Asian Journal of Computer Science and Technology
ISSN: 2249-0701 (P) Vol.7 No.2, 2018, pp.87-91

DOI: https://doi.org/10.51983/ajcst-2018.7.2.1865

A Sound Assessment of Test Suite Minimization Techniques

Fayaz Ahmad Khan
Department of Computer Science and Applications, Barkatullah University, Bhopal, Madhya Pradesh, India
E-Mail: kfayaz1012@gmail.com

(Received 6 July 2018; Revised 19 July 2018; Accepted 5 August 2018; Available online 12 August 2018)

***Abstract* - During software development, testing and re-testing occurs frequently to ensure that the software is working correctly before and after modifications. To carry out an effective testing process a test suite is created and executed to detect the faults in the existing code as well as in the modified code. The manual approach of test suite creation and execution is time consuming and labour intensive task as compared to automatically generated test data or test suite. The automatic test data generation is supposed to be an effective way, but a lot of redundant test cases are generated that increase the time, effort and cost of testing. Therefore, test suite minimization techniques are used to further minimize or reduce the number of test cases by selecting a subset from an initially random and large test suite to test the code before as well as after modification. In this study, a comprehensive analysis of the different test suite minimization techniques is presented in order to extend the existing studies and to propose new ideas in this direction.**

***Keywords:* Software Testing, Regression Testing, Test Suite Minimization**

I. INTRODUCTION

Due to the application of automated test data generation techniques [1-10], a large number of test cases are generated which are redundant and execute the same requirements multiple times. Automatic test data algorithms are usually exhaustive with respect to the specified coverage criteria. Therefore, depending on the enormity of the model, these algorithms or the tools based on them create a huge number of test cases that are infeasible to be considered for practical execution [1-10]. Also, most of the test cases can be redundant in the sense of exercising common features of the code under test (for example, the same lines of code) and revealing same sets of defects [11]. So, most of the times, running an entire automatically generated test suite is not an adequate way as it will consume a significant amount of time and resources during regression testing. Also, in software testing, a single test case from an infinite input domain of program variables can hardly satisfy all the test requirements. Therefore, to find an optimal or sub set of test cases from usually large infinite domain of test cases that can satisfy the same requirements as the original test suite is a research problem commonly known as test suite reduction or test suite minimization [11]. The obvious reasons that make test cases redundant are (1) due to changes or modifications in code their input/output relation is no longer remains meaningful, (2) these inputs were generated for a specific program that has undergone modification or (3) their structure is no longer in compliance with the software coverage [12, 13]. Harrold and Gupta in [11] defined the test suite minimization as follows:

Given: A test suite ST, a set of test requirements {r1,r2 . . . , r(n)}, that must be satisfied to provide the desired testing coverage of the program, and subsets of ST, T1, . . . , T(n), one associated with each of the r_i's such that any one of the test cases t_j belonging to Ti can be used to test r_i's.

Problem: Find a representative set, T', of test cases from T that satisfies all r_i's.

The testing criterion is satisfied when every test requirement in {r1, r2. . . , r(n)} is satisfied. A test requirement, r_i, is satisfied by any test case, t_j, that belongs to the Ti, a subset of T [11]. Trying to find the minimal hitting set of a test suite that covers the same set of requirements covered by the original test suite is a NP-complete problem [11]. NP-completeness of the test suite minimization problem encourages the usage of different techniques based on heuristics, genetic algorithm and integer linear programming [11]. The basic classification of test suite minimization techniques is shown in figure 1.

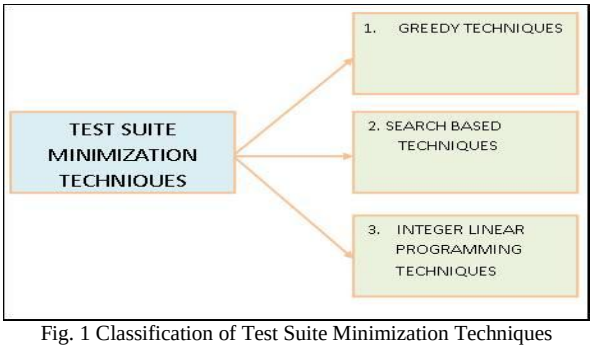

Fig. 1 Classification of Test Suite Minimization Techniques

A. Greedy Techniques

The most widely used test suite reduction or minimization techniques are greedy in nature [11, 14, 15]. Greedy techniques make locally optimal choice of test cases with a hope that the choice will be globally optimal one. The following are some of the important techniques developed for the minimization of test suites:

In 1993, Harrold *et al.* [11] proposed a greedy heuristic algorithm that help in managing the test suite by identifying and removing the obsolete test cases with respect to a given set of test requirements. The goal of the technique is to find

the essential test cases form the test suite that when removed some test requirements can never be fulfilled [11]. The algorithm begins by computing cardinality of all the test cases and then, selects the test cases in increasing order of the cardinalities until all requirements are satisfied.

In 1993 and in1998, Wong *et al.* [16, 17], conducted a number of empirical experiments to show the effect of minimization on the fault detection effectiveness of reduced test suites. Wong et al; performs minimization with respect to all-uses criterion and their results confirms that significant test suite reduction can be achieved with a little to no loss in fault detection effectiveness.

In 1996, Chen and Lau [18] proposed another greedy heuristic algorithm for test suite minimization based on the mix of three strategies: the greedy strategy, the essential strategy, and 1-1 redundancy strategy. The greedy strategy proceeds by selecting test cases that have satisfied the maximum quantity of not-yet satisfied test requirements [18]. The essential strategy is used to select the essential test cases and the 1-1 strategy is used to remove all the 1-1 redundant test cases. In each step, all the three strategies select one test case and the selection process continues when all the test requirements are satisfied.

In 1998 and 2002, Rothermel et al in [19, 20], also conducted different experiments to study the impact of test suite minimization on the fault detection effectiveness loss. The experiments on Siemens suite showed that the size of the test suite is significantly reduced but at the cost of decreased fault detection effectiveness. The experiments conducted by Wong *et al.* [16, 17] and Rothermel et al [19, 20] differ in the following number ways:

1. The subject programs in both the studies were different.
2. The minimization approaches in both studies were also different.
3. The requirement criterion was also different. Rothermel *et al.* uses edge coverage while Wong *et al.* uses all uses criteria for minimization.
4. The test case generation techniques were also distinct, Rothermel *et al.* generated test cases using various white box and Black-box testing criteria while as Wong *et al.* uses a random generated test suite.

In 2005, Tallam and Gupta [21], proposed another greedy heuristic approach known as Delayed-greedy for the test suite minimization. The Delayed-greedy approach exploits both the implications among the test cases and the implications among the requirements and then removes the implied rows and columns in the table. A potential weakness of the traditional greedy approach is that the early selection made by them can eventually be rendered redundant by test cases subsequently selected. For example, consider the table 1, which shows the relation between test cases and test requirements. The traditional greedy approaches will choose **t1** first as it satisfied the maximum number of testing requirements, and then continues to select t2, t3 and t4. However, after the selection of t2, t3 and t4, t1 is considered redundant. Tallam and Gupta tried to overcome this potential weakness by developing a concept lattice using delayed greedy approach. The delayed-greedy performs in three main phases:

1. Applies object reductions (i.e., remove test cases which test requirements is subsumed by other test cases).
2. Apply attribute reductions (i.e., remove test requirements that are not in the minimal requirement set) and
3. Build reduced test suite from the remaining test cases using a greedy method [11].

The minimized test suite by this approach will be either the same size or smaller than those minimized using traditional greedy approaches.

TABLE 1 RELATIONSHIP MATRIX (BETWEEN TEST CASES AND REQUIREMENTS) [62]

Test cases	Test Requirements					
	r1	r2	r3	r4	r5	r6
t1	X	X	X			
t2	X			X		
t3		X			X	
t4			X			X
t5					X	

In 2005, Jeffrey and Guppta [21], proposed a modified form HGS algorithm [11] known as Reduction with Selective Redundancy in order to retain redundant test cases to ensure that reduction did not compromise the fault detection ability of the minimized test suite. The proposed algorithm deals with the limitations of traditional single-criterion minimization techniques by taking in account several sets of testing demands (e.g., coverage of different entities) and introducing selective redundancy in the minimized test suites. For example see Table 2, Jeffrey and Gupta [21] tried to generate a satisfactory minimized branch coverage test suite by keeping some redundant test cases and collect the coverage information for secondary criteria, such as the all def-use pair's criteria, for all the test cases in the test suite T.

TABLE II BRACH COVERAGE INFORMATION FOR TEST CASES IN [63]

TEST CASES	B_1^T	B_1^F	B_2^T	B_2^F	B_3^T	B_3^F	B_4^T	B_4^F
t1	X		X			X		
t2		X		X	X			X
t3		X	X			X		
t4		X	X		X		X	
t5		X		X	X		X	

In the example presented by Jeffrey *et al.* [21], after inserting t1 and t2 in the minimized suite by HGS algorithm [11], t3 is described as redundant with respect to branch coverage since all the branches covered by t3 are already covered by t1 and t2. Jeffrey *et al.* [21] did not throw away t3, but verify it with respect to the secondary coverage criteria and do not identify t3 as redundant and add it to the minimized test suite. Next, either t4 or t5 needs to be chosen to cover the branch BT4 and finally, they added t4 to the constituted test suite and mark the requirement BT4 as coverage. Therefore, the final minimized test suite generated by their approach for this example is {t1, t2, t3, t4} as they identified t5 redundant with respect to both the adequacy criterions. Jeffrey *et al.* in [63], proceeded with an empirical evaluation using branch coverage as the first set of testing requirements and all-uses coverage information obtained by data-flow analysis.

In 2007, Scott & Atif [22] defined a new metric for coverage based test suite reduction based on the average probability of detecting each fault. As most approaches to test suite reduction are based on eliminating test cases redundant with respect to some coverage criterion. Consequently, no existing procedures determine the likelihood of various coverage criteria to force coverage-based reduction that retains test cases in order to expose specific set of faults.

In 2008, Sampath et al [23] used the concept lattice to identify reduced test set that provide the same coverage as the original one. They presented three strategies, including the tie-breaker concept, for integrating customized usage-based test requirements with existing test requirements to increase the effectiveness of reduced test suites. However, the reported work is more specific for web application testing. In their work, they also proposed a metric called Fault Detection Density (FDD), to understand the spread of how many faults are detected with the test cases.

In 2009, Jun & Chin [24] proposed a new technique for test suite reduction called Reduction with Tie-Breaking (RTB). The techniques uses additional criterion to break the ties during the reduction process. In their study, it was also suggested that all existing test suite minimization techniques could be integrated into this framework through the proposed decision process in order to produce more effective results.

In 2010, Saeed & Alireza [25], proposed a Bi-Objective Greedy (BOG) algorithm that addresses the fault detection effectiveness. The proposed technique begins with the construction of test case requirement matrix and then it is multiplied by its transposed matrix to generate the multiplied matrix. The diagonal elements denote the number of unmarked requirements between the test cases while the non-diagonal elements represent the minimum overlap in requirement coverage with other test cases in the matrix [25]. In next step the diagonal elements of the multiplied matrix are updated for unselected test cases and if the diagonal value of the matrix become'0' the corresponding test case is discarded and is considered redundant [25]. In this way the algorithm finally selected most favorable test cases that detect faults until the coverage of the test requirements was achieved.

II. SEARCH BASED TECHNIQUES

Search Based Software Engineering is an emerging paradigm in which search based optimization algorithms are used to balance multiple software engineering objectives. Search based software engineering techniques have been used to solve various issues associated with software testing McMinn [26, 27].

Local search techniques get trapped in local optima, so global search techniques also known as "Evolutionary Testing" approaches like Genetic Algorithms have been considered and applied in various software testing domains [26, 27]. They are characterized by iterative procedures that work in parallel on a number of potential solutions for a population of individuals. Genetic Algorithms differ from local search techniques in that they maintain a population of candidate solutions rather than just one solution [27]. The population is, therefore, capable of sampling many points at once and, thus, is more robust to entrapment in local optima. In 1999, Mansour & El-Fakih [27], adapted a hybrid genetic algorithm to the test suite reduction problem.

In 2005, Ma *et al.* [28] investigated the application of Genetic Algorithm for the test suit minimization problem. Ma *et al.* instead of using code coverage criterions, proposed cost-aware criterion that is the combination of a block based coverage criterion and a test-execution cost criterion for test suite reduction. However, it leads to the loss as the complexity of the test increases.

In 2011, Arvinder & Shivangi [29], proposed a Bee Colony Optimization algorithm using the concept of worker bees, scout and forager bees for maximum fault coverage of the test suites. In their work, Average Percentage of Fault Detection (APFD) metric was also proposed.

A. Integer Programming Language Based Techniques

In 2004, an improved effort was proposed by Black *et al.* [30] to overtake the limitation of existing approaches. Black *et al.* [30] approach is based on two objectives: minimization up to a particular coverage while simultaneously trying to enhance the fault detection rate with respect to a specific fault using an integer linear programming solver. The main idea behind their approach is to formulate the minimization as a binary linear programming problem with a weighting factor that will determine the degree to which each of the two objectives contribute influence toward the final outcome. But, the major limitations of Black et al [30] approach is that its scope is limited to few problems and the fault detection was also concerned to a single fault (rather than several faults),

and there may be limited confidence that the minimized test suite be useful in detecting several other faults.

In 2009, Hsu and Orso [31] such as Black *et al.* [30] also observed the fact that the majority of the proposed minimization techniques have two limitations: they perform minimization based upon a single criterion and produce approximated suboptimal solution. Hsu and Orso [31] extended the work of Black *et al.* [30] using a multi-criteria ILP formulation: the weighted-sum approach, the prioritized optimization and a hybrid approach. The possible weakness of these approaches is that they require additional input from the user in the forms of weighting coefficients or priority assignment, which might be biased, unavailable or costly to provide.

The other useful technique proposed in 2012, by Kapfhammer et al [32] for database applications testing. Database applications are commonly implemented and used in both industry and academia. Database applications are complex and rapidly evolving applications and often undergo rapid changes in the source code of the program and the state and structure of the database [32]. So to avoid any bad effect due to frequent changes in code, the approach removes all the redundancy from the test suite before applying it [32].

In 2013, Loreto et al [33] also proposed a technique based on post optimization algorithm. Loreto et al represents the test suite interaction using a covering array and the objective was to reduce the number of rows in the array. When wild cards were detected, the rows in the covering array were merged. In this way, the rows in the covering array were reduced which ultimately reduce the entire test suite size.

In 2014, Khan *et al.* [34, 35, 36] utilized data clustering algorithms for the test suite minimization. The goal of the studies [34, 35, 36] is to apply data clustering algorithms to partition the test suite into multiple partition or segments as the initial test suite is found to be redundant and large in size. Khan et al [34, 35, 36] claim to have achieved minimization through data clustering by grouping the similar or redundant test cases into the appropriate clusters and then selecting a single test case from each of the clusters to create a subset from the whole test suite. But the main drawback of the study [34, 35, 36] is that the minimization or reduction was carried-out without taking the code coverage criteria into consideration.

In 2017, Khan *et al.* [37] proposed an efficient heuristic based test suite minimization approach that takes multiple code coverage criteria's into consideration. The minimization was carried out in [37] on statement coverage, branch coverage and independent path coverage matrices individually and after that the union of all the test suite is calculated to form a single minimized test suite and to further remove the redundant test cases. But, the drawback of the study [37] is that it calculates the representative subset from each matrix individually and also, it at any instant generates the subset from the top-down fashion while leaving other test cases with un-noticed. Therefore, further work in this direction is possible to enhance the working of the proposed approach for better results.

III. CONCLUSION AND FUTURE DIRECTIONS

Automatic test data generation techniques generate lot of test cases which are found redundant and are infeasible to be considered for practical execution. Hence, due to automatic test data generation and limited time in regression testing, running an entire test suite is not considered an adequate practice due to time and resource constraints. Therefore, in this study a detailed discussion on the test suite minimization problem, its need and different techniques is presented. Further work in this direction can be extended in order to further improve the existing techniques discussed in this study and new techniques could also be proposed to get the better results in order to reduce the time, effort and cost in regression testing.

REFERENCES

[1] A. K Gupta and F. A. Khan, "An Efficient Test Data Generation Approach For Unit Testing", *IOSR (JCE)*, Vol. 18, No. 4, Ver. V, pp. 97-107, 2016.

[2] B. Korel, "Automated software test data generation", *IEEE Transactions on Software Engineering*, Vol. 16, No. 8, pp. 870–879, 1990.

[3] D. Cohen, I. C. Society, S R Dalal, M L Fredman and G C Patton, "The aetg system: An approach to testing based on combinatorial design", *IEEE Transactions on Software Engineering*, Vol. 23, pp. 437–444, 1997.

[4] R. Lammel and W. Schulte, "Controllable combinatorial coverage in grammar-based testing", *In TestCom*, pp. 19–38, 2006.

[5] P. Purdom, "A sentence generator for testing parsers", *BIT*, Vol. 12, No. 3, pp. 366– 375, 1972.

[6] Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Artoo, adaptive random testing for object-oriented software", *In 30th International Conference on Software Engineering*, pp. 71–80, 2008.

[7] C. Csallner and Y. Smaragdakis, "Jcrasher, An automatic robustness tester for java", *Software Practice Experimentation*, Vol. 34, No. 11, pp. 1025–1050, 2004.

[8] C. Pacheco and M. D. Ernst, "Eclat: Automatic generation and classification of test inputs", *In ECOOP*, pp. 504–527, 2005.

[9] W. Visser, C. S. Pasareanu and S. Khurshid, "Test input generation with java pathfinder", *In ISSTA*, pp. 97–107, 2004.

[10] T. Xie, D. Notkin, "Automatically identifying special and common unit tests for object-oriented programs", *In ISSRE*, pp. 277–287, 2005.

[11] 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*, Vol. 2, No. 3, pp. 270– 285, 1993.

[12] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer and R. S. Roos, "Time aware test suite prioritization", *in International Symposium on Software Testing and Analysis (ISSTA 06). Portland, Maine, USA*, ACM Press, pp. 1–12, 2006.

[13] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection", *in International Symposium on Software Testing and Analysis (ISSTA'07)*, ACM Press, pp.140–150, July 2007.

[14] T. Y. Chen and M. Lau, "Dividing strategies for the optimization of a test suite", *Information Processing Letters*, Vol. 60, No. 3, pp. 135–141, 1996.

[15] J. W. Lin and C. Yu. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques", *Journal of information and software technology*, Vol.51, pp. 679-690, 2009.

[16] W. E. Wong, J. R. Horgan and A. P. Mathur, "Effect of test set minimization on the fault detection effectiveness", *Software Practice and Experience*, Vol. 28, No. 4, pp. 347-369, 1999.

[17] W. E. Wong, J. R. Horgan, A. P. Mathur and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application", *Journal of Systems and Software*, Vol. 48, No. 2, pp. 79-89, 1999.

[18] 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", *In Proceedings of the International Conference on Software Maintenance, ICSM*, Washington, DC, USA, pp. 34–43, 1998.

[19] G. Rothermel, M. J. Harrold, J. V. Ronne and C. Hong "Empirical Studies of Test-Suite Reduction", In *Journal of Software Testing, Verification, and Reliability*, Vol. 12, No.4, pp. 219-249, 2002.

[20] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization", *SIGSOFT Software Engineering Notes*, Vol. 31, pp. 35–42, 2005.

[21] D. Jeffrey and N. Gupta, "Test suite reduction with selective redundancy", *In Proceedings of the 21st IEEE International Conference on Software Maintenance*, *IEEE Computer Society*, pp. 549–558, 2005.

[22] M. C. Scott Master and A. Memon, "Fault detection probability analysis for coverage based test suite reduction", *in proceedings of the 21st IEEE International Conference on Software Maintenance*, 2007, pp. 335-344.

[23] D. Sampath, R. Bryce, G. Viswanath, V. Kandimalla, A. Gunes, "Prioritizing User- Session bases test cases for Web Application Testing", *International Conference on Software Testing verification and validation (ICST)*, pp.141-150, 2008.

[24] P. Saeed, K. Alireza, "On the optimization approach towards test suite minimization", *International Journal of Software Engineering and its Applications*, Vol. 1, No. 4, pp. 15-28, 2010.

[25] P. McMinn, "Search-based software test data generation: a survey", *Software Testing Verification and Reliability*, Vol. 14, No. 2, pp. 105-156, 2004.

[26] M. Harman, "The Current State and Future of Search Based Software Engineering", *In Proceedings of the 29th IEEE International Conference on Software Engineering (ICSE 2007)*, Minneapolis, USA, Vol. 34, No. 5, pp. 342–357, 2007.

[27] N. Mansour and K. El- Fakih, "Simulated annealing and genetic algorithms for optimal regression testing", *Journal of Software Maintenance*, Vol.1, No.1, pp. 19-34.

[28] X. Y. Ma, Z. F. He, B. K. Sheng, and C. Q. Ye, "A Genetic Algorithm for Test-Suite Reduction", *IEEE International Conference on Systems, Man and Cybernetics*, Vol.1, pp. 133- 139, 2005.

[29] K. Arvinder, G. Shivagi, "A Bee Colony optimization algorithm for fault coverage based regression test suite prioritization", *International Journal of Advanced Science and Technology*, Vol. 29, pp. 17-30, 2011.

[30] J. Black and E. Melachrinoudis e David Kaeli, "Bi-criteria models for All uses test suite reduction", *In Proceedings of the 26th International Conference on Software Engineering, ICSE '04*, IEEE Computer Society, pp. 106–115, 2004.

[31] H. You Hsu and A. Orso, "MINTS: A general framework and tool for supporting test-suite minimization", *IEEE Computer Society*, pp. 419–429, 2009.

[32] GM. Kapfhammer, "Towards a method for reducing the test suites of database applications", *Proceedings of the 5th International Conference on Software Testing, Verification and Validation*, (ICST 2012), pp.964-965, 2012.

[33] G. H. Loreto, T. J. Jose, R. V. Nelson and B. R. Josue, "A Post-optimization strategy for combinatorial testing: test suite reduction through the identification of wild cards and merge of rows", *Advances in Computational Intelligence Lecture Notes in Computer Science*, Vol. 7630, pp. 127-138, 2013.

[34] F.A. Khan, A.K. Gupta and D.J. Bora. "Profiling of Test Cases with Clustering Methodology", *International Journal of Computer Applications* Vol.106, No. 14, pp. 32-37, November, 2014.

[35] F.A. Khan, A.K. Gupta and D.J. Bora, "An Efficient Approach to Test Suite Minimization for 100% Decision Coverage Criteria using K-Means Clustering Approach", *IJAPRR*, Vol. II, No. VII, pp. 18-26, 2015.

[36] F.A. Khan, A.K. Gupta and D.J. Bora, "An Efficient Technique to Test Suite Minimization using Hierarchical Clustering Approach", *International Journal of Emerging Science and Engineering (IJESE)*, ISSN: 2319–6378, Vol. 3, No. 11, September, 2015.

[37] F.A. Khan, A.K. Gupta and D.J. Bora, "An Efficient Heuristic Based Test Suite Minimization Approach", *Indian Journal of Science and Technology*, Vol. 10, No. 29, pp. 1-8, August, 2017.