International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 18 (2015) pp 39841-39848
© Research India Publications. http://www.ripublication.com

Test Suite Reduction Mechanisms: A Survey

A.D.Shrivathsan

Assistant Professor Deparment of Computer Applications School of Computing SASTRA Univeristy Thanjavaur-613401 Tamilnadu India a.d.shrivathsan@gmail.com

K.S.Ravichandran

Associate Dean Department of ICT School of Computing SASTRA University Thanjavur-613401 Tamilnadu India raviks@it.sastra.edu

K.R.Sekar

Assistant Professor Department of CSE School of Computing SASTRA University Thanjavur-613401 Tamilnadu India sekar_kr@cse.sastra.edu

Abstract
Software testing is playing a paramount role in software development, release and maintenance. Test suite length reduction without compromising on detecting faults improves efficiency. To achieve this, few test cases of the test suite needs to be eliminated. And, it is called as Test Suite Reduction. This scenario occurs mainly due to redundancy in coverage of requirements as well as code. And the elimination of such redundancies is called as reduction mechanisms. This is a breakthrough in software testing activities, as it reduces the time and effort consumption. This is an attempt to achieve software test optimization, which improves efficiency in testing without compromising on efficaciousness. There are other mechanisms available to achieve optimum amount of test cases. Such mechanisms are Test Case Prioritization and Test Case Selection. The need to strive for optimization is the time restriction in delivery of software product. This paper focuses and reviews the research articles pertaining to the Test Suite Reduction mechanisms. The objective of this study is to scrutinize what kind of techniques employed in test suite reduction. We searched the following electronic databases: Science Direct, Springer, and IEEE Explore. This paper attempts to make a literature survey of test suite reduction mechanisms attempted by various researchers.

Key terms Test suite Reduction, Test suite minimization, Test Case, Test Suite, Bug detection

1. Introduction
Software systems verification and validation activities aremeant to achieve quality. Testing is one among the activity. Meanwhile, the testing has no stopping point. Hence, the test manager decides to freeze the testing process at one stage. This decision ispurely subjective and is based on the resources availability and especially the time. The time is a crucial factor in the testing process. Removing the test cases, which are focusing on the already covered bugs is mandatory for efficiency increase. Also, eliminating a test case, when it is repeated in some other test suites will improve test efficiency. The two aforesaid activities are called as reduction or minimization in test suite.
Bug identification with less time and less effortwithout compromising on quality is a noteworthy aspect. Hence, this aspect is taken for consideration by researchers, subsequently contributing to theimprovement in testing process. The reduction approaches are based on various factors such as Requirements specification, Source code, Fault coverage, Design models, Execution profiles, and so on. This reduction is carried out in system integration test as well as in regression test. And, at both stages, reduction is very much necessary to optimize the testing efficiency.
This paper makes a literature survey on various research articles focusing on test suite reduction. Section 2 describes the rationale behind software testing, its various types and the need for test suite reduction. Section 3 describes the research articles proposed by various researchers having various methodologies in test suite minimization. Techniques such as genetic algorithm, clustering, heuristics, set theory, evolutionary algorithms, fuzzy techniques and so on are covered. The conclusion is made available in Section 4.

2. Background
Software testing happens prior to the release of product. It is to ensure that the system functions as expected by the customer. The requirements are mapped with system and defects if any are identified. Without testing, quality cannot be ascertained. Testing will not remove faults, rather it detects it. These defects or failures should be fixed.
The system built is thoroughly checked for bugs against requirements. This activity is called System test. Some are calling this as system integration test. When bugs are arising, it is handed over to the development team to fix them. This process is known as Debugging. Figure 1 shows various layers in software testing.

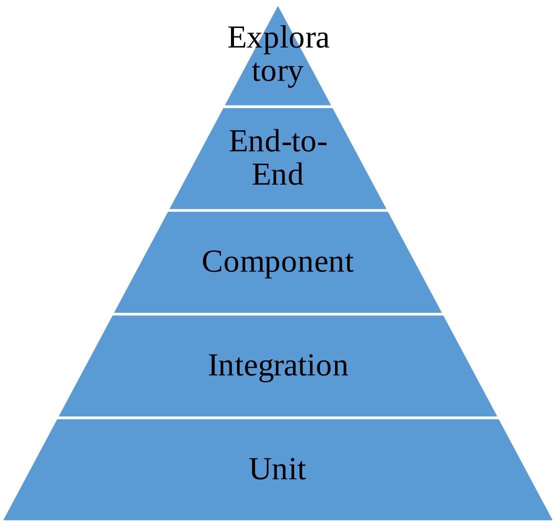

Fig 1- Testing Layers

Debugging leads to modifications made in any or all of the following: Requirement, Design, and Code. Hence, it becomes mandatory to test the modified areas of the system in subsequent testing cycles.Certain other parts of the system would have been impacted because of the changes made. Also, added or modified code needs to undergo testing. Therefore, some of the previously exercised test cases and newly generated test cases have to be exercised to identify further bugs. This process is known as Regression testing. At every iteration of regression test, new set of errors identified, and thus goes into vicious cycle. To stop this regression testing process at one stage, time and resources are used as criterion.
Another issue to be focused is, the efficiency of testing. There is always a thrust to identify high amount of defects with less amount of test cases to save time and energy. To achieve this optimality, three major techniques proposed by researchers are: a) Test Case Prioritization b) Test Suite Reduction and c) Test Case Selection
This paper does literature survey on Test Suite Reduction based research articles.

Definition:
The Test Suite Reduction problem may be stated as:

Given :
Test suite TS having test cases t_i, (i=1, 2, …, n), having given set of system requirements Rconsisting of requirements r_j, (j=1, 2, …, m), every r_j be satisfied by at least one t_i.

Problem:
$TS = \{t_1, t_2, \ldots t_n\}$ and $R = \{r_{1,} r_2, \ldots r_n\}$ find$TS\square \subset TS$such that $TS^{\square}$ satisfies all $r_i s$and $(\forall TS\square\square)$,
$(TS\square\square \subset TS)(TS\square\square$satisfies all$r_i s)$ $[|TS\square\square| \leq |TS\square|]$

Objective:
To find a subset of TS, signifya representative set RS, to satisfying all requirements.

3. Test Suite Reduction Mechanisms

3.1 Condition based reduction

James A. jones, Mary Jean Harrold[1] have done reduction of test suite and prioritization in test cases based on the conditions modified and Decisions being covered. The test cases that cover all the truth and false vectors as well as their contributions to find number of decisions covered, are considered in this approach.They have devised break down algorithm and build up algorithm for reduction, and number of entities covered determined prioritization.

3.2 Greedy and heuristic based reduction

Chu-Ti Lin, Kai-Wei Tang, Gregory M. Kapfhammer [2] devised two algorithms namely, Greedy Redundant algorithm and Greedy Redundant Essential algorithm. The approach in this is purely black box. i.e., they assessed the coverage of requirements by test cases. Accordingly test suite reductions are done. They formulated metrics, Irreplaceability and Extended Irreplaceability based on cost factor in test cases. These were used in reduction methodology.
HaoZhong, Lu Zhang, and Hong Mei [3] compared four test suite reduction techniques in terms of scalability with respect to complexity of test cases, representative set sizes and common subset amongst them. The first technique by harrold et al., is heuristic approach towards covering requirements by inputs.Chen and Lau applied Greedy Redundant Essential algorithm, which works based on requirements mapped by test case to reduce test cases.Mansour, El-Fakin employed hybrid geneticalgorithm to achieve reduction in suite size.The Black et al., 'sapproach uses IntegerLinear programming models. One of the model is to minimize amount of test cases. In the other model, two objectives are considered, which balances between minimal representative set and prominent test cases error revealing capacity.
Jun-Wei Lin, Chin-Yu Huang [4] developed Reduction with Tie Breaking approach. In this, when tie occurs between test cases in terms of coverage, then the test case containing most definition use pairs is selected.And, this approach was integrated with GRE and HGSalgorithms to improve efficacy in test case reduction.
T.Y. Chen, M.F. Lau [5] presented a heuristic GRE based on strategies like Greedy, Redundancy, and Essentials.
The essentials strategy is applied in the beginning.Essential strategy is finding test cases based on its ability to satisfy requirements. This process is repeated with each set of requirements until all essential test cases are identified. Essential strategy is to select only the indispensable test cases for covering requirement and is represented by $E_{ss}$and, $Opt\_Rep(S)$ = $Opt\_Rep(S\ (T \setminus (E\_SS,)\ R \setminus Req(E\_SS)\ )) \oplus \{E_{ss}\}$.Unless all requirements are covered by essential test cases, the sufficiency does not lie with this strategy alone.
In Greedy approach, selecting all test cases which are essential in every set is made.This may end up in redundancy.So, an enhanced version called GE strategy established, in which the indispensable test cases are selected in first step and as a next step heuristic named greedy is applied.1-1 redundancy strategy aims only at satisfiability relation reduction and incorporated with other strategies.The strategy named Redundancy removes one-to-one redundant

test cases one at a time, repeatedly until no one-to-one redundant test cases are left with. After this strategy, $Opt\_Rep(S\ (T \setminus \{t\_(1-1)\ \}, R)) \subseteq Opt\_Rep(S)$ also, a 1-to-1 redundant test case not handling high amount of requirements is not picked by greedy strategy. Moreover, if a 1-to-1 redundant test case satisfies a set of requirements, the same set of requirements might have been satisfied by another test case, say t_2. In this case, the test cases belonging toS(T, R) and $S(T \setminus \{t_{1-1}\}, R)$ by applying greedy are one and the same.One to one redundancy strategy aims to reduce the satisfiability relation and need to be incorporated with other strategies.

The Greedy strategy reduces satisfiability relation between requirements and test cases, ignoring 1-to-1 redundancy in test cases on the go.So, one to one redundancy strategy need not be combined with greedy strategy. Hence, heuristic GRE works as follows: As a first step, the essential strategy is applied and then the one to one redundancy strategy is applied repeatedly, removing one test case at a time. Finally the greedy is put into force, when the above two strategies cannot be used. And gives an optimal representative set of test cases.

TsongYueh Chen, and Man Fai Lau [6] developed a dividing strategy based on essentiality and redundancy of test cases towards coverage of requirements. In this approach, they decomposed the problem into smaller sub problems. Optimality in test suite is found in each sub problem and reconstructed it for the original problem.

The given problem is divided into k sub problems. To reconstruct the optimal solution from the decomposed sub problem, the test cases set T is divided into k (≥ 2) mutually disjoint sub sets T_1, T_2, …, $T_k$in such a way that all requirements be satisfied by T_i(i=1, 2,..., k), and these are mutually disjoint.

The strategy called essential dividing makes$S(T, R)$divided into $S(E, Req(E))$ and $S(U, R_U)$Where, E is the essential set of test cases T with respect to$S(T, R)$, $U = T \setminus E$,$R_U = R \setminus Req(E)$The one to one redundancy dividing strategy, divides $S(T, R)$ into $S(\{t\}, \emptyset)$ and $S(V, R_V)$where, t is one-to-one redundant test case, $V = T \setminus \{t\}$, $R_v = R$. Applying anyone of the above strategies on$S(T, R)$ gives the following satisfiability relations: $S(T, R) = S(T_0, R_0) \rightarrow S(T_1, R_1) \rightarrow \cdots S(T_i, R_i) \ldots \rightarrow$.In every transition of the above sequence, one among the following occurs: 1)during the application of essential dividingstrategy, $Opt(S(T_i, R_i)) = Opt(S(T_{i+1}, R_{i+1})) \oplus \{E_i\}$2)When the one-to-one redundancy strategy is used, $Opt(S(T_{i+1}, R_{i+1})) \subseteq Opt(S(T_i, R_i))$. From these optimal representative sets is reconstructed in a guaranteed way.

T.Y.Chen, M.F. Lau [7] conducted simulation study overfour different heuristics called as, Greedy, Greedy Essentials, Greedy Redundancy Essentials, and Heuristic based on overlapping of requirements:

In Greedy (G) approach, the test cases are selected repeatedly which are satisfying the maximum amount of unsatisfied requirements. And, only one of the test cases is selected at a point in time. Also, this the selected one satisfying at least one of the unsatisfied requirement.

In heuristic (H) approach, requirements are grouped according to number of satisfiablity to test cases. Those which are satisfying lesser number of requirements are having more essentialness. The heuristic selects test cases with more essentialness first. And, subsequently lesser essential test cases which satisfies unsatisfied requirements are selected during successive steps.

In the heuristic Greedy Essentials method, the essentials strategy is applied in first step, and in the next step greedy strategy is used, wherein the test case is selected at every step so as to satisfy maximum number of unsatisfied requirements.

Heuristic GRE is comprising the following: greedy, 1-to-1 redundant and essential strategies. One-to-one redundancy in test cases are removed as early as possible. Then strategy called essentials is applied. After which sometest cases may evolve as one-to-one extra. So, one-to-one redundant strategy and essentials strategy are both applied alternately. The greedy is applied only if other two strategies are not used. And if the greedy strategy is not used at all, then it is ascertained that the representative set arrived is the optimum set.

The authors have done simulation study on these approaches, with the assumption of equal overhead with respect to every test case. The performance among these heuristics is measured using ratio of overlap which is defined to be$n\mu/m$, where n represents total requirements, μ represents average number of requirements satisfied, and ‘m’representing total test cases.Based on this value, appropriate heuristic is suggested.

Wan Youngbing, Xu Zhongwei, Yu Gang, Zhu YuJun [21], developed an algorithm to partition the test cases based on requirement coverage. Then another algorithm, which combines greedy algorithm and linear search does the test suite reduction significantly.

Sara Sprenkle, SreedeviSampath, Amie Soter [22] developed test suite reduction mechanism for web application with an eye on user sessions. Every user session is denoted as user request in the URL form and its associated name value. A test case takes the form of HTTP requests pertaining to user sessions.

Three techniques based on requirement to URL coverage is analyzed in terms of Random, Greedy, and HGS. Also, three variations of concept analysis in terms of reduced suite size, coverage of program, bug detection effectiveness, time and storage constraints with a base in random, Greedy, and HGS heuristics. And, they concluded that concept clustering achieved better coverage with less cost while reducing test suite size.

3.3 Fault based reduction

Gong Dandan, Wang Tiantian, Su Xiaohong, and Ma Peijun [8] devised test suite reduction with an eye on localizing the fault. They considered coverage vector and path vector. The coverage matrix is fine-tuned by removing the pass yielded test cases, whose relevancies are week in the fault localization requirements, and is called coverage matrix based reduction.This approach is complemented as detailed: When the coverage vectors are identical for the test cases, then the redundancy in terms of identical path vectors are found and deleted.

Execution path of every test case t, PATH (t) in program P to be sequence of statements $= \langle S_1, S_2, \ldots S_i, \ldots \rangle$.In the path found, a statement S_i occuring several times if found inside looping structure.

Coverage vector is given as, $(t) = S' = \langle S'_1, S'_2, \ldots S'_n \rangle$, where 'n' is the number of statements of program P. This vector contains binary values based on statement coverage as:

$$S'_j = \begin{cases} 1, PATH(t)\ covered\ j-th\ statement \\ 0, PATH(t)\ uncovered\ j-th\ statement \end{cases} (1 \le j \le n)$$

Test suitehaving$\{t_1, t_2, \ldots t_m\}$, 'm' be amount of test cases for program P. The coverage matrix of statements is

$Cover(T) = \{Cover(t_1), Cover(t_2), \ldots Cover(t_m)\}$

The weekly relevant statements need not be considered for fault localization.A weakly relevant statement $S_k$of Program P for the test suite T, if and only if for all pairs of i and j ($1 \le i \le m$; $1 \le j \le m$; $i \ne j$), Cover ($t_i - s_k$) = = Cover ($t_j - s_k$)

Suspiciousness score of every statement is found to be difference between numberstest cases which succeeds and fails. The suspiciousness score for every weakly relevant statement is found to be relatively small. Weakly relevant statements and their corresponding test cases are not considered in coverage matrix, and is called as remaining coverage matrix RCOV(T).

The Fault Localization Requirements Vector $FL_{req}$is calculated as:The failed test suite for program P be $T = \{t_1, t_2, \ldots t_m\}$. To localize a single fault, the statement which failed in test execution should be considered. And all the test cases which failed its execution should be involved. $FL_{req} = Cover(t_1) \cap Cover(t_2) \cap \ldots \cap Cover(t_m)$. To localize multiple faults, $FL_{req} = Cover(t_1) \cup Cover(t_2) \cup \ldots \cup Cover(t_m)$, as one buggy statement need to be executed by one or more test cases failing it, and one test case which failed may not be executing all buggy statements.The vector remaining in the coverage vector with respect toFL_{Req} is Rcov(FL_{Req}).

For the fault localization in reqirements, coverage matrix is formed by joining the coverage matrix of successful inputs with vector on fault localization requirementsFL_{Req}.Remaining coverage matrix Rcov(T) is formed after ignoring weekly relevant statements. The test cases passed and also weakly relevant to fault localization requirement are removed.For test cases t_1 and t_2, the inference that t_1 is weakly relevant to t_2 if

$Rcov(t_1) \cap Rcov(t_2) = \langle z_1, z_2, \ldots z_n \rangle$

$where\ z_1 = z_2 = \cdots = z_n = 0$

Path vector based reductionis stated as:

For every pair of $Path(t_1) = \langle x_0, x_1, \ldots x_i, \ldots x_{|Path(t_1)|} \rangle$ and $Path(t_2) = \langle y_0, y_1, \ldots y_i, \ldots y_{|Path(t_2)|} \rangle$.Where, $|Path(t)|$ is the amount of statements in$Path(t)$. $Path(t_1)$and$Path(t_2)$ are identical paths denoted as $Path(t_1)$== $Path(t_2)$, if

1. $|Path(t_1)| == |Path(t_2)|$
2. $x_0 = y_0, x_1 = y_1, \ldots, x_i = y_i, \ldots, x_{|PATH(t_1)|} = y_{|PATH(t_2)|}$

The repeat sequence of statement and number of times it is being repeated.In next step this repeat sequence is removed from $Path(t)$

For each pair of $Path(t_1) = \langle x_0, x_1, \ldots x_i, \ldots \rangle$ and $Path(t_2) = \langle y_0, y_1, \ldots y_i, \ldots \rangle$, the execution path vectors towards loop standardization are $Path'(t_1)$ and $Path'(t_2)$ respectively. $Path(t_1)$ And $Path(t_2)$ are similar paths and denoted as$Path(t_1) \approx Path(t_2)$, if $Path'(t_1)$ and $Path'(t_2)$ are identical paths.To achieve loop standardization, passed test cases having similar paths are removed.The said approach is validated with experiments and proved to be effective.

Gregg Rothermel, Christie hong, Jeffery von ronne and Mary Jean Harrold [12] have conducted experiments and showed that test suite minimization algorithms severely compromises the bug detection capabilities of test suites. Savings in terms of the number and percentage of test cases removed is measured as follows:

$Number\ of\ test\ cases\ removed = |T| - |T_{min}|$

$Percentage\ of\ test\ cases\ removed = [(|T| - |T_{min}|) / |T|]$

Two kinds of costs are considered. First kind of cost is tool execution cost to achieve reduction. Second cost is based on the discarded fault revealing test cases, as this reduced effectiveness compounded over subsequent releases. Two methods are used to measure the cost of missed faults.

First method identifies test cases that reveal a fault, which is not present in reduced suite.

Second method classifies the results of test suite reduction, based on fault in one of the following ways 1) find ineffective test case in suite, which are not fault revealing 2) test cases in a suite which are fault revealing, and need not be eliminated 3) some of the test case elimination may compromise bug detection.

The following are found:

$number\ of\ bugs = F - F_{min}$

$percentage\ reduction\ in\ bug\ detection\ effectiveness = [(F - F_{min})/F] \times 100$

By conducting experiments, they inferred the following: 1) Minimization algorithm differences will not affect result differences. 2) Program size as well as structure may lead to differences in bug detection effectiveness. 3) Size of the test suite will impact reduction and bug detection effectiveness. 4) Powerful test cases when included in reduced test suite will lead to little loss in fault detection effectiveness. 5) Types of bugs will make difference in bug detection effectiveness. 6) Interaction of factors - program characteristics, test suite design and fault types facilitate determining faults.

In a coverage based reduction, the ratio of inputs to locate the bug and other inputs that locate the same fault is used in minimization. They conclude that test suite reduction may or may not impact fault detection effectiveness.

3.4 Cluster based reduction

SreedeviSampath, Renee C. Bryce [9] developed a heuristic, for test suite reduction and prioritization. They used clustering called concept analysis. This was used to cluster user session based test cases in the first stage. In next stage, heuristics applied in selection process from the clusters forming reduction in test suite. For every requirement, test cases are clustered and the suitable ones are selected with the help of heuristics.The heuristic used is to test-all-exec-requests, and it selects test cases form different concept analysis clusters so as to cover all test requirements, while maintaining different use cases. This test suite reduction method is complimented with the prioritization technique which uses many criteria as detailed below:

Count based criteria to prioritize are:http requests, parameter values, http requests length in descending/ascending order, Values length of parameters in ascending/descending order.

Frequency based criteria to prioritize are:Frequency of http requests access sequence in descending order, Frequency in all access http request sequence in descending order.

Combinatorial based criteria:Parameter interaction is main focus here. In single way interaction, the test which comes across the most uncovered parameter value is chosen.In dual way interaction, the test which comes across the most uncovered dual way interactions is selected.Logged ordering and random permutation are also used for test case prioritization.Mod_APFD_C metric to measure effectiveness is found and is as follows:

Consider testsuite T having n number of test cases, with execution costs t_1, t_2, …$t_{n..}$ Let F be a set ofm faults. These bugs are uncovered during test with fault severities as f_1, f_2, …$f_{m.}$The position of test case being denoted as $TF_{i,}$ and T is ordered as T'. Technique G in generation time, t_{gen} uncovers bug i. The $APFD_C$ for T' is measured as

$$Mod\_APFD_C = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_j}^{n} t_j - \frac{1}{2} t_{TF_i}\right)\right)}{\left(\sum_{i=1}^{n} t_i + t_{gen}\right) \times \sum_{i=1}^{m} f_i}$$

AlirezaKhalilian and Saeed Parsa [15] developed an approach to minimize test suites based on Cluster analysis. They analyzed execution profiles to form clusters. The procedure is based on two different two different coverage criteria. Test cases are executed over the instrumented program to collect execution profiles. Clustering of profiles is done with clustering algorithm with the help of wekatool.The test suite reduction algorithm takes testing requirements and clustered test cases as inputs. The algorithm finds test cases as effective, when they satisfy most requirements and exposes most of the faults. While satisfying requirements, execution paths are formed. Overlapping execution paths lead to redundancy in test cases and hence need to be removed. At the same time Definition-Use pairs may prove that those are not redundant, as they uncover errors in those execution paths. Hence, both criterias used in the algorithm and found to be effective in test suite reduction.

Subashini, Jeyamala [17] have used k-means clustering technique with the help of Wekatool.The technique uses Control Flow Graph of any program. Then the independent paths are found. These paths are clustered with an objective of minimizing squared error function. Ultimately reducing test cases count of a suite.

KartheekMuthayala, Rajshekhar Naidu [18] developed an algorithm to achieve reduction in test suite using data mining. As a first step, they applied k-means clustering algorithm to cluster test cases of similar behavior. Then pick-up cluster algorithm is used to select a representative test case from each cluster. For all these implementation, they relied on Weka tool. If clustering does not cover a particular behavior(for example, path coverage) for the entire system, then take some higher values of k and reframe the clusters.

Saran Prasad, Mona Jain, Shrada Singh, and Patvardhan [19] proposed a technique based on coverage criteria with respect to function, statement, and branch. As a first step, hierarchical clustering is applied. Clusters are formed with test cases whose functional coverages are same. A binary matrix is used to represent the functional coverage of test cases. PureCov parser and GCov parser tools are used to formulate this matrix. Function call sequences among the clusters of similar test cases are compared in the next step. To maintain these records, a stack based architecture, Call Stack is used. So, a set of test cases having same functional flow is grouped further. Test cases are further compared on the basis of same statement coverage and grouped. And from this obtained group, set of test cases covering same branch or path of the function is obtained. The approach finally determines test cases whose branch coverage is same within a function are redundant and hence reduction is applied.

SriramanTallam, Neelam Gupta [20] devised greedy algorithm based on concept analysis to minimize the test cases. Concept analysis figures out maximal grouping. Here, the concepts are formed by grouping objects and attributes. It is a hierarchical clustering technique as it relies on concept table, concept lattice and table of concepts. Object Reduction Rule: object O1 implies O2, if concept having O1 is in bottom of lattice than the concept having O2 and both concepts are in sequence. Then the row corresponding to the object O2 may be dismissed from context table. Attribute Reduction Rule: attribute a1 implies a2, if concept having a1 lies in bottom of lattice than the concept having a2 and both concepts are in sequence. Then the column corresponding to the attribute a2 may be dismissed from context table. Owner Reduction Rule: strongest concepts are the ones which are in higher layer to a concept in the lattice. Strongest concept having an attribute implies that a test case in the concept must be chosen to deal with that attribute. Inference exists in a lattice, when two concepts c_i and $c_j$such that c $\leq_R$ c_i along with c$\leq_R$$c_j$ whichare neighbors of c. when a lattice has inference, it has no object implication, attribute implication and no strongest concept. Then the delayed greedy approach is applied, which is as follows: test case covering maximum requirements is found and its corresponding row removed from context table. Also, the requirements mapped by these inputs are dismissed from the table.

3.5 Multi objective based reduction

Shuai Wang, Shaukat Ali, Arnaud Gotlieb [10] performed test suite minimization in product line engineering. The effectiveness measures used by them are:Feature pairwise coverage, Test minimization percentage, Fault detection capability, Average execution frequency and Overall execution time. And they used the following ten search algorithms :Weight Based GA's (WBGA) setsfixed weight for every objective defined.WBGA in Multi Objective Optimization (WBGA-MO) employs a pool of weights. Weights are assigned randomly per objective during every generation. RWGA Randomly assigns normalized weights to multiple objective functions for each solution when selecting fittest individuals at each generation. Non dominatedSorting Based GA (NSGA-II) is based on Pareto Dominance Theory, which outputs a set of non-dominated solutions for multiple objectives. Cellular based GA (MOCell) is on the assumption that an individual only interacts with its neighbors during the search process.Improved Strength Pareto Evolutionary Algorithm (SPEA2), fitness value for each solution is calculated by summing up a strength raw fitness based on the defined objective functions and density estimation.Pareto Archived Evolution Strategy(PAES) by applying dynamic cross over and mutation operators aims at maximizing the stability for the selected solutions. Speed Multi Objective Particle Swarm Optimization constrained on speed selects best solutions by calculating crowding distance. Cellular Genetic Algorithm with Differential Evolution (CellDE), uses MOCell as a search engine and DE calculates the weighted difference between two randomly selected solutions and integrate the obtained parts into third solution for generating a new solution. Random Search (RS), stochastic algorithm randomly generate solutions during each generation.Also, a tool called Test Minimization with Search Algorithms (TEMSA) is developed for Test Suite Minimization.

Shin Yoo, Mark Harman [11] used three objectives namely, Coverage, fault history and Cost with a base on Pareto optimality. They devised an additional greedy algorithm for two objectives in test suite minimization.Also, they devised hybrid NSGA-II algorithm, which comprises additional greedy algorithm with NSGA-II based on pareto fronts.The Pareto frontier is stated as:

PF1:There is no alternative subset in attaining better coverage than C while not consuming much time than T

PF2: There is alternative subset to complete in smaller time than T whilst coverage is more or equal to C

Greedy algorithms are effective for single objective optimization problems. A variant, additional greedy algorithm for multiple objective is formed in order to measure coverage per unit time. This objective is achieved by cost cognizant greedy algorithm. The additional greedy algorithm lead selection cannot be dominated and at the same time Pareto efficient solutions cannot be made.

The greedy approach may be extended to consider 'n' test cases inachievingpareto optimality.But this is equivalent to exhaustive search, and is infeasible.

NSGA-II has two significances.First, selection is based on paretooptimality. With non dominated sorting individual solutions are classified into different dominance levels. Second difference lies in the crowding distance.When at equal dominance level, individual having greater crowding distance is rewarded.

The additional greedy algorithm results were taken as initial population to NSGA-II algorithm forming HNSGA-II algorithm. The elitism is achieved by gaining diversity among initial population yielded by additional greedy algorithm. To compare different algorithms, a reference pareto frontier is formed by combining best in every approach.

Alessandro Marchetto, Mahfuzul Islam, Angelo Susi, Giuseppe Scanniello [16] proposed multi objective test cases reduction. They have done three dimension analysis. Analysis phase test cases is the focus in structural perspective. User/system requirements is focused in functional perspective. The cost dimension focuses on the time to execute test cases. Traceability links among source code, requirements and test cases are needed. For this Latent Semantic Indexing as the Information Retrieval technique is used to recover traceability links.

$$CCov(t) = \sum_{s \in statements} \begin{cases} 1 & s \in Code\ covered \\ 0 & otherwise \end{cases}$$

$$cumCCov(t_i) = \sum_{j=0}^{i-1} CCov(t_j)$$

Coverage of test cases according to predefined weights is

$$WRCov(t) = \sum_{r \in Reqs} \begin{cases} W_r & r \in ReqsCovered \\ 0 & otherwise \end{cases}$$

$$w_r = \begin{cases} 1 & r \in TesterRelevant_r \\ 0.5 & TesterPartialRelevant_r \\ 0 & TesterNonRelevant_r \end{cases}$$

The strength is

$$w_{rD}(r_l, r_m) = \frac{w_{req}(r_l, r_m) + w_{code}(r_l, r_m)}{2}$$

$$w_{req}(r_l, r_m) = IRSimilarity(r_l, r_m)$$

$$w_{code}(r_l, r_m) = \frac{overlapClasses(r_l, r_m)}{totalClasses(r_l, r_m)}$$

The final requirement coverage of t is

$$WRCovD(t) = \sum_{r \in Reqs} w_r * \left(\sum_{r_l \neq r \in Reqs} w_{reqs}(r, r_l) \right)$$

$$cumRCov(t_i) = \sum_{j=0}^{i-1} WRCovD(t_j)$$

The overall cost of a suite S is the sum of execution cost of all test cases.

$$InverseCost(t_i, S) = Cost(S) - \sum_{j=1}^{i} Cost(t_j)$$

The Non dominated Sorting Genetic Algorithm II is applied to maximize the three considered dimensions. The pareto front brings the optimal tradeoff among the structural, functional and cost dimensions.

3.6 Linear programming model based reduction

Dan Hao, Lu Zhang, Hong Mei, Xingxia Wu, and Gregg Rothermel [13] uses coverage information when reducing test suite. For the fault detection capability, confidence level and upper limit on loss acceptabilityis the threshold. In this methodology, as a first step data on losses in bug detection capability for statements is collected. In the second step, two integer linear programming models constructed in reducing test suites.

Assume test suite T is reduced to T'.For statement containing mutation faults, which are executed by i_1 test cases in T, and by i_2 test cases in T'

$$CovDiff(i_1, i_2) = \frac{\#Faults(T_{i1}) - \#Faults(T'_{i2})}{\#Faults(T_{i1})}$$

On-demand test suite reduction is made with the help of integer linear programming models. Both models have the common objective function, predicate variables with distinct constraints.

Objective function is $min \sum_{i=1}^{n} x_i$, Where x_i representing whether test case $t_i$has been selected to be part of reduced suite.

Decision variables:For test suite$T = \{t_1, t_2, \dots t_n\}$, having n variables $x_i (1 \leq i \leq n)$ to decide whether test case t_i is included in the reduced test suite T'.

$$x_i = \begin{cases} 1, & if\ test\ case\ i\ is\ slected \\ 0, & otherwise \end{cases}$$

For a program P, having m statements$(S = \{S_1, S_2, \dots S_m\})$, Boolean predicate variables defined below to denote changes in coverage during reduction of T to T'.

$$w_{i,q} = \begin{cases} 1, iff\ q\ test\ cases\ in\ T'cover\ S_j \\ 0, otherwise \end{cases}$$

Constraints:

And, this $w_{i,q}$ will be true for only one value denoting amount of test cases in T' covers S_j, then $\sum_{q=1}^{pj} w_{j,q} = 1$ for any $1 \le j \le m$,

This can also be represented as $\sum_{i=1}^{n} x_i \times C(i,j)$

Where, the coverage information is,

$$C(i,j) = \begin{cases} 1, if\ test\ case\ t_i covers\ statement\ s_j \\ o, otherwise \end{cases}$$

The local constraints over loss in bug detection capability towards each statement is given as $\sum_{q=1}^{p_j} w_{j,q} \times V_c(p_j, q) \le l\%$. Where $V_c(p_j, q)$ is the loss in bug detection capability of a statement at confidence level c% and coverage varying from p_j to q

The global constraint over loss in bug detection capability for the program is given as $\sum_{j=1}^{m} \sum_{q=1}^{p_j} w_{j,q} \times V_c(p_j, q \le m \times l\%)$

Both the above two Integer Linear Programming models proved empirically to be effective.

3.7 Interaction based reduction

Dale Blue, Rachel Tzoref-Brill, ItaiSegall, and AviadZlotnick [14] presented an interaction based test suite minimization which is complemented by combinatorial test design. The rationale behind this approach is that most software faults are caused by interaction between small number of parameters.An algorithm is developed which covers target, which covers the same target covered by original suite. The features such as avoiding unnecessary calculations, test prioritization and counting uncovered targets were added in the algorithm.

4.Conclusion

An in-depth survey of various test suite reduction mechanisms were explored. We have scrutinized various techniques devised by many different researchers in various different dimensions. This survey paper gives insight into test suite reduction techniques and motivates the researchers to bring innovation in improving software testing efficiency. We have explored techniques in clustering, data mining, evolutionary algorithms, and heuristics.

References

[1] James A. jones, Mary Jean Harrold, “Test suite Reduction and Prioritization for Modified condition/decision Coverage”, IEEE Transactions on Software Engineering, Vol.29, No.3, March 2003

[2] Chu-Ti Lin, Kai-Wei Tang, Gregory M.Kapfhamme, “Test Suite Reduction methods that decrease regression testing costs by identifying irreplaceable tests”, Information and Software Technology(54)10, October 2014

[3] HaoZhong, Lu Zhang, Hong Mei, “An experimental study of four typical test suite reduction techniques”, Information and Software Technology50 (2008) 534–546

[4] Jun-Wei Lin, Chin-Yu Huang, “Analysis of test suite reduction with enhanced tie-breaking techniques”, Information and Software Technology 51 (2009) 679–690

[5] T.Y. Chen, M.F. Lau, “A new heuristic for test suite reduction”, Information and Software Technology 40(1998) 347-354

[6] TsongYueh Chen, Man Fai Lau, “Dividing strategies for the optimization of a test suite”, Information Processing Letters 60 (1996) 135- 141

[7] T.Y. Chen, M.F. Lau, “A simulation study on some heuristics for test suite reduction”, Information and Software Technology40 (1998) 777–787

[8] Gong Dandan, Wang Tiantian, Su Xiaohong, Ma Peijun, ”A test-suite reduction approach to improving fault-localization effectiveness”, Computer Languages, Systems & Structures 39 (2013) 95–108

[9] SreedeviSampath, Renée C. Bryce, “Improving the effectiveness of test suite reduction for user-session-based testing of web applications”, Information and Software Technology 54 (2012) 724–738

[10] Shuai Wang, Shaukat Ali, Arnaud Gotlieb, “Cost-Effective Test suite Minimization in Product lines Using Search Techniques”, The Journal of Systems Software (2014), Volume 103, May 2015, pages 370-391

[11] Shin Yoo, Mark Harman, “Using hybrid algorithm for Pareto efficient multi-objective test suite minimization”, The journal of Systems and Software 83(2010), 689-701

[12] Gregg Rothermel, Mary Jean Harrold, Jeffrey Von Ronne, Christie Hong, “Empirical Studies of Test Suite Reduction”, Software Testing Verification and Reliabilty 12 (2002) 219-249

[13] Dan Hao, Lu Zhang, Xingxia Wu, Hong Mei, Gregg Rothermel, “On- Demand Test Suite Reduction”, 34th International Conference on Software Engineering (ICSE 2012), June 2-9, 2012, pages 738-748, Zurich, Switzerland

[14] Dale Blue, ItaiSegall, Rachel Tzoref-Brill, AviadZlotnick, “Interaction-Based Test-Suite Minimization”, 35thInternational Conference on Software Engineering, (ICSE 2013), May 18-26, 2013, 182-191, San Fransisco, USA

[15] AlirezaKhalilian and Saeed Parsa, “Bi-criteria Test Suite Reduction by Cluster Analysis of Execution Profiles”, Advances in Software engineering Techniques, in : 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Krakow, Poland, October, 12-14, 2009, Volume 7054 2012

[16] Alessandro Marchetto, Md. Mahfuzul Islam, Angelo Susi, Giuseppe Scanniello, “A Multi-Objective Technique for Test Suite Reduction”, The Eighth International Conference on Software Engineering Advances, October 27, 2013, 18-24

[17] Subashini.B, Jeyamala.D, “Reduction of Test Cases using Clustering Technique”, International Journal of Innovative Research in Science, Engineering and Technology, Volume 3, Special Issue 3, March 2014

[18] KartheekMuthayala, Rajshekharnaidu, "A Novel approach to test suite reduction using Data Mining, Indian Journal of Computer Science and Engineering", Vol 2 No. 3 – June –July 2011

[19] Saran Prasad, Mona Jain, Shradha Singh, "Regression Optimizer – A multi coverage criteria Test Suite Minimization Technique", International Journal of Applied Information Systems, Volume 1 – no. 8, April 2012

[20] SriramanTallam, NeelamGupta, "A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization", Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, Volume 31 Issue 1, January 2006, pages 35-42

[21] Wan Yongbing, Xu Zhongwei, Yu Gang, Zhu YuJun, "A Test Suite Reduction Method based on Test Requirement Partition, International Journal of Grid and Distributed Computing", Vol. 6, No. 4, August 2013

[22] Sara Sprenkle, SreedeviSampath, Emili Gibson, Lori Pollock, Amie Souter, "An Empirical Comparison of Test Suite Reduction Techniques forUser-session-based Testing of Web Applications", 21stInternational Conference on Software Maintenance, 26-29, Sep 2005, 587-596