RESEARCH ARTICLE OPEN ACCESS

Test Suite Reduction Using Greedy Approach based on Code coverage

B. Bhaskar Kumar Rao[1], R.N. Mounika[2], T. Tejaswini[3]
1Assistant Professor, dept. of IT, Sreevidyanikethan Engineering College, Tirupati
Email: bhaskarkumarraob@gmail.com
2,3Dept of IT, Sreevidyanikethan Engineering College, Tirupati
Email: binnymounika@gmail.com, tejaswini12a2@gmail.com

Abstract:

Programming testing is an action to discover most extreme number of blunders which have not been found yet with ideal time and exertion. As the product advances the span of the test suite and develops with new experiments being added to the test suite. Be that as it may, because of time and asset requirements rerunning all the experiments in the test suite isn't conceivable, each time the product is adjusted, keeping in mind the end goal to manage these issues, the test suite size ought to be sensible. In this paper a novel approach is exhibited to choose a subset of experiments that activity the given arrangement of necessities with for information stream testing. With a specific end goal to, express the viability of the proposed calculation, both the current Harrold Gupta and Soffa (HGS) and Bi-Objective Greedy (BOG) calculations were connected to the produced test suites. The outcomes acquired from the proposed calculation were contrasted and the condition of-workmanship calculations. The aftereffects of the execution assessment, when contrasted with the current methodologies demonstrate that, the proposed calculation chooses close ideal experiments that full fill greatest number of testing necessities without bargaining on the scope angle.

*Keywords:**Test suite, experiments, coverage, testing, goal product.***

I. INTRODUCTION

Software testing is a vital activity in the development of software to find bugs as early as possible. The objective of software testing is to detect faults in the program and therefore, provide more assurance for customers on the quality of the software. Any software that is developed and put into use may be subjected to addition or modification of existing features. With a tremendous number of possible test cases available as software evolves, testers have no means to control the size of the test suite. The literature survey [10, 12] throws light upon the fact that software testing consumes a greater chunk of the development cost. With software projects also being subjected to time and resource constraints, ways to address test suite reduction has become a topic of interest among researchers. During test case generation or after creating the test suite, the effectiveness of the test process can be improved if a minimal subset of test cases could be determined to exercise all the test requirements as the original test suite. Apparently, the lesser the number of test cases, the lesser time it takes to test the program which consequently reduces the computational effort of running the entire test suite.

Nonetheless, another imperative issue to be tended to amid test suite diminishment is the scope angle. Likewise, scope based lessening procedures ought to guarantee that greater part of the execution ways of the given program are worked out. The general ramifications from the past research work [15, 18] is that test case(s) that don't add to the scope of a test suite will probably be

incapable in fulfilling the predefined necessities. From the writing overview it can be derived that test suite lessening approaches essentially diminish the extent of the test suite [2, 3, 16]. In any case, how far the lessened test suite got full fills the test metric(s) under thought is an imperative issue to be tended to. Indeed, some potential downsides saw in test suite lessening review includes arbitrary choice of experiment in case of a tie (at least two experiments fulfilling a similar arrangement of necessities), complex scientific operation for test suite decrease, nature of test case(s) [7, 14] and so on., Thus, the exchange off amongst scope and ideal experiment choice is key in test suite diminishment.

In this paper another calculation for Test Suite Reduction called Coverage Based Test Suite Reduction (CBTSR) has been proposed. The commitments of this paper incorporate the accompanying:

- Identifying an ideal delegate test set including experiments which are identified with the given testing objective.
- Applying information stream testing to produce test cases and prerequisites to analyse the physical structure of the program and find sub-ways navigated by factors.
- Using the proposed CBTSR calculation for test suite decrease.
- Performing a set of empirical studies on ten subject programs. Then comparing the relative performance and effectiveness of the proposed reduction algorithm with the state-of-art Harrold Gupta and Soffa (HGS) [7] and Bi-Objective Greedy (BOG) [14] algorithms.

II. Test Suite Reduction Problem

As indicated by the meaning of test suite issue given [7, 10]:

- A test suite T of experiments {t1, t2, t3...tn}, known as all-inclusive test suite.
- An arrangement of testing prerequisites {r1, r2,...,rm} that must be secured to give the coveted scope to the program under thought.
- Subsets {T1, T2,...Tm} of T known as test sets where each test set is related with ri, to such an extent that any one test case(s) having a place with Ti full fills ri.

  The goal of test suite minimization issue is to locate the delegate set (diminished test suite) Trs that activities a similar arrangement of those exercised by the original test suite T.

2.1. Background:

The issue of finding the delegate set is compared to the set-cover issue [10]. The set-cover issue has been appeared to be NP finished [7] in HGS calculation. By the by, there has been some examination work [7, 14] in the territory of processing ideally limited test suites. The vast majority of the other research works in test suite minimization have however depended on heuristics for registering close ideal arrangements [2, 3, 11, 16, 17]. A few methodologies have been proposed in writing [1, 2, 4, 7, 8, 9, 11, 14, 16, 17] for tending to test suite diminishment issues. Practically speaking test suite lessening approaches by and large concentrate on expelling old and excess experiments from the general test suite [14]. The goal of test suite minimization in programming testing is to hold the best experiments just [14, 15]. Further, these experiments ought to be equipped for fulfilling the most number of test prerequisites and thus likewise uncover the deficiencies in presence. Alongside the test suite lessening systems, use of scope angles is additionally essential. Scope criteria [7, 8, 9, 11], for

example, branch scope, articulation scope, information stream scope, MC/DC and call stack scope to give some examples, practice more noteworthy affirmation to the quality, dependability and ceasing rules [4, 5, 6, 12, 18] for test engineers.

HGS calculation proposed by Harrold et al. [7] has drawn a ton of consideration towards test suite decrease. This calculation utilizes the idea of cardinality (number of event of an experiment in each test set) to lessen the test suite estimate. It starts the test suite minimization process by choosing singleton test cases (test cases with cardinality one) and continues to the following higher cardinality test cases. Likewise, the as of late proposed BOG calculation by Saeed and Alireza [14] utilizes complex grid operations to decrease the test suite estimate. Be that as it may, a potential downside of the HGS calculation is the irregular choice of experiments amid test suite lessening in case of a tie. Further, in BOG calculation the requests in which test sets are subjected to diminishment antagonistically influence the estimations of the agent set. Thus, from the writing study it is very evident that there is a requirement for investigate work to concentrate on issues emerging while at the same time streamlining the test suite estimate. In the following area the CBTSR calculation is portrayed with a case.

III. Test Suite Reduction Algorithm

3.1. Related Concepts

The quantity of prerequisites R might be limited or boundless. Be that as it may, from an even minded perspective, it is expected that R is limited. For every prerequisite, ri⊂ R, there is an experiment tj in the info area that fulfills it. Subsequently, a limited test suite T additionally exists. The factors m and n are utilized to signify the extent of R and T, separately. The Boolean framework an of size m × n is utilized to depict the fulfilment connection amongst prerequisites and test sets with the end goal that ∀ri∈R and ∀tj∈T Equation 1:

$$A_{ij} = \begin{cases} 1 \text{ if } r_k \text{ is covered by } t_j \\ 0 \quad \text{otherwise} \end{cases}$$

where, for i=1,2,…,m and j=1,2,…,n.
The sum vector S represents the count of all "1"in the ith row of aij. The representation of vector S is denoted in Equation 2:

$$S = \begin{bmatrix} \Sigma a_{1n} \\ \Sigma a2n \\ \vdots \\ \Sigma a_{mn} \end{bmatrix}$$

Thus, the simplification can be further done on sum of vector values as given in Equation 3:
S={ci},i® 1 to m
Where, ci=Σnj=1 aij
In the process of test suite reduction, the mapping f: T→R can be defined as a Boolean function. The coverage relationship expressed as a requirement matrix can be considered as the satisfaction relationship among test requirements and test cases inthe optimal representative set selection problem. Thus, the Boolean function simplification problem can be equated to the optimal representative set selection problem.

3.2. CBTSR Algorithm

The calculation CBTSR demonstrates the age of the diminished test suite through basic

numerical operations. In the calculation the accompanying suppositions were made: Let n signify the quantity of experiments in a test set and m mean the quantity of test necessities. The other related issues are: Each test set Ti comprises of experiments comparing to a necessity. The diminishment procedure in the proposed CBTSR approach starts with the development of experiment necessity grid 'A'. This lattice maps the experiments with the testing prerequisites. A relationship between an experiment and necessity is demonstrated by one or zero generally. In the network each ith push signifies the prerequisite scope and each jth segment means the experiment cover with the requirement(s). The calculation initially incorporates all the experiments tjs that happen as a solitary component in the test set Tis (singleton experiment), to the brief set Ts. At that point:

Algorithm 1: CBTSR

Input:*Test cases in the given test sets along with requirements. Test Sets: T1,T2, ...,Tm.□Associated requirements: r1,r2,..., rm. Test cases: t1,t2,...,tn.*

Output:*Reduced test suite. Trs a representative set of T1, T2,..., Tm.*

BeginA: is a Boolean matrix, 1-covered and 0 – uncovered. sel_tc:= {}: selected test cases returned by the subroutine. Select optimal (). list_t: list of test caseslist_r: list of requirementsTs:={}: selected singleton test casesTtemp:={}: temporarily selected test casesalgorithm CBTSR

{
Begin
list_t : =all tj∈ T //Contains all the test cases. list: =all ri∈ R //Contains all the requirements construct
the matrix A; //matrix denoting relationship between requirements and test cases for each ri do. Construct the vector S //Vector consisting of sum of the elements from row 1.m of matrix A
Ts: =∪Ti; //Assign test set(s) with row sum= 1 to the temporary set Ts.
update list_t:= remove all tj selected;
// Update by removing all the marked test cases update list_r:= remove all ri selected;
//Update by removing all the marked requirements
endfor//Consider unmarked requirements where i→1 to m and select the□Corresponding test set
for each Ti such that there exists ri do
sel_tc=select optimal(list_r, list_tc);□//Invoke the subroutine by passing the list of test cases and requirements
updatelist_t:= remove all tj selected
// Update by removing all the marked test cases
updatelist_r:= remove all ri selected;

//Update by removing all the marked requirements Ttemp :=∪ { sel_tc}; // distinct test cases withhighest coverage value
endfor□Trs = Ttemp∪Ts // union of all distinct optimal test cases end
}end CBTSR
Subroutine 1:*select optimal (list_r, list_t)*
*/*selects test sets to be included in the representative set */□Input: unmarked test cases and requirements□Output: Representative set□S: integer vector denoting number of requirements covered by a test sets*
T: test set with highest coverage
value□max(): *returns S vector row(s) having highest sum value*
Begin {
reconstruct the vector S; // sum of requirements covered by test cases if max(S) > 0

return (T=max(S)); // return the row with the highest
value to the variable sel_tc else return; // return to the main program }
end
endselect_optimal

The comparing events of the prerequisites ris and, test cases tjs in the experiment prerequisite grid are then reset to the esteem zero as spoke to in Equations 4 and 5. Condition 4 resets every one of the components in section 'j' to the esteem zero and Equation 5 resets every one of the components in push I to the esteem zero. This is trailed by expelling the experiment and prerequisite from the rundowns: list_t and list_r. At that point, the subroutine select_optimal() is recursively called to choose the rest of the experiments:

A[All, ti]=0 (4)

A[ri]=0 (5)

Another subroutine max() when conjured restores the row(s) with the most extreme test case(s) covering the given prerequisite, from the vector S. The tjs set apart in row(s) returned are added to another transitory set T . Again the comparing necessity r s and t s temp ij are reset to the esteem zero and the points of

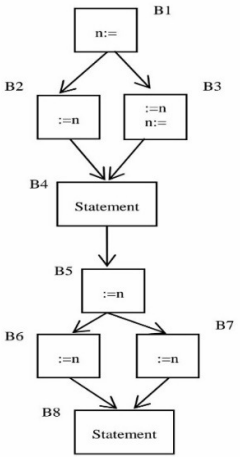


Figure 1: Control Flow Graph for odd even program

Table 1.

Beginning EXPERIMENT NECESSITY LATTICE FOR THE PROGRAM.

S.no	Testcase,t_j	Execution path	DU-pair(s) in the execution path
1	t_1	B_1,B_2,B_4,B_5,B_6,B_8	(B_1,B_2) $(B_1,B_5)(B_1,B_6)$
2	t_2	B_1,B_2,B_4,B_5,B_7,B_9	(B_1,B_2) (B_1,B_5) (B_1,B_7)
3	t_3	B_1,B_3,B_4,B_5,B_6,B_8	(B_1,B_3) (B_3,B_5) $B_3,B_6)$
4	t_4	B_1,B_3,B_4,B_5,B_7,B_8	(B_1,B_3) (B_3,B_5) $B_3,B_7)$

Table 2
Test sets produced for the DU sets.

S.no	Test Sets, T_i	DU-pair	Test cases in T_i
1	T_1	(B_1,B_2)	$\{t_1,t_2\}$
2	T_2	(B_1,B_5)	$\{t_1,t_2, t_3,t_4\}$
3	T_3	(B_1,B_6)	$\{t_1,t_3\}$
4	T_4	(B_1,B_7)	$\{t_2,t_4\}$
5	T_5	(B_1,B_3)	$\{t_3,t_4\}$
6	T_6	(B_3,B_5)	$\{t_3,t_4\}$
7	T_7	(B_3,B_6)	$\{t_3\}$
8	T_8	(B_3,B_7)	$\{t_4\}$

The lessening procedure for the proposed CBTSR, HGS and BOG calculations starts with the development of experiment prerequisite grid as in Figure 2. In the framework and, each line speaks to the prerequisite ri and every segment the experiment tj.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure2. Beginning experiment necessity lattice for the example program.in the delegate set Trs. Advance the experiments chose into the delegate set, acquired utilizing BOG calculation additionally did not choose the DU-combine (B1, B7). Hence, the actual objective to determine whether a positive number is odd or even could only be partially tested using the test cases in the representative set, of both HGS and BOG algorithms Table 3. However, retesting the sample program using the test cases in the representative set Trs of the proposed CBTSR algorithm provides the desired coverage of all DU-pairs Table 3 and also satisfies all the requirements to determine whether a number is odd or even.

TABLE 3

DELEGATE SET ACQUIRED FOR ODD/EVEN PROGRAM.

Algorithm(s)	|T|	|DU|	T_{rs}	DU-pair(s) Not covered by The representative Set
HGS	16	8	$\{t_1,t_3,t_4\}$	(B_1,B_7)
BOG			$\{t_1,t_3,t_4\}$	(B_1,B_7)
CBTSR			$\{t_1,t_2,t_3,t_4\}$	None

IV. EXPERIMENTS AND ANALYSIS

An empirical study was conducted to evaluate the proposed CBTSR algorithm and state-of-art algorithms, using ten program units each consisting of 11 to 24 lines of coding that cover a wide range of applications. The program description along with the lines of coding is shown in Table 4. The selection of test cases was done using Rapps and Weyuker data flow criterion [13]. Each program considered for experimentation used DU-pair(s) that were hand- instrumented. All the test suite reduction approaches considered in this work had been implemented using Java.

TABLE 4
DEPICTION OF PROGRAM UNITS.

S.no	Test Sets, T_i	DU-pair	Test cases in T_i
1	T_1	(B_1,B_2)	$\{t_1,t_2\}$
2	T_2	(B_1,B_5)	$\{t_1,t_2,t_3,t_4\}$
3	T_3	(B_1,B_6)	$\{t_1,t_3\}$
4	T_4	(B_1,B_7)	$\{t_2,t_4\}$
5	T_5	(B_1,B_3)	$\{t_3,t_4\}$
6	T_6	(B_3,B_5)	$\{t_3,t_4\}$
7	T_7	(B_3,B_6)	$\{t_3\}$
8	T_8	(B_3,B_7)	$\{t_4\}$

$$RCov = \frac{|R_{cov}|}{|R_{tot}|} \times 100$$

Where, | Rtot | means the aggregate number of test prerequisites under thought amid test determination and |Rcov| is the quantity of necessities fulfilled by the test cases in the agent set. Higher RCov implies better necessity scope by the agent set Trs.

- The level of test Suite Size Reduction (SSR) [14, 15] is characterized as in Equation 7:

$$SSR = \frac{|T| - |T_{rs}|}{|T|} \times 100$$

Where, |T| means the quantity of experiments in the first test suite and |Trs| the quantity of experiments in the delegate set. Ideal SSR with better RCov is attractive.

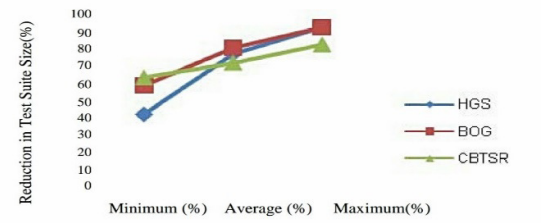
RCov: The following metric assessed was the necessity scope by the agent sets. The perceptions made amid experimentation demonstrated that the prerequisite scope was reliably high. In spite of the fact that the normal test SSR was high when utilizing HGS and BOG calculations, the normal RCov was insignificantly less when contrasted with the proposed CBTSR approach. Accordingly, the normal estimations of the test measurements considered amid execution assessment for the proposed CBTSR calculation was superior to anything the condition of-craftsmanship calculations Figure 4.

From the trials directed, the perceptions made are outlined as takes after:

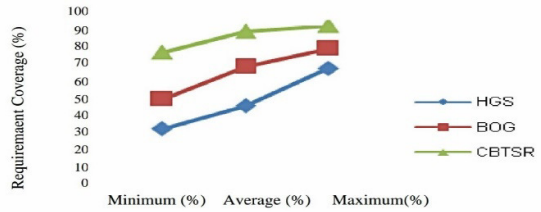
- HGS calculation concentrated on the cardinality of test sets to build the agent set. In the HGS calculation the recursive capacity for test suite minimization was summoned atleast once, to break the ties among similarly imperative experiments. In this calculation such recursions backed off the minimization procedure. Promote on account of a tie between test cases, irregular experiment choice likewise changed the scope of prerequisites.

- The proposed CBTSR test suite diminishment calculation expelled excess experiments presented amidprogram improvement and held just the best experiments that added to the necessity scope. The experiments that were held could likewise give the greatest necessity scope. From Figure 4 it is very clear that when CBTSR calculation was utilized, the experiments in the agent set Trs fulfilled more number of prerequisites and

accordingly along these lines offered better scope by navigating more DU-ways in a given program.



a) Test suite reduction.



b) Requirement coverage.

V. Conclusion

The commitments of this work concentrate on enhancing the viability of programming testing downstream as unit testing. In spite of the fact that, there has been some current work around there, in the present work endeavours have been made to decrease the extent of the test suite utilizing a basic approach that spotlights on the test measurements: Size and necessity scope. The proposed CBTSR calculation produced a decreased test suite iteratively utilizing straightforward grid operations. The execution assessments of the proposed CBTSR approach demonstrate that:

1. CBTSR calculation offered reliably preferable RCov over the condition of-workmanship calculations HGS and BOG.
2. Be that as it may, the normal test SSR of the proposed CBTSR (74.77%) was insignificantly not as much as the condition of-craftsmanship calculations HGS (79.5%) and BOG (82.9%).

Along these lines, from the perceptions made in this work it can be surmised that CBTSR decreases the measure of the test suite by holding experiments that offer most extreme RCov. In future this work might be stretched out for another test metric to decide the blame discovery capacity.

REFERENCES

[1] Agrawal H., "Efficient Coverage Testing using Global Dominator Graphs," in Proceedings of ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Toulouse, France, pp. 11-20, 1999.

[2] Black J., Melachrinoudis E., and Kaeli D., "Bi-Criteria Models for All-Uses Test Suite Reduction," in Proceedings of the 26th International Conference on Software Engineering, Edinburgh, UK, pp. 106-115, 2004.

[3] Chen T. and Lau M., "Dividing Strategies for the Optimization of a Test Suite," Information Process Letters, vol. 60, no. 3, pp. 135-141, 1996.

[4] Errol L. and Brian M., "A Study of Test Coverage Adequacy in the Presence of Stubs," Journal of Object Technology, vol. 4, no. 5, pp. 117-137, 2005.

[5] Hutchins M., Foster H., Goradia T., and Ostrand T., "Experiments on the Effectiveness of Dataflow and Control-based Test Adequacy Criteria," in Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, pp. 191-200, 1994.

[6] Horgan J. and London S., "ATAC: A Data Flow Coverage Testing Tool for C" in Proceedings of Symposium on Assessment of Quality Software Development Tools, New Orleans, Louisiana, pp. 2-10, 1992.

[7] Harrold M., Gupta R., and Soffa M., "A Methodology for Controlling the Size of a Test Suite," ACM Transactions in Software Engineering and Methodology, vol. 2, no. 3, pp. 270-285, 1993.

[8] Jeffrey D. and Gupta N., "Test Suite Reduction with Selective Redundancy," in Proceedings of the 21st IEEE International Conference on Software Maintenance, Budapest, Hungary, pp. 549-558, 2005.

[9] Jones J. and Harrold M., "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage," IEEE Transactions on Software Engineering, vol. 29, no. 3, pp. 195-209, 2003. [10] Naresh C., Software Testing Principles and Practices, Oxford University Press, India, 2011.