COMPUTATIONAL SCIENCE AND TECHNIQUES
Volume 2, Number 1, 2014, 275-288

Publisher: Klaipėda University
http://journals.ku.lt/index.php/CST
Online ISSN: 2029-9966
DOI: http://dx.doi.org/10.15181/csat.v2i1.396

MAXIMAL FREQUENT SEQUENCE BASED TEST SUITE REDUCTION THROUGH DU-PAIRS

Narendra Kumar Rao Bangole[1], RamaMohan Reddy Ambati[2]
[1]JNTU College of Engineering, JNTUH, Hyderabad, India
[2]Sri Venkateswara University College of Engineering, Tirupati, India
narendrakumarrao@yahoo.com, ramamohansvu@yahoo.com

Abstract. The current work illustrates the importance of clustering the frequent items of code coverage during test suite reduction. A modular most maximal frequent sequence clustered algorithm has been used along with a requirement residue based test case reduction process. DU-pairs form the basic code coverage requirement under consideration for test suite reduction. This algorithm farewell when compared with few other algorithms like Harrold Gupta and Soffa (HGS) and Bi-Objective Greedy (BOG) algorithms and Greedy algorithms in covering all the DU-Pairs. The coverage criteria achieved is 100% in many cases, except for few insufficient and incomplete test suites.

Keywords: software testing, test case, code coverage criteria, test suite, test suite reduction.

Introduction

Software testing intends to disclose as many defects in the system under test to deliver a quality product to its customers. Software under test undergoes changes due to modifications in code and change in requirements. Hence, the testing counterpart has no alternative other than writing test cases for the modified system. In this scenario the size of the test suite for the system grows tremendously and becomes difficult to control it.

The problem arises where the test cases have to be reduced based on the redundancy and similarity without affecting defect detection capability of the existing test suite. The aim of research in this direction is to determine minimal set of test cases which exercises all the coverage requirements as the entire suite is capable of achieving. This results in minimal number of test cases and hence reduced time and cost for testing.

Code coverage based test suite reduction involves exercising of all the components of the system under study. The components can be methods, conditional statements, statements, classes and few other advanced components. From literature it is learnt that test cases which do not satisfy the code coverage requirements are less effective than rest of other types of test cases and coverage reduced test suites are capable of controlling the size of the test suites (Harrold et al., 1993).

There have been quite a few test suite reduction techniques based on code coverage criteria. The common drawback that needs to be addressed is to cover all coverage requirements without residue and handle an event such that when a tie occurs between two test cases. Few residual requirements (through DU-pairs) are not covered by test suite

reduction approaches (few). This can be handled by additional test cases without compromising for optimal test case reduction.

In current work, test cases are clustered based on maximal frequent item based clustering before the actual test suite reduction process. Clustering process reduces the search space for the test suite reduction. The actual test suite reduction is performed by residue requirements based process, where in test cases chosen for coverage requirement is a maximal code coverage test case from a given cluster of test cases of the test suite.

1. Related Work

Test suite reduction problem is attributed for a NP-complete problem similar to set-cover problem. There has been sufficient research through (Agrawal, 1999; Black et al., 2004; Errol and Brian., 2005; Harrold et al., 1993; Jeffrey and Gupta, 2005; Jones and Harrold, 2003; Offutt et al., 1995; Saeed and Alireza, 2010; Tallam and Gupta, 2005; Scott and Memon, 2007) addressing the test suite reduction. It aims at removing redundant test cases from the test suite and achieves completeness in testing through covering all the code requirements of the system under test, further these test cases should be capable of revealing defects.

Harrold et al. has proposed test suite reduction based on cardinality to its reduction. The reduction procedure is based on selection of singleton test cases from a test set until all test cases in the increasing order of cardinality and coverage requirements..

Saeed and Alireza, proposed Bi-Objective Greedy algorithm for test suite reduction, which satisfies maximum number of requirements of coverage with minimum overlap in coverage of requirements with other test cases. This algorithm is evaluated using space program from SIR repository to identify effects on fault detection capability and size reduction metrics. This algorithm proceeds in three steps and is defined as follows:

The three steps are briefly defined as follows:

Step 1: Matrix transpose is performed to identify and indicate the number of requirements coverage overlapping of a test case with others.

Step 2: Selection of optimum test case until all testing requirements are covered or visited.

Step 3: Updates the cumulative selected vector with respect to the selected test case and cumulative coverage of the reduced suite is updated.

Preethi and Nedunchezhian proposed an approach for selection of requirements for data-flow testing. A Coverage based test Suite reduction algorithm is introduced to cover all the def-use pairs during testing. A Greedy approach was used for the purpose and this algorithm takes optimal test cases into account for reduction.

Researchers targeted on code coverage items to measure and analyze test coverage. There are many coverage item types like statement, branch, block, decision, condition, method, class, package, requirements, and data flow coverage etc...The following Table-1 classifies

the items based for code coverage taken up by several authors. Code coverage has been discussed in depth in following table (Table-1).

Table 1. Code coverage Items.

Author	Year	Code Coverage Items
Díaz and Blanco	2004	branch
Lormans	2005	requirements
Angeletti and Giunchiglia	2005	line, condition, method
Lingampally et al.	2007	branch, block, method, predicate
Kapfhammer and Soffa	2008	data flow coverage
Koochakzadeh	2010	method, class, package

Survey on code coverage criteria clustering is as in (McMaster and Memon, 2008; Dickinson and David, 2009; Khalilian and Saeed, 2009; Shin Yoo et al. 2009; Arvind, 2012; Carlson et al., 2011; Arafeen and Hyunsook Do, 2013; Vipindeep et al., 2009; Yi Miao et al., 2012) are discussed.

2. Problem Definition

The Problem of test suite reduction is defined as given below:

- A test suite T comprises of test cases $\{t_1, t_2, t_3,...t_n\}$ also known as universal set capable of testing all coverage requirements, but not optimally.
- Test requirements are the items represented by $R = \{r_1, r_2, r_3,...r_n\}$ of the desired program under consideration.
- Subsets of clusters or test case sets satisfy corresponding to requirements r_i.

3. Proposed Approach

3.1 DU pairs

A def-use pair (DU) for variable *x* is a pair of nodes (n_1, n_2) such that

- *x* is in DEF(n_1).
- The definition of x at n1 reaches n_2.
- *x* is in USE(n_2).

In other words, the value that is assigned to *x* at n_1 is used at n_2, since the definition reaches n_2, the value is not killed along some path $n_1...n_2$.

3.2 Data Structures

SBT_{ij} - Statement block table
RTC_{ij} - Test case *vs* Blocks table
RTS_{ij} - Cluster of test cases
DU-List - Def-use pair coverage list
$DU\text{-}List_{temp}$ - Def-use pair temporary list

$freqCount_m$ – Frequency count of coverage items in RTC_{ij}
A_i - Blocks of code.
TS_i - Test Suite
D_i - List of Definition variables
U_i - List of Usage variables
R_{safe} - Set of selected test cases after test suite reduction.
t_k - test case

3.3 *Approach*

1 Identify and tabulate the given program statements as blocks (Statement vs Blocks Table) SBT_{ij}.
2 From the given program generate the *D, U* Lists.
   a Populate the *D - List* with statement numbers of declaration and assignment statements of form (*a*:=).
   b Populate the *U - List* with statement numbers of computation and usage of variable of form (:=*a*).
3 Classify the DU as blocks based on statement number in DU list and generate valid combinations of DU-pairs such that a pair is formed with(A_i, A_j) and i > j (If required manually validate the DU-pairs).
4 Generate a visited bit-vector of size of number of DU pairs and correspond it as *DU-List*.
5 Populate the *DU-List* with valid DU pairs and store the list in DU-$List_{temp}$, and mark all the values as 0 (un-visited).
6 Populate RTS and perform Most Maximal Frequent Trace Clustering (MMFTC) of test cases.
7 From the clusters eliminate all redundant test cases with same block flow and coverage (RedntEliminate).
8 From clusters generated by MMFTC, for all test cases and generate the residue DU-pairs for all the test cases in the given MMFTC Cluster by using following steps.
   a Mark the number of blocks or DU-pairs contained in the test case by using $DU\text{-}List_{temp}$.
   b Count the number of blocks selected by a given test case.
   c If there are few DU pairs such that (A_i, A_j) and (A_k, A_j) such that i<k, then retain (A_k, A_j) and mark the corresponding DU pair as visited and unmark (A_i, A_j) in $DU\text{-}List_{temp}$.
   d If a tie occurs for maximum number of blocks covered by a test case, then select a test case of maximal length from the given cluster.
9 Perform *OR* operation on *DU-List* and $DU\text{-}List_{temp}$ to retain previous visited set of DU-pairs and retain test cases in R_{Safe}.
10 Repeat steps from 5 until all the requirements are marked visited in DU-List or test cases have been selected from Test Suite or

Figure 1 represents the step-wise graphical flow representation of approach presented in section 3.3.

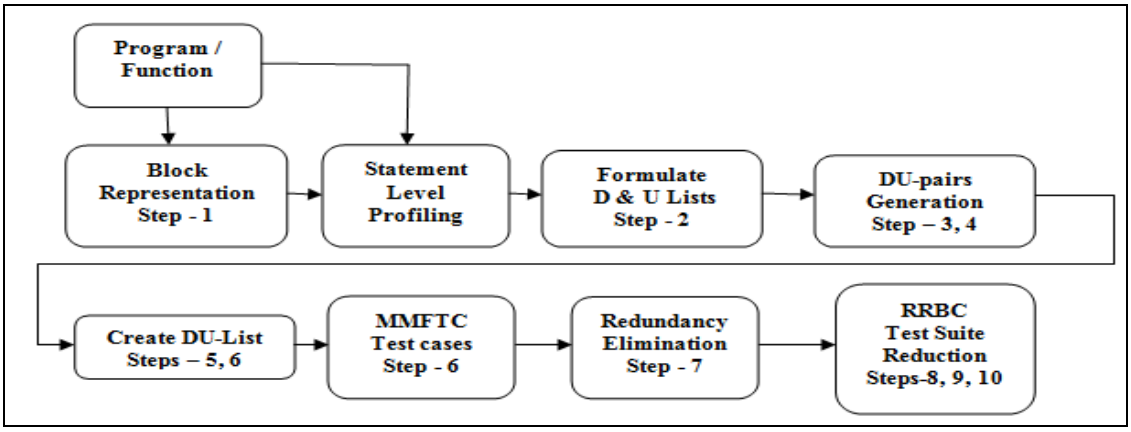

Figure 1. Flow Representation of the approach.

3.4 *Algorithms*

Algorithm 1: Most Maximal Frequent Test case Clustering (MMFTC)

Input: RTC_{ij}, $freqCount_m$

Output: RTS_i, *comprising of clustered test cases*

1 For all pair of code items that have not been visited or until all items are visited or marked as visited **do**

2 Assign i, k with max($FreqCount_m$), max($FreqCount_{m-1}$) from RTC_{ij}

$$RTS_{temp}=\{(x_i, x_k)/x_i, x_k \in RT \wedge \{F(x_i, x_k)= \max_{i=0,\, k=0}^{n-1,\, n-2}\{ \cap \ F(x_i, x_k)\}\}$$

//Perform intersection of maximal most frequent coverage.

3 Repeat $F(x_i \cap x_{k,}\, x_{k+1})=F(x_i \cap x_k) \cap F(x_{k+1})$.

//Repeat the process recursively for all coverage items.

4 $RTS_n = RTS_n \cup RTS_{temp}$

//Recursively gathers all the clustered test cases into a cluster.

5 n++;

6 $RTC_{ij} = RTC_{ij} - \{t_j\}$ // Eliminate the test case from further clustering process.

Algorithm 2: Residue Requirements Test Suite Reduction (RRBC)

Input: RTS_{ij}, DU_{kl}, $DU\text{-}List_i$, $DU\text{-}List_{temp(i)}$
Output: R_{Safe}
1 Until all the code requirements are visited or test suite is empty **do**
2 For all test cases in given cluster do the following
3 For all pairs of DU_{kl} check the following
4 Mark block as visited in $DU\text{-}List_{temp}$, iff $DU_{kl} \in RTS^{(i)}_j$.
5 $R_{Safe} = R_{Safe} \cup t_j$; iff $\{t_j / t_j \in RTS_i \wedge \max(\text{count}(DU\text{-}List_{temp}(t_j))) \wedge tj \in R)\}$
//Consider the test case with maximum coverage of DU-pair blocks per test case.
6 If max(count($DU\text{-}List_{temp}(t_j)$) == max(count($\text{DU-List}_{\text{temp}}(t_K)$) then
7 For (A_i, A_j) and (A_k, A_j) such that i<k, then retain (A_k, A_j) and mark the corresponding DU pair as visited and unmark (A_i, A_j) in $\text{DU-List}_{\text{temp}}$.
8 $DU\text{-}List = DU\text{-}List$ OR $DU\text{-}List_{temp}$. //Computes the residual requirements for the next iteration.

Algorithm 3: Redundant Test case Elimination (RedntElmnt)

Input: RTS_{ij}
Output: R_{Safe}
1 For all the clusters do the following
a. For all test cases in RTS_i do
b. If (Length(t_j) == Length(t_k))
c. If (CompareString(t_j, t_k)=0) then test case tj is redundant and eliminate it from *RT*.
d. If (RTS_i={t_k}, append it to RTS_n where n∈|*RTS*|-1.

3.5 *Concept Illustration*

Following is the illustration of the approach in section 3.3. Variable referencing is an essential feature in data flow based testing. A variable can be examined throughout the program in terms of definition and usage, also termed as def-use pair. A variable linked from point of declaration to its point of reference is also known by the name DU-pairs or Definition-Use pair.

Test cases are designed from data flow view point in terms of blocks and internally representing the DU-pairs. DU-pairs are used as criteria to assess the coverage of requirements of the test cases in the current work. In present illustration, DU-pairs criterion is used for comparison with approaches like HGS, BOG, Greedy approach and MMFTC-RRBC approach. A hypothetical program is chosen to demonstrate the MMFTC-RRBC as described in section below.

Sample Program

Line No	Program	Block-Identifier

1
2
3
4
	Start program
	int *a, neg_no*;
	scanf("%d", &*a*);
	if (*a*>0)
		A_1

5	printf("number is positive-%d", *a*);	A_2

6
7
8
	else
	printf("number is negative-%d", *a*);
	neg_no=a;
		A_3

9	*a*=num_convert(*neg_no);*	
10	printf("natural number determined");	A_4
11	if (*a*%2==0)	A_5
12	printf ("number even-%d", *a*);	A_6
13 14	else printf ("number odd-%d", *a*);	A_7
15 16	printf("type of number determined"); end program	A_8

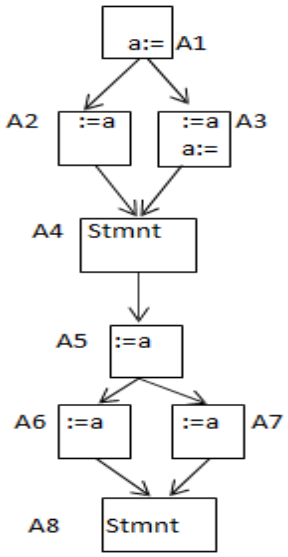

Table 2. Blocks vs Statements

Blocks	***Statements***
A_1	1,2,3,4
A_2	5
A_3	6,7,8,9
A_4	10
A_5	11
A_6	12
A_7	13,14
A_8	15

Figure 2. Blocks representation of Sample Program.

Identify the Blocks and Statements from program:
D - List of Statements-{{2,3},9}
U - List of Statements-{4, 5,{7,8},11,12,14}
D- Blocks list - {A_1, A_3} from Table-2.
U- Blocks list - {A_1, A_2, A_3, A_5, A_6, A_7}

Valid DU-Pairs:{$(A_1,A_2),(A_1,A_3),(A_1,A_5),(A_1,A_6),(A1,A_7),(A_3,A_5),(A_3,A_6),(A_3,A_7)$}

Block-wise test case traces:
T_1->A_1-A_2-A_4-A_5-A_6-A_8
T_2->A_1-A_2-A_4-A_5-A_7-A_8
T_3->A_1-A_3-A_4-A_5-A_6-A_8
T_4->A_1-A_3-A_4-A_5-A_7-A_8

Table 3. Test case-Code coverage Requirements

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	1	2	0	3	4	5	0	6
T_2	1	2	0	3	4	0	5	6
T_3	1	0	2	3	4	5	0	6
T_4	1	0	2	3	4	0	5	6
Freq	4	2	2	4	4	2	2	4

Most Maximal Frequent Clustering f test cases (From Table 3):
F(A_1)∩ F(A_4) = {T_1, T_2, T_3, T_4}
F(A_1∩A_4)∩ F(A_5) = {T_1, T_2, T_3, T_4}
F(A_1∩A_4∩A_5)∩ F(A_8) = {$T_1, T_2, T_3. T_4$}
F(A_1∩A_4∩A_5∩A_8)∩ F(A_2) = {T_1, T_2}
F(A_1∩A_4∩A_5∩A_8∩A_2) = {T_1, T_2}
||| ly F(A_1∩A_4∩A_5∩A_8∩A_3) = {T_3,T_4}
Residue Requirement based Test case reduction is illustrated as below:
CLUSTER-I - {T_1, T_2}
CLUSTER-II - {T_3, T_4}

Table 4. DU-List(Initial)

A_1-A_2	A_1-A_3	A_1-A_5	A_1-A_6	A_1-A_7	A_3-A_5	A_3-A_6	A_3-A_7
0	0	0	0	0	0	0	0

CLUSTER–I
T_1->A_1-A_2-A_4-A_5-A_6-A_8
T_2->A_1-A_2-A_4-A_5-A_7-A_8

Table 5. DU-List$_{temp}$

D-U	Visited
A_1-A_2	1
A_1-A_3	0
A_1-A_5	1
A_1-A_6	1
A_1-A_7	0
A_3-A_5	0
A_3-A_6	0
A_3-A_7	0
# pairs	03

Table 6. DU-List$_{temp}$

D-U	Visited
A_1-A_2	1
A_1-A_3	0
A_1-A_5	1
A_1-A_6	0
A_1-A_7	1
A_3-A_5	0
A_3-A_6	0
A_3-A_7	0
#pairs	03

T_{ie} between T_1-T_2 since #pairs = 3. T_1-selected.R_{Safe} = {T_1}

Table 7. DU-List

A_1-A_2	A_1-A_3	A_1-A_5	A_1-A_6	A_1-A_7	A_3-A_5	A_3-A_6	A_3-A_7
1	0	1	1	0	0	0	0

CLUSTER-II
T3->A_1-A_3-A_4-A_5-A_6-A_8 *T4->A_1-A_3-A_4-A_5-A_7-A_8*

Table 8. DU-List$_{temp}$

D-U	Visited
A_1-A_3	1
A_1-A_7	0
A_3-A_5	1
A_3-A_6	1
A_3-A_7	0
#pairs	03

Table 9. DU-List$_{temp}$

D-U	Visited
A_1-A_3	~~1~~ 0
A_1-A_7	0
A_3-A_5	0
A_3-A_6	0
A_3-A_7	1
#pairs	01

T_3 - selected since #pairs-03.
R_{Safe} = {T_1, T_3}.

Table 10. DU-List

A_1-A_2	A_1-A_3	A_1-A_5	A_1-A_6	A_1-A_7	A_3-A_5	A_3-A_6	A_3-A_7
1	1	1	1	0	1	1	0

CLUSTER-I
T_2-> A_1-A_2-A_4-A_5-A_7-A_8

Table 11. DU-List$_{temp}$

D-U	Visited
A_1-A_7	1
A_3-A_7	0

Table 12. DU-List

A_1-A_2	A_1-A_3	A_1-A_5	A_1-A_6	A_1-A_7	A_3-A_5	A_3-A_6	A_3-A_7
1	1	1	1	1	1	1	0

T_2 -Selected. No other test cases in cluster to compare residue requirements.
R_{Safe} = {T_1, T_3, T_2}

CLUSTER-II
T_4->A_1-A_3-A_4-A_5-A_7-A_8

Table 13. DU-List$_{temp}$

D-U	**Visited**
A_3-A_7	1

Table 14. DU List (Final)

A_1-A_2	A_1-A_3	A_1-A_5	A_1-A_6	A_1-A_7	A_3-A_5	A_3-A_6	A_3-A_7
1	1	1	1	1	1	1	1

T4 -Selected. No other test cases in cluster to compare residue requirements.
R_{Safe} = {T_1, T_3, T_2, T_4}.

In proposed method, MMFTC-RRBC, test cases are clustered based on frequency of coverage items (blocks) in this case. The traces are stored as in table-3 with requirements representing the columns of the mesh format. The output of MMFTC algorithm is clusters containing test cases C_1- {T_1, T_2} and C_2-{T_3, T_4}. After clustering step Residual Requirements based clustering is in effect and test case reduction starts by considering the distinct DU-pairs or code coverage items satisfied by the corresponding test case. A test case satisfying most distinct DU-blocks corresponds to a selected test case. This is illustrated by table- 4-14 above. The same illustration is applied to HGS, BOG and Greedy approaches

and facts are recorded (the approach is discussed in introduction).

The test cases selected are stored in R_{Safe}, ensuring that all blocks and DU-pairs were covered by the approach. Blocks and DU-pairs covered by each of the approaches is illustrated in table-15. The table hypothecates the fact that for current scenario, Greedy approach and MMFTC-RRBC fare well in terms of DU-pair coverage.

Table 15. Comparison of Approaches.

Approach	%DU pairs	Test cases Selected	DU-pairs not covered
HGS	8	$\{T_1, T_3, T_4\}$	(A_1,A_7)
BOG		$\{T_1, T_3, T_4\}$	(A_1,A_7)
Greedy		$\{T_1, T_2, T_3, T_4\}$	*None*
MMFTC-RRBC		$\{T_1, T_2, T_3, T_4\}$	*None*

4. Experiment Analysis

An Empirical study was carried out with five different program ranging from 12-24 lines of code to understand the effectiveness of the proposed MMFTC-RRBC approach. DU-coverage criteria are chosen for establishing the approach. All DU-pairs were program instrumented. All these programs were implemented in C-language, since paradigm is not the point of consideration in our current work.

Following metrics were point of focus for evaluation of the current work to evaluate the efficacy of the approach:

a. Code Requirement Coverage percentage:

R_{tot} - represents the total number of test consideration requirements and R_{cov} is the total number of requirements satisfied by test cases selected during reduction.

$$R_{COV(\%)} = \frac{|R_{\text{cov}}|}{|R_{\text{tot}}|} \times 100 \quad (1)$$

b. Percentage Reduction in Size of Test Suite

$$TS_{(\%\,\text{Re}\,d)} = 1 - \frac{|T_{\text{rs}}|}{|T|} \times 100 \quad (2)$$

$|T|$- total number of test cases in the original suite and $|T_{rs}|$ represents number of test cases in representative set.

The size metric was evaluated for the approaches as described like HGS, BOG, Greedy and MMFTC-RRBC, observation was that Greedy approach fare well with MMFTC-RRBC as represented in the Figure 3.

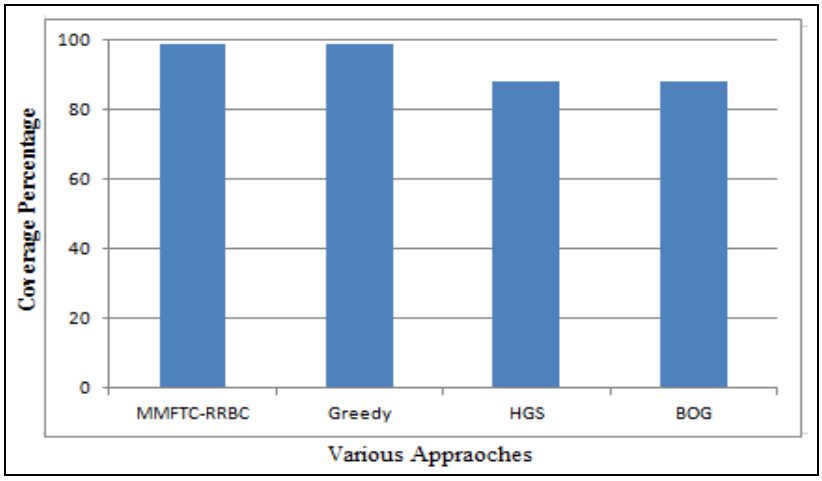

Figure 3. Comparison based on requirement coverage metric.

Test Suite requirement coverage was compared using Mean DU detected and Mean % size reduced metrics. Table - 16 represents the program versus reduction algorithm performance based on DU-pairs detection. The results of this comparison are represented in Figure 4.

Table 16. Programs used for Evaluation and comparison.

Program Name	LOC	# Test cases	# Detectable DU pairs	# DU pairs detected (MMFTC-RRBC)	# DU pairs detected (Greedy Approach)	# DU pairs detected (BOG Approach)	# DU pairs detected (HGS Approach)
Odd_Even	14	4	8	8	8	7	8
Sum_Digits	12	3	4	4	4	4	4
Cost Cal	24	6	5	5	5	4	4
Interest Calculator	16	4	7	7	7	6	6
Prime Number	20	5	4	4	4	4	3
Total			28	28	28	25	25
Percentage				100	100	89.2	89.2

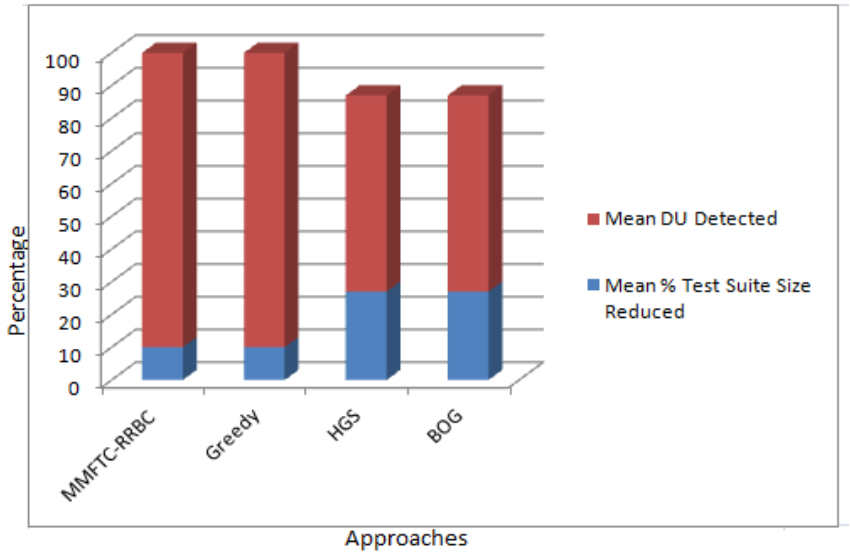

Figure 4. Comparison based on test suite requirement coverage.

Conclusion

In this current work, we proposed MMFTC-RRBC using DU-pairs. Approach is capable of achieving complete requirement coverage effectively. The proposed approach is simple and tries to solve the purpose and is extendible to all complex variables of the program. Fault detection effectiveness metric is a possible extension to the current work.

References

Arafeen, M.J., Hyunsook, Do (2013). Test Case Prioritization Using Requirements-Based Clustering. Software Testing,Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, vol., no., pp.312-321.

Agrawal, H. (1999). Efficient Coverage Testing Using Global Dominator Graphs. PASTE '99 Proc. Workshop Program Analysis for Software Tools and Engineering, Toulouse, France, vol. 1, pp. 11-20.

Angeletti, D., E. Giunchiglia (2009). Automatic Test Generation for Coverage Analysis of ERTMS Software. *Software Testing Verification and Validation, 2009. ICST '09. International Conference on* , vol., no., pp.303-306.

Arafeen, M.J., Hyunsook, Do (2013). Test Case Prioritization Using Requirements-Based Clustering. Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, vol., no., pp.312-321.

Arvind, K. (2012). Pair-Wise Time-Aware Test Case Prioritization for Regression Testing. Information Systems, Technology and Management Communications in Computer and Information Science, vol., no.285, pp.176-186.

Black, J., Melachrinoudis, E., Kaeli, D. (2004). Bi- Criteria Models for All-Uses Test Suite Reduction. ICSE '04 Proc. 26th International Conference on Software Engineering, Edinburgh, United Kingdom, pp.106-115.

Carlson, R., Hyunsook, Do., Denton A. (2011). A clustering approach to improving test case prioritization: An industrial case study. Software Maintenance (ICSM), 2011 27th IEEE International Conference on, vol., no., pp.382-391.

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

Eugenia, Díaz, J.T., Raquel Blanco (2004). A Modular Tool for Automated Coverage in Software Testing. Software Engineering Notes, vol. 26, no.5, pp. 256-267.

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

Jeffrey, D., Gupta, N. (2005). Test Suite Reduction with Selective Redundancy. In the Proc. 21st IEEE International Conference on Software Maintenance, Budapest, Hungary, vol.3, pp. 549-558.

Jones, J.A., Harrold, M.J. (2003). Test-Suite Reduction and Prioritization for Modified condition/ Decision Coverage. IEEE Transactions on Software Engineering, vol. 29, no. 3, pp. 195-209.

Lormans, M.D. (2005). Reconstructing Requirements Coverage Views from Design and Test using Traceability Recovery via LSI. TEFSE, Long Beach, California, USA, ACM.

Lingampally, R., Gupta, A., Jalote, P. (2007). A Multipurpose Code Coverage Tool for Java. System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on , vol., no., pp.261b,261.

Kapfhammer, G.M., Soffa, M.L. (2008). Database-Aware Test Coverage Monitoring. Proceedings of the Ist India Software engineering conference ISEC'08, Hyderabad, India, ACM, vol.1, 77-86.

Khalilian, A., Saeed, P. (2009). Bi-criteria test suite reduction by cluster analysis of execution profiles. In Proceedings of the 4th IFIP TC 2 Central and East European conference on Advances in Software Engineering Techniques (CEE-SET'09), Tomasz Szmuc, Marcin Szpyrka, and Jaroslav Zendulka (Eds.). Springer-Verlag, Berlin, Heidelberg, 243-256.

Koochakzadeh V. G. (2010). An Empirical Evaluation to Study Benefits of Visual versus Textual Test Coverage Information. Lecture Notes in Computer Science, vol. 6303, pp 189-193.

McMaster S., Memon A.M., (2008). Call-Stack Coverage for GUI Test Suite reduction. Software Engineering, IEEE Transactions on, vol.34, no.1, pp.99-115.

Offutt, A.J., Pan, J. and Voas, J.M. (1995). Procedures for Reducing the Size of Coverage-Based Test Sets. In Proc. of the 12th International Conference on Testing Computer Software, Washington, USA, vol.2, pp. 111-123.

Preethi, H., Nedunchezhian, R. (2014). A Greedy Approach for Coverage-Based Test Suite Reduction. International Arab Journal of Information Technology, vol.11, issue-1.

Parsa, S., Khalilian, A., (2010). On the Optimization Approach towards Test Suite Minimization", International Journal of Software Engineering and Its Applications Vol. 4, No. 1, pp. 15-28.

Shin Yoo, Mark Harman, Paolo T, Angelo Susi (2009). Clustering test cases to achieve effective and scalable prioritization incorporating expert knowledge. In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, NY, USA, 201-212.

Saeed, P., Alireza, K. (2010). On the Optimization Approach towards Test Suite Minimization. International Journal of Software Engineering and its Applications, vol. 4, no.1, pp. 15-28.

Tallam, S., Gupta, N. (2005). A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization. Proc. of the 6th ACM SIGPLANSIGSOFT workshop on Program Analysis for Software Tools and Engineering, Lisbon, Portugal, vol.2, pp. 35-42.

Vipindeep, V., Jacek Czerwonka, Phani Talluri (2009). Test case comparison and clustering using program profiles and static execution. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE '09). ACM, New York, NY, USA, 293-294.

William Dickinson, David Leon, (2009). Finding Failures by Cluster Analysis of Execution Profiles. Intl. Conf. on Software Engineering, Toronto, Candada, pp. 339-348.

Xue-ying, Ma et al. (2005). A genetic algorithm for test suite reduction. In Proc. IEEE International Conference on Systems, Man and Cybernetics, pp. 133-139, Hawaii, USA.

Yi Miao, Zhenyu Chen, Sihan Li, Zhihong Zhao, Yuming Zhou (2012). Identifying Coincidental Correctness for Fault Localization by Clustering Test Cases. International conference on Software Engineering and Knowledge Engineering, 267-272.

Narendra Kumar Rao Bangole obtained Bachelor Degree in Computer Science and Engineering from University of Madras, M.Tech in Computer Science from JNTU, Hyderabad and at present pursuing Ph.D. He has more than 13 years of experience in Area of Computer Science and Engineering which includes four years of Industrial Experience and NINE years of Teaching Experience. Research interests include Software Engineering and Embedded Systems. Currently he is working as Associate Professor in Department of Computer Science and Engineering at Sree Vidyanikethan Engineering College.

RamaMohan Reddy Ambati obtained his Bachelor Degree in Mechanical Engineering and Master's degree in Computer Science Engineering from NIT Warangal and Ph.D degree from Sri Venkateswara University and at present working as a Professor in Department of Computer Science and Engineering, Sri Venkateswara University College of Engineering. His areas of interest include Software Architecture and data mining. He has more than 28 years of experience in teaching and research.

MAKSIMALIAI DAŽNOMIS SEKOMIS GRINDŽIAMA TESTINIO RINKINIO REDUKCIJA NAUDOJANT DU-PORAS

Narendra Kumar Rao Bangole, RamaMohan Reddy Ambati

Santrauka

Šiame darbe nagrinėjamas dažnų kodo uždengimo elementų svarbumas naudojant bandomojo rinkinio redukciją. Buvo naudojamas modulinis didžiausio dažnio sekos grupuojantis algoritmas, turint omenyje paklaidos likučio mažinimo reikalavimą. DU-poros suformuoja pagrindinį kodo uždengimo reikalavimą, kurio toliau paisoma bandomojo rinkinio redukcijos proceso metu. Šio algoritmo veikimas buvo palygintas su tokiais algoritmais kaip antai Harrold Gupta ir Soffa (HGS), ir Bi-Objective Greedy (BOG) algoritmu bei Greedy algoritmu, kai yra uždengiamos visos DU-poros. Daugeliu atveju pasiektas 100% uždengimo kriterijus, išskyrus kelis bandymus su nepakankamais ir nepilnais bandomaisiais rinkiniais.

Pagrindiniai žodžiai: programinės įrangos testavimas, testavimo atvejis, kodo uždengimo kriterijai, testinio rinkinio redukcija.