified faults. But program. Adding some value means improving the quality and reliability of the program.

Inadequate testing is one of the major cost factors. Early detection of faults and failure reduces maintenance costs as well as fewer corrections needed. In software testing, the testing requirements are gathered from SRS (Software Requirement and Specifications). Once a set of requirements is found, a set of test cases (test suite) generated to fulfill the requirements manually or automatically [6, 7]. According to the IEEE definition [2], a test case is a collection of input data and expected output results which are mainly created to evaluate a particular software function or test requirement. It 's hard for a single test case to satisfy the entirely given test requirements. That's why; a number of test cases are generated and collected in a test suite [3].

During the software development lifecycle, regression testing plays an important role. Maintenance requires some modifications, which leads to growth in software and it results in an increment in test suites size. Over time, some test cases in a constructed test suite may become redundant because the test cases created specifically for a testing requirement or testing criteria may also satisfy other requirements, and a requirement may still satisfy by some of the proper subsets of the test suite. A test case in a test suite either said to be redundant or essential. Two test cases are termed as duplicate or redundant if their satisfied testing objectives (testing requirements or criteria) are same. On the other hand, some of the test cases are termed as essential if their testing purpose is unique (not satisfied by remaining test cases).

Due to the constraints of time and resources, it may be not viable to re-execute all the test cases to test the modified software. Therefore, it is advantageous to maintain test suite sizes by removing duplicate test cases. This process is known as test suite minimization (also known as test suite reduction) [13].

There is mainly three reasons for test suite reduction problem [26]: (1) redundancy in a constructed test suite; (2) cost of executing test cases may be high; (3) cost of maintaining test suite may be high. The objective of test suite reduction is to find a subset of the test suite (Optimal Representative Set) that can still fulfill all the testing requirements [22]. Thus, it is not always cost effective.

However, the experimental study conducted in [14] observed that minimized test suites can rigorously compromise the fault detection capabilities (FDC) of the test suites. They stated two situations: High Test Suite Size Reduction without considerable loss in FDE (fault detection effectiveness) and High Test Suite Size Reduction at the cost of significant loss in FDE.

Test suite reduction, therefore, is a trade-off between the suite's size and fault detection effectiveness. An important strategy for test case optimization could be to identify an optimal Representative Suite (RS) while maintaining their Fault Detection Capability (FDC) as achieved by the original test suite [8]. There are different parameters available, based on that reduction or minimization process continues. Coverage criteria are also used as a parameter to decide when a program is sufficiently tested. In this case, extra tests are added until the test suite has achieved a particular coverage level according to a specific adequacy criterion [9].

Test case selection problem is also one of the research problem related to regression testing. After a software system is modified, it is necessary to provide the confidence that the modification does not obstruct the functionality of the existing system. It is not necessary to execute all the test cases in the original test suite which may be too costly. So test case selection can be performed to identify the subset of test cases used for testing to get that confidence [12].Test case selection is a densely discussed problem, and empirical studies on various test case selection techniques can be found in [11, 15]. Test case selection and test suite minimization both are about selecting an optimal subset of test cases from the test suite. But they are differing in their selection procedure. In minimization, test cases are selected mainly based on coverage information. By contrast, test case selection aims to select the test cases based on some changes in the current version of a software system.

Besides test suite reduction and test case selection, test case prioritization is also the interesting and required research topic for optimizing the result of reduction problem (see e.g. [4, 8]).Test case prioritization seeks to order the test cases for testing in such a manner that when the test cases are executed, tester gets maximum benefit regarding early fault detection.

The remaining parts of the paper are organized as follows: in Section 2 we provide a background of regression testing and some information regarding test cases. Section 3 describes the test suite minimization problem and different parameters used to analyze minimization techniques. Section 4 presents and defines a brief discussion on existing algorithms for Test Suite Reduction. Section 5 comparatively analyzes these reduction techniques based on selected parameters. Finally, Section 6 reports the overall results and conclusion.

BACKGROUND

In this section, we briefly discuss the basic concepts of regression testing, their techniques, and classification of test cases.

Regression Testing

The purpose of regression testing is to re-establish the confidence that the newly introduced features or any function in the updated software program have not badly affected existing features. It ensures that previous program code still works once the new code modifications are done. In the maintenance phase, regression testing works as a major component. Retesting of the program is most of the time as tedious as the original test. It means if the modification causes the existing functional part of the program to fail, this error often goes undetected. To solve this problem we can use retest-all approach [21].

However, as software grows after required modifications, the test suite also grows accordingly, which means it may be too costly to execute whole test suite. This drawback forces researcher to consider the development of techniques that aims to reduce the cost and effort required for regression testing in various ways [21]. Regression testing can be characterized into Progressive Regression testing and Corrective Regression testing [26]. Progressive Regression testing involves changes of requirement specifications because of new enhancement or new data requirements in a system. In contrast, Corrective Regression testing does not involve changes in requirement specifications, but only in some design decisions and actual instructions of the program [21].

The main branches to aid the regression testing process have been studied. These are test suite minimization, test case selection, and test case prioritization. Test suite minimization is a process that tries to identify similarity and diversity between test cases and then accordingly remove the obsolete or duplicate test cases from the test suite. Test case selection mainly concerned with the problem of choosing a subset of test cases that will be used to verify the modified parts of the software. At last, test case prioritization concerns the finding of the `ideal' ranking of test cases that improves desirable coverage properties, such as early fault detection [21].

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

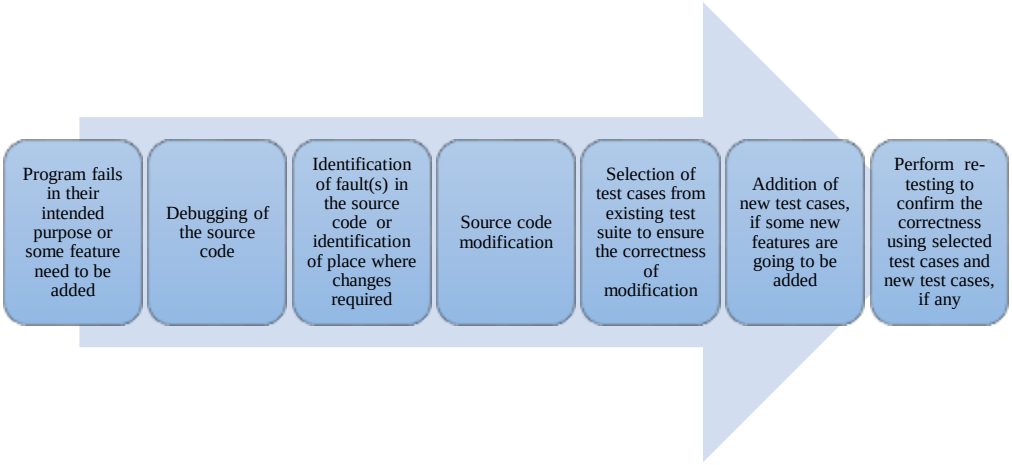

Figure 1: Regression Testing Process

Categories of Test Cases

Leung and White [26] classify test cases into five different groups.

Reusable: This class of test cases only executes the parts of the software program that remain unchanged between two versions P and P0. However, they are termed as reusable because they may still be reused for the regression testing of the updated versions of program P0.

Retestable: This class of test cases executes the parts of program P that have been changed in P0. Thus, retestable test cases must be re-run to test P0.

Obsolete: This class of test cases no longer proves what they were designed to test due to modifications in the program.

After the modification, two new groups or classes may be created which contains those test cases that have yet to be generated for the regression testing of P0.

New-structural: This class of test cases tests the modified or updated program constructs. They are usually created for structural coverage of the modified parts in P0.

New-specification: This class of test cases tests the new code generated from the modified parts of the requirement specifications of P0.

TEST SUITE MINIMIZATION

In this section, we review the definition of the test suite reduction problem and analysis parameters for different minimization techniques.

Background and Definition

Regression testing is the key process in the development and maintenance of emergent software. The biggest issue related to that, it requires a lot of test cases to test modified parts of the software. Figure 1 shows the steps of regression testing process. Test-suite reduction techniques try to cut the costs of saving and reusing test cases during software maintenance by removing duplicate test cases from test suites. According to Rothermal [4], a Software contains above 10,000 of LOC (Lines of Codes) requires a significant amount of time and effort to execute all the test cases [4].

To eliminate duplicate or redundant test cases as well as to optimize the regression testing process, test suite minimization approach is a must. The first formal definition of optimal test suite minimization problem introduced in 1993 by Harrold et al. [1] as follows:

Given: {t1, t2,…, tm} is the test suite TS of m test cases and {r1, r2,…, rn} is set of test case requirement that must be satisfied in order to satisfy desirable coverage of the program under test and each subsets {TS1, TS2,…, TSn} from TS are associated to one of ri's such that each test case tj belonging to TSi satisfies ri.

Problem: Find minimal test suite TS' from TS which satisfies all ri's.

The requirements in the earlier statement represent various test case requirements: branches, source statements, decisions, def-use associations, or specification items [9].

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

The main objective of test suite reduction is to optimize the regression testing process while maintaining Fault Detection Effectiveness (FDE) [5]. At the time of minimization process, it may be useful to choose the test cases that are likely to reveal faults instead of including more test cases in the reduced suite. For a particular program, a test selection criterion translates into a requirement set, whose satisfaction provides that how much this fulfill the specified criterion.

Table 1. Example of requirement coverage information of test cases in a test suite

	R1	R2	R3	R4	R5	R6	R7
T1	×		×	×			
T2		×		×			
T3		×	×			×	×
T4			×		×		
T5		×			×	×	

See Table 1 that shows an example of the coverage information of test cases in a test suite {T1, T2, T3, T4, and T5}. The symbol (×) represents satisfaction of a requirement by a test case. Here we find that a subset {T1, T3, T4} of the suite cover all the requirements {R1, R2, R3, R4, R5, R6, R7}, whereas test cases T2 and T5 become duplicate due to the requirements covered by them are fulfilled by the other three test cases T1, T3, and T5.

Analysis Parameter

This section defines some set of analysis parameters for analyzing existing algorithms for test suite reduction. We have found these analysis parameters from the referenced papers which work as a base to carry out the comparison process.

Requirements: It represents the core requirements of each test suite minimization heuristic. It works as the primary inputs which help the proposed test suite reduction algorithm to work efficiently.

Coverage Based: It represents the requirement coverage criteria for a particular reduction technique to generate the Representative Set (RS). It can be based on Control-flow (i.e., statement coverage, branch coverage, path coverage, etc.), Data-flow (i.e. def-use Association, p-use, c-use, etc.) [13] and Modified Condition/Decision Coverage also [17].

Testing Approach: It represents the method, or we can say that what type of strategies used for test suite minimization purpose.

Type of Algorithm: It represents the algorithm category at a more abstract level. It can be polynomial time, quadratic time or linear time.

Tool Used: It represents the tool support for particular Test Suite Reduction technique.

Test Suite Size Reduction (TSSR): It represents the TSSR percentage of a particular reduction method [18].

$$TSSR = \frac{|TS_{orig}| - |TS_{red}|}{|TS_{orig}|} \times 100\% \quad (1)$$

Where, $|TS_{orig}|$ and $|TS_{red}|$ represents the size of original and minimized test suite, respectively [19].

Fault Detection Capability (FDC) Loss:

It represents the percentage of FDC loss of a particular TSR heuristic [18].

$$FDCLOSS = \frac{|F_{orig}| - |F_{red}|}{|F_{orig}|} \times 100\% \quad (2)$$

Where, $|F_{orig}|$ and $|F_{red}|$ represents the total number of unique fault exposed by the original and minimized test suite, respectively.

EXISTING TEST SUITE MINIMIZATION TECHNIQUES

In this section, some well-known test suite minimization techniques or mechanism is briefly discussed. After discussion, we comparatively analyze these techniques based on chosen parameters in a further section.

Greedy Algorithm

The Greedy algorithm [20] is a well-known method for finding the optimal solution to the test suite minimization problem. This algorithm repeatedly transfers the test which covers the most unsatisfied test

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

requirements from the test suite set TS to RS until all of the requirements are satisfied or covered. Steps are given below:

Step 1: Create an empty set RS and mark all test case requirements as ''unsatisfied/uncovered''.
Step 2: Identify a test case t in TS, where t is the test case that covers the most unsatisfied test requirements.
Step 3: Move the test case t from TS to RS.
Step 4: Mark the test case requirements which are covered by t as ''satisfied/covered''.
Step 5: Repeat Steps 2–4 until all test case requirements are covered, and then return the representative set RS.

There are various existing test suite reduction methods based on the concept of the Greedy algorithm [21].

HGS Algorithm

HGS algorithm presented by Harrold et al. [22] accepts the testing sets T_i for each requirement and finds an RS that covers all requirements. It first considers the T_i's of cardinality one (single element), then places test cases that belong to these T_i into the RS and marks all Ti's containing any of these test cases. Next, the test case that occurs the most times among all T_i's of cardinality two is added into the representative set. Again, all unmarked T_i's containing these test cases are marked. It is repeated for the T_i's containing 3, 4, 5, 6… max, where max represents the maximum cardinality of the Ti's. When we examine the T_i's of cardinality m, there may be a tie, because some test cases present in the maximum number of T_i's of that size. So, the test case that occurs in the most unmarked T_i's of cardinality (m + 1) is examined when this condition arises.

If a decision cannot be made, the T_i's with the highest cardinality are recursively examined, and finally, a random choice is made. This algorithm can provide better test suite size reduction, but minimization process takes more computing effort due to recursive function calls.

GE and GRE Algorithm

Chen and Lau [23] propose two greedy heuristics called as Greedy Evolution (GE) and GRE. Here ''G'', ''R'', and ''E'' represents ''Greedy'', ''Redundant'', and ''Essential'', respectively. They categorize test cases in a test suite: the essential test cases and the 1-to-1 redundant test cases. A test case termed as essential if there exist a requirement which is only covered by that particular test case. In contrast, a test case t_i is said to be 1-to-1 redundant if there exists a test case t_j such that the set of requirements covered by t_i is a subset of the set of requirements covered by t_j.

Table 2. An example of 1-to-1 redundancy

	R1	R2	R3	R4	R5
T1	×	×		×	×
T2	×		×		

For example, Table 2 shows that T2 is 1-to-1 redundant because of the requirements of set {R1, R4} is the subset of {R1, R2, R4, R5}, which is covered by T1.The GRE algorithm is the enhanced version of GE, which includes the following strategies:

Essential Strategy: In this strategy, all essential test cases are selected first.
One-to-One redundant strategy: 1-to-1 redundant test cases are identified and removed.
Greedy strategy: This strategy will be applied to the remaining test cases that satisfy the maximum number of uncovered requirements.

It is noticed that, if both the essentials strategy and the one-to-one redundancy strategy cannot be applied only then the greedy strategy is adopted.

Delayed Greedy Algorithm

Tallam and Gupta [31] proposed a Formal Concept Analysis (FCA) based heuristic called delayed greedy algorithm. It is a hierarchical clustering based approach which takes three inputs: test cases, testing requirements, and coverage of test cases to corresponding requirements, and then finds the maximal groupings. After that, it determines a Representative Set (RS) by overlapping between test cases among their set of requirements covered.

Modified Condition/Decision Coverage based Algorithm

Modified condition/decision coverage (MC/DC) is stricter form of decision/branch coverage. Each decision statement must calculate either true or false on some execution of the software program. MC/DC pair needs to be covered to satisfy the criterion for a condition of decision coverage [17, 24]. Jones and Harrold [17] discussed two techniques for Test Suite Minimization (TSM): build-up and break-down. In a build-up technique, the algorithm begins with an empty TS' and adds test cases to it, whereas in a break-down technique, the algorithm begins with T and removes test cases from it to get TS'.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

TestFilter

TestFilter [25] is a Statement Coverage Based test suite minimization technique. This technique identifies the test cases according to their statement coverage termed as weight. Here, weight represents the total number of existences of a particular test case that satisfy different branch or statement of the program under test. Based on that, it picks non-redundant test cases very efficiently.

Step 1: The technique first calculates a Weighted Set (WS) of all generated test cases.

$$Weight(tc_k) = \sum_{i=1}^{n} Contains(Domain(TS_i), tc_k) \qquad (3)$$

Step 2: Select the first test case (tc_h) of the highest weight from the WS (Weighted Set). In the case of a tie between test cases, random selection strategy is used.

Step 3: Move tc_h to the RS, and mark all test suites from STS (Set of Test Suites), which have tc_h in their domain. If all test suites are marked then stop, otherwise go back to step 1.

Reduced with Selective Redundancy (RSR)

This heuristic [5] called Reduced with Selective Redundancy (RSR) modified the existing HGS algorithm with multiple coverage criteria for Test Suite Reduction. Before minimization, when test suite contains lots of redundancy on a coverage criterion, it may be beneficial to keep some of the duplicate or redundant test cases selectively in the reduced test suite so as to preserve more fault detection effectiveness in the minimized suite without significantly affecting the amount of test suite size reduction.

Step 1: Select test cases that satisfy the primary requirements that are currently uncovered by the reduced suite (initially empty).

Step 2: Select the next test case according to the Primary Requirement.

Step3: Mark Newly-Covered Requirements and Update Coverage Information.

Step 4: Select Redundant Test Cases. Here, redundancy may be added to the minimized test suite.

For each redundant test case on the primary criterion, additional secondary requirements of the minimized suite are generated.

Reduction with Tie-Breaking

This heuristic [13] extends the HGS algorithm [22] and the GRE algorithm [23] by incorporating two coverage criteria: branch coverage and def-use pair coverage. The primary focus is to resolve the ties among test cases during TSR, called as reduction with tie-breaking (RTB). Here, secondary criterion is used to break the tie among test cases.

Reduction using Signature Values (RSV)

This heuristic proposed four metrics: Block coverage equivalence M1, Control flow divergence M2, DU equivalence M3 and Data divergence M4. After calculating these metrics values based on their coverage, signature values is calculated. First one is Equivalence Signature Value, and another one is Divergence Signature Value. Each test case signature is an aggregate quantifiable metric that is used to identify the amount of similarity or dissimilarity of the test cases of a test case pair. A signature is a weighted average of a subset of the K metric generated for a test case pair.

$$Equivalance\ Signature = \frac{P_1 M_1 + P_3 M_3}{2} \qquad (4)$$

Where, $P_1 = P_3 = 1$ (P_x is the weight for the x^{th} metric), M_x is the x^{th} metric value for test case pair, M_1= Block coverage equivalence metric value, M_4 = DU equivalence metric value.

$$Divergence\ Signature = \frac{P_2 M_2 + P_4 M_4}{2} \qquad (5)$$

Where, $P_2 = P_4 = 1$ (P_x is the weight for the x^{th}metric), M_x is the x^{th} metric value for test case pair, M_2 = Control Flow Divergence metric value, M_4 = Data Flow Divergence metric value.

FLOWER

In this heuristic [27] Gotlieb, A., & Marijan, D. derives a completely new approach for test suite reduction, called FLOWER mainly based on a search among maximum network flows. FLOWER forms a flow network through the given information about test suite and the requirements covered by the suite. This flow network is

then traversed to find its maximum flows. FLOWER optimally uses the Ford-Fulkerson method to evaluate maximum flows and to search optimal flows it uses Constraint Programming Techniques [27].
Three steps are there:
Step1: Encoding Test Suite Reduction with a Flow Network
Step2: Finding a Representative Test Suite Using Maximum Flows
Step3: Finding a Minimal-Cardinality Subset
FLOWER obtains the same reduction rate as ILP because both approaches compute optimal solutions. When we compare this to the simple greedy approach, it takes 30% more time (average) and generates from 5% to 15% smaller test suites [27].

RZOLTAR

In this heuristic [28] Campos, Juan, and Rui Abreu proposed an approach, RZOLTAR. It creates the coverage matrix through the relation between a test case and its testing requirements. After that, it maps this matrix into a set of constraints and evaluates a collection of optimal minimal representative sets. This maintains the same coverage as the original suite by leveraging a fast constraint solver [28]. This approach is based on constraint solving programming, which efficiently minimizes the size of the test suite, maintaining full coverage and FDE.
This approach works in following steps:
Step1: It executes the system under test (SUT) with the current test suite to obtain the coverage matrix.
Step2: The coverage matrix is then transformed into a set of constraints.
Step3: The constraints are solved with MINION (off-the-shelf constraint solver), and prioritized using a particular criterion.

Bi-Criteria Test Suite Reduction

In this heuristic [29], new approach for test suite reduction is discovered that attempts to identify useful test cases by their FDE (fault detection effectiveness). This method basically works on the clustering of the test cases execution profiles. Group the test suite such that identical test cases regarding processing a particular coverage criterion would be in the same clusters. Hence, determining duplicate or redundant test cases from essential ones could be done efficiently and naturally. To improve the FDE of the reduced suite, two coverage criteria are used during the reduction process.
This proposed work is motivated by combining the two general techniques called distribution-based and coverage-based techniques.

Distribution-based

Capable of determining similar test cases using clustering. Cluster test cases within a test suite to identify test cases which process similar execution paths within the program.

Coverage-based

The main concentration is on test suites with complete coverage rather than non-overlapping test cases. Therefore, combining these two techniques, a new bi-criteria technique is created to form complete coverage minimized test suites with less overlapping in the execution profiles of the including test cases.

COMPARATIVE ANALYSIS OF EXISTING TEST SUITE MINIMIZATION TECHNIQUES

In this section, we comparatively analyze the minimization techniques based on defined parameters. We delivered this analysis in a tabular form (see Table 3), which makes the process of choosing reduction technique in future work is more comfortable.

CONCLUSION

In this paper, we performed an empirical comparison of different existing test suite minimization techniques based on some chosen parameters. The comparison targets various factors of test suite reduction. We have observed that majority of the reduction algorithms have focused on solving the single-objective TSM optimization problem, i.e., to minimize the size of RS as much as possible. We have also observed that some greedy algorithms have focused on solving the bi-objective TSM optimization problem, i.e., to significantly reduces the size of the representative test suite while maintaining the FDC as well.
Some of the techniques identify the similarity level of test cases for minimization purpose. The majority of minimization techniques have used control-flow based coverage criteria, Block/Branch coverage (Statement) and MC/DC Coverage.
We observed that in future, we could incorporate clustering and prioritization in these techniques for the optimal solution. We should focus on proposing such type of techniques that efficiently solve the multi-objective Test

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

Suite Reduction optimization problem as per the needs of current applications. Also, the automation support for the proposed algorithms for TSR can also drastically reduce the testing effort.

Table 3. Comparative Analysis of TSM Techniques

Test Suite Minimization Techniques	Analysis Parameters						
	Requirements	Coverage Based	Testing Approach	Types of algorithm	Tool Used	TSSR	FDC
Greedy Algo.	Satisfiability relation	BC	Recursive approach	Polynomial Time	NG	NG	NG
HGS Algo.	Cardinality of test suites	DUC	Based on Cardinality of test cases	Polynomial Time	NG	60%	NG
GE & GRE Algo.	Satisfiability relation	BC	Based on essential, 1-1 redundant and greedy strategies	Polynomial Time	NG	41.67%-50%	NG
Delayed Greedy Algo.	Context table	BC, DUC	FCA Based	Polynomial Time	NG	NG	NG
MC/DC Coverage Based Algo.	Truth vectors	BC, MC/DC	MC/DC Coverage	Polynomial Time	NG	92%	10.2%
TestFilter	Statement Coverage	SC	Based on weights of test cases (WS)	Polynomial Time	NG	89.8	NG
RSR	Selection of selective redundancy criterion	BC, DUC	Selectively keep some of the redundant test cases	Polynomial Time	ATAC	8.87%-87.38%	1.45%-46.43%
Bi-criteria TSR	Distribution and coverage based	Distribution and coverage based	Reduction by Cluster Analysis of Execution Profiles	Polynomial Time	ATAC	30%-87%	2.28%-55% Fault Detection Loss
RTB	Primary and Secondary Test Coverage	BC, DUC	Breaks the ties by using a secondary criterion	Polynomial Time	NG	76%-82%	16%-55%
RSV	Four different Coverage parameters	BC, DUC, Data divergence, CF divergence	Reduction based on Signature Values	Polynomial Time	NG	NG	NG
FLOWER	TSR with bipartite graph	Maximum flows in a network	Reduction based on network maximum flows.	Polynomial Time	Ford-Fulkerson method	5-15% smaller than ILP	NG
RZOLTAR	Coverage matrix	Code coverage	Minimization based on constraint solving programming		GZOLTAR	64.88% (Avg.)	Same as the original test suite

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

REFERENCES

[1] Gregg Rothermel, Mary Jean Harrold, Jeffery von Ronne, and Christie Hong, "Empirical Studies of Test Suite Reduction", Journal of Software Testing, Verification, and Reliability. 12(4):219-249, 2002.
[2] D. Binkley, "Semantics guided regression test cost reduction", IEEE Trans. Softw. Eng. 23 (8) (1997) 498–516.
[3] H. Zhong, L. Zhang, and H. Mei, "An experimental study of four typical test suite reduction techniques", Inform. Softw. Technol. 50 (6) (2008) 534–546.
[4] G. Rothermal, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing", IEEE Transactions on software Engineering, 27 (10)(2001) 929-948Tavel, P. 2007 Modelling and Simulation Design. AK Peters Ltd
[5] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction", IEEE Transactions on software Engineering, 33(2):108–123, 2007.
[6] R. A. DeMillo and A. J. Offutt, "Constraint-based automatic test data generation", IEEE Trans. on Soft. Eng.", 17(9):900–910, 1991.
[7] Gupta, Rajiv, and Mary Lou Soffa, "Employing static information in the generation of test case", Software Testing, Verification and Reliability 3.1 (1993): 29-48.
[8] T.Y. Chen, and M.F. Lau, "A Simulation Study on Some Heuristics for Test Suite Reduction", Journal of Information and Software Technology, vol. 40, issue 13, pp. 777-787, 1998.
[9] J.V. Ronne, "Test Suite Minimization: An Empirical Investigation", Bachelors Thesis, June 1999. Retrieved from URL:http://www.ics.uci.edu/jronne/pubs/jvronne uhc-thesis.pdf
[10] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies", IEEE Transactions on Software Engineering28 (2) (2002) 159–182.
[11] T. Graves, M. Harrold, J. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques", ACM Transactions on Software Engineering and Methodology (TOSEM) 10 (2)(2001) 184–208.
[12] Zhong, Hao, Lu Zhang, and Hong Mei, "An experimental comparison of four test suite reduction techniques", Proceedings of the 28th international conference on Software engineering.ACM, 2006.
[13] J.W. Lin, and C.Y. Huang, "Analysis of Test Suite Reduction with Enhanced Tie-Breaking Techniques", Journal of Information and Software Technology, vol. 51, issue 4, pp. 679–690, 2009.
[14] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites", International Conference on Software Maintenance, pages 34–43, November 1998.
[15] F.I. Vokolos, and P.G. Frankl, "Empirical evaluation of the textual differencing regression testing technique", in Proceedings of 14thInternational Conference on Software Maintenance, 1998, pp.44–53.
[16] C. Sharma, S. Singh, "Mechanism for identification of duplicate test cases", In Recent Advances and Innovations in Engineering (ICRAIE), 2014 (pp. 1-5). IEEE.
[17] J.A. Jones, and M.J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", IEEE Transactions on Software Engineering, vol. 29, issue 3, pp. 195-209, 2003.
[18] W.E. Wong, J.R. Horgan, S. London, and A.P. Mathur "Effect of Test Set Minimization on Fault Detection Effectiveness of the All-Uses Criterion", Proceedings of the 17th International Conference on Software Engineering (ICSE'94), pp. 41–50, 1994.
[19] Khan, Saif Ur Rehman, et al. "An Analysis of the Code Coverage-based Greedy Algorithms for Test Suite Reduction". The Second International Conference on Informatics Engineering & Information Science (ICIEIS2013),The Society of Digital Information and Wireless Communication, 2013.
[20] Chvatal, Vasek., "A greedy heuristic for the set-covering problem." Mathematics of operations research 4.3 (1979): 233-235.
[21] S. Yoo, and M. Harman, "Regression testing minimization, selection and prioritization: a survey", Softw. Test., Verif. Reliab. 22 (2) (2012) 67–120.
[22] M.J.Harrold, R. Gupta, and M .L. Soffa, "A methodology for controlling the size of a test suite".ACM Transactions on Software Engineering and Methodology, 2 (3) (1993) 270–285.
[23] T.Y. Chen ,and M.F. Lau, "A New Heuristic for Test Suite Reduction", Journal of Information and Software Technology, vol. 40, issues 5–6, pp. 347-354, 1998.
[24] J. J. Chilenski, and S. P. Miller, "Applicability of modified condition/decision coverage to software testing. Softw. Eng. Journal, 9(5):193–200, 1994.
[25] S.U.R. Khan, A. Nadeem, and A. Awais, "TestFilter: A Statement Coverage based Test Case Reduction Technique", Proceeding of 10th IEEE International Multitopic Conference (INMIC'06), pp. 275-280, 2006.
[26] H. K. N. Leung and L. White, "Insight into regression testing", In Proceedings of Interntional Conference on Software Maintenance (ICSM 1989). pages 60{69. IEEE Computer Society Press, October 1989.
[27] Gotlieb, Arnaud, and Dusica Marijan, "FLOWER: optimal test suite reduction as a network maximum flow", Proceedings of the 2014 International Symposium on Software Testing and Analysis, ACM, 2014.
[28] Campos, Juan, and Rui Abreu, "Leveraging a Constraint Solver for Minimizing Test Suites, "Quality Software (QSIC), 2013 13th International Conference" on IEEE, 2013.
[29] Khalilian, Alireza, and SaeedParsa. "Bi-criteria test suite reduction by cluster analysis of execution profiles", Advances in Software Engineering Techniques, Springer Berlin Heidelberg, 2012, 243-256.
[30] Usaola, Macario Polo, Pedro Reales Mateo, and Beatriz Pérez Lamancha, "Reduction of test suites using mutation", Fundamental Approaches to Software Engineering, Springer Berlin Heidelberg, 2012, 425-438.
[31] S. Tallam, and N. Gupta "A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization", ACM SIGSOFT Software Engineering Notes, vol. 31, issue 1, pp. 35-42, 2006.