# A METRIC BASED EVALUATION OF TEST CASE PRIORITATION TECHNIQUES-HILL CLIMBING, REACTIVE GRASP AND TABUSEARCH

**[1]M.Manjunath, [2]N.Backiavathi**
[1]PG Scholar, Department of Information Technology,Jayam College of Engineering and Technology, Dharmapuri.
[2]Assistance. Professor, Department of Information Technology,Jayam College of Engineering and Technology, Dharmapuri.
[1]idealmanju@gmail.com, [2]ishaaliniiha@gmail.com

**Abstract:** Regression Test Selection Technique attempt to reduce the cost of Regression testing by selecting and running a subset of an existing test suite. The goal of the Regression Test Selection Technique is to select the Reduced Test Suite for the modified version to minimize the cost of the maintenance phase. Total Statement Coverage approach involves ranking Test Cases from the Reduced Test Suite based on the number of Statements covered by the Test Case such that the Test Case covering the maximum number of Statements would be executed first. The test case prioritization process is done in three different approaches, the first approach is based on Instrumented code, in this the test case that covers maximum changed statements will execute first. The second prioritization method is based on Hill Climbing algorithm and the third prioritization method follows Tabu based metaheuristic search procedure. The validation of the three prioritization technique is evaluated with APSC(Average Percentage of Statement Coverage) and APDC(Average Percentage of Decision Coverage).

**Keywords-**Regression Test, Metaheuristic Search, Average Percentage of Statement Coverage, Average Percentage of Decision Coverage

## 1. INTRODUCTION

Regression testing is applied to the modified version of the software to ensure that it behaves as intended, and that modifications have not adversely impacted its quality. Rerunning the entire Test Suite of the original version to test the modified version increases the cost of Regression testing. Control Flow Graph is generated from the program and the Cyclomatic Complexity is calculated. Independent Control Flow paths are extracted from the CFG. Test cases are generated based on the Cyclomatic Complexity. Each independent Control Flow path is assigned to the Test case. The Test cases that have impact on the Changed entities are selected from the Test Suite of the original version. Regression Test Selection technique saves the cost of Regression testing by selecting only subset of Test cases that has impact on the modified version.

The reduced Test cases are prioritized based on the Statement Coverage. Total Statement Coverage approach involves ranking Test cases based on the number of statements covered by the Test case such that the Test case covering the maximum number of statements would be executed first. This approach first selects the Test case with the maximum Statement Coverage, adjusts the Coverage information on the remaining Test cases to reflect the statements not covered by that Test case, and then iteratively selects a Test case that provides the largest Statement Coverage until all Program statements have been covered. Few drawbacks of the existing system are as follows.

Regression testing that reuses the coverage data collected when Test suite Ti is run on the original version for testing subsequent versions so that the expense of re-computing it for each subsequent version of the original version is avoided[7,8]. Test suite can be reused when regression test is executed on modified versions, so that time and resources of generating test cases can be reduced. But it is inefficient as it executes the entire test suite for the subsequent modified versions.

- A safe regression-test-selection technique selects every test case from the original test suite that can expose fault in the modified program.
- Regression-Test-Selection techniques are particularly effective in environments in which changed software is tested frequently.
- Regression testing tasks, based on original version, may be inaccurate for subsequent modified versions.

## 2. RELATED WORK

RECOVER (Re-computing Coverage Data), that implements our technique, along with a set of empirical studies conducted on a set of Java programs ranging from 1 to 75 KLOC. These studies show the inaccuracies that can exist in results of an application RTS when the outdated or estimated coverage data are used. For the six subjects we used, RTS applied to outdated coverage data resulted in, on average, 42.51, 80.26, 83.12, 82.38, 75.41, and 99.01 percent false positives,4 respectively, and 14.61 percent false negatives 5 over RTS used with updated coverage data. For the six subject we used, RTS applied to estimated coverage data resulted in, on average, 0.68, 54.52, 70.12, 75.30, 68.96, and 90.56 percent false positives, respectively, and 9.28 percent false negatives over RTS used with updated coverage data. The studies also show the efficiency of our technique. For the six subjects we used, when RTS is augmented with our technique to compute the mappings and selectively instrumented programs are run with the test cases selected, the overall regression testing time is reduced, on average, 11.53 percent over RTS and 65.74 percent over retest-all. The main contributions of this paper are a description of a novel technique that computes accurate,

updated coverage data when a program is modified, without rerunning unnecessary test cases, a discussion of a tool, RECOVER, that implements the technique and integrates it with RTS, and a set of empirical studies that show, for the subjects we studied, that our technique provides an effective and efficient way to update coverage data for use on subsequent regression-testing tasks.

**Example: (V0)**

```
Public class Grade{
Public int calcGrade(int finalsScore, int MidTermScore)
{
S1 int Grade=0;
S2 if (finalScore>70){
S3 if(midTermScore>80){
S4 grade=4; } else {
S5 grade=3; }
S6 } else if (final score <50){ S7 grade = 2;} else{
S8 grade = 3;}
S9 System.out.println("Grade=" +grade);
S10 return grade;
```

**Table 1: V0 –Statement Coverage Matrix**

| | Original Coverage | | | |
|---|---|---|---|---|
| | $t1$ | $t2$ | $t3$ | $t4$ |
| $s1$ | 1 | 1 | 1 | 1 |
| $s2$ | 1 | 1 | 1 | 1 |
| $s3$ | 1 | 1 | 0 | 0 |
| $s4$ | 1 | 0 | 0 | 0 |
| $s5$ | 0 | 1 | 0 | 0 |
| $s6$ | 0 | 0 | 1 | 1 |
| $s7$ | 0 | 0 | 1 | 0 |
| $s8$ | 0 | 0 | 0 | 1 |
| $s9$ | 1 | 1 | 1 | 1 |
| $s10$ | 1 | 1 | 1 | 1 |

To illustrate the impact that changes can have on the Coverage information, consider above example which shows version v0 and subsequent versions v1 and v2, respectively, of a program consisting of class Grade and method calcGrade. Version v1 shows changes c1and c2 from v0 and version v2 shows change c3 from v1.The test suite T for calcGrade is shown in Table 1. Figs. 1, 2, and 3 also show the corresponding coverage matrices based on statements (i.e., statement coverage matrices) for the versions. In the matrices, for a particular test case, indicates that a statement was covered during execution of ti and "0" indicates that a statement was not covered during execution of ti. For version v0 (shown in Fig. 1), the matrix shows the original coverage data because T is run with the base version of the program (i.e., v0); note that version v0 has 100 percent statement coverage with respect to T. For versions v1 and v2 (Figs. 2 and 3, respectively), the matrix on the left shows the outdated coverage data when the coverage data for v0 are used for the subsequent versions, and the matrix on the right shows

the updated coverage.

**Example: (V1)**

Public class Grade{ Public int calcGrade(int finalsScore, int MidTermScore) {
S1 int Grade=0;
S2 if (finalScore>70){
S3 if(midTermScore>80){
S4 grade=4; } else {
S5 grade=3; }
**S6 } else if (final score <60){// change c1**
**S7 grade = 3;**
**S8 } else if(final score <35){// change c2**
**S9 grade =1; } else{**
**S10 grade =2;}**
S11 System.out.println("Grade=" +grade);
S12 return grade;

**Table 2: V1 Statement Coverage matrix**

| | Outdated Coverage | | | | | Updated Coverage | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *t*1 | *t*2 | *t*3 | *t*4 | | *t*1 | *t*2 | *t*3 | *t*4 |
| *s*2 *s*3 | 1 1 | 1 1 | 1 0 | 1 0 | *s*2 *s*3 | 1 1 | 1 1 | 1 0 | 1 0 |
| *s*4 | 1 | 0 | 0 | 0 | *s*4 | 1 | 0 | 0 | 0 |
| *s*5 | 0 | 1 | 0 | 0 | *s*5 | 0 | 1 | 0 | 0 |
| *s*6 *s*7 | 0 0 | 0 0 | 1 1 | 1 0 | *s*6 *s*7 | 0 0 | 0 0 | 1 0 | 1 0 |
| *s*8 *s*9 | 0 1 | 0 1 | 0 1 | 1 1 | *s*8 *s*9 | 0 0 | 0 0 | 1 1 | 1 0 |
| *s*10 | 1 | 1 | 1 | 1 | *s*10 | 0 | 0 | 0 | 1 |
| *s*11 *s*12 | ? ? | ? ? | ? ? | ? ? | *s*11 *s*12 | 1 1 | 1 1 | 1 1 | 1 1 |

We consider RTS, which was briefly described in Section

- We use an RTS technique imple-mented as DEJAVOO. DEJAVOO creates control-flow graphs for the original (Porig) and modified (Pmod) versions of a program. The Technique traverses these graphs synchronously over like labeled edges, in Porig and Pmod are such that both edges have no label, a true label, a false label, or a matching label in a switch (or case) statement. The technique performs the travels or in a depth-first order to identify dangerous edges. The edges whose sinks differ and for which test cases in T that executed the edge in Porig should be rerun on Pmod because they may behave differently in Pmod.

**2.1 Study 1:**

The goal of Study 1 is to address research question RQ1. What are the effects of the three techniques for providing coverage data outdated, estimated, and updated on regression test selection (RTS)?

To answer this research question, we used all six subjects described in Section 4.2. For these subjects, we populated outdated, estimated, and updated coverage data. For outdated coverage data, we ran T on v0 to collect m0, the coverage data for version v0 of program P. We then usedm0 for RTS activities on subsequent versions of v0. For estimated coverage data, we used JDIFF [12] to estimate the coverage data and populate miþ1 for each version viþ1 using mi, the coverage data for vi. JDIFF compares two Java programs, vi and viþ1, and identifies both differences and correspondence between the two versions. Because JDIFF uses heuristics to determine differences and correspondences, it can result in both false positives and false negatives. The technique is based on a representation of object-oriented programs that handles

object-oriented features, and thus, can capture the behavior of the program. Using the correspondence, which is a mapping between statements in the two versions, it estimates the coverage for viþ1 using the coverage data from vi and uses it to populate miþ1. For updated coverage data, we used our tool RECOVER to calculate miþ1 for version viþ1 using mi for version vi. Recall that the updated coverage data that our technique computes are identical to those computed if all test cases were rerun. As a check of our RECOVER implementation, we computed the updated coverage data by running all test cases on the versions of P and comparing these accurate coverage data with those obtained using RECOVER. In all cases, the coverage data were the same.

The next three columns show the results when DEJAVOO is run using outdated coverage data the number of test cases selected, the number of false positives4 in that set of test cases, and the number of false negatives5 in that set of test cases. The next three columns show similar results when DEJAVOO is run using coverage data estimated with JDIFF. The last column shows the number of test cases selected by DEJAVOO using updated coverage data (the same coverage data as would be obtained by rerunning all test cases in the test suite). For Jakarta Regexp, ProAX,
and Darpan, the tables show the results of running DEJAVOO on all pairs of versions. For nanoXML and

JABA, the tables show only a representative subset of the results of running DEJAVOO on all pairs of versions

| Subject | Full instrumentation (all branches) | Selective instrumentation (branches reachable from the change) | Savings |
|---|---|---|---|
| Jakarta Regexp | 162 | 158 | 2.46% |
| NanoXML | 596 | 581 | 2.51% |
| ProAX | 19672 | 14573 | 25.92% |
| Assent | 49269 | 49187 | 0.16% |
| JABA | 26310 | 26033 | 1.06% |
| Darpan | 82315 | 81842 | 0.57% |

### 2.2 Study 2

The goal of Study 2 is to evaluate research question RQ3. What is the efficiency of our technique for pdatingcoverage data as part of a regression testing process?

To answer this question, we measured and compared regression-testing time for four approaches: 1. running all test cases in T on all versions of the program P (i.e., retest-all); 2. selecting T0 using DEJAVOO and running the test cases in T0 on all modified versions of P; 3. selecting T0 and recording mappings using MODDEJAVOO, updating coverage data for T , T0 using RECOVER, instrumenting the modified versions of P with full instrumentation, and running the test cases in T0 on the fully instrumented modified versions of P; and 4. selecting T0 and recording the mappings using MOD-DEJAVOO, updating coverage data for T T0 using RECOVER, instrumenting modified versions of P using selective instrumentation, and running test cases in T0 on the selectively instrumented modified versions of P. Table 11 shows the average timings for regression testing for the four techniques studied. In the table, the first column shows the subject on which the experiment was performed. The second column shows the sum of the time to perform RTS usingDEJAVOOand the time to run the selected test cases T0. The third column shows the sum of the time to perform RTS

| Subject | Retest all (time in seconds) | DEJAVOO (time in seconds) | MOD-DEJAVOO + RECOVER (time in seconds) | MOD-DEJAVOO + RECOVER with Selective Instrumentation (time in seconds) |
|---|---|---|---|---|
| Jakarta Regexp | 75.75 | 36.25 | 36.75 | 32.81 |
| NanoXML | 301.81 | 15.12 | 15.39 | 11.63 |
| ProAX | 266.83 | 109.75 | 115.37 | 98.56 |
| Assent | 406.02 | 211.25 | 235.49 | 199.15 |
| JABA | 490.29 | 254.75 | 262.13 | 212.06 |
| Darpan | 1447.91 | 499.82 | 515.33 | 478.9 |

**EVALUATION TEST CASE PRIORITATION:**

This system is to develop a system that provides the Optimized Test Suite of the modified version by selecting a subset of Test cases from the Test Suite of the original version and prioritizing the selected Test cases based on the coverage data of each Test case. The system takes as input the original program and its versions. It then generates Test cases based on the independent control flow paths of the program versions. The Test cases that have impact on the Changed statements in the modified versions are selected from the Test Suite. The Test cases are then prioritized based on the Statement Coverage of the Test cases so that the Test case that covers more number of statements in the program is given the highest priority and is executed first. Thus the tester has no need to run all the Test cases of the original version for the changed statements minimizing the testing time in the maintenance phase.

The problem of maintaining updated coverage data, without incurring the expense of rerunning the entire test suite or the inaccuracy of using outdated or estimated coverage data, a technique is developed that influence existing RTS technique to compute accurate, updated coverage data without rerunning any test cases that do not execute the change.

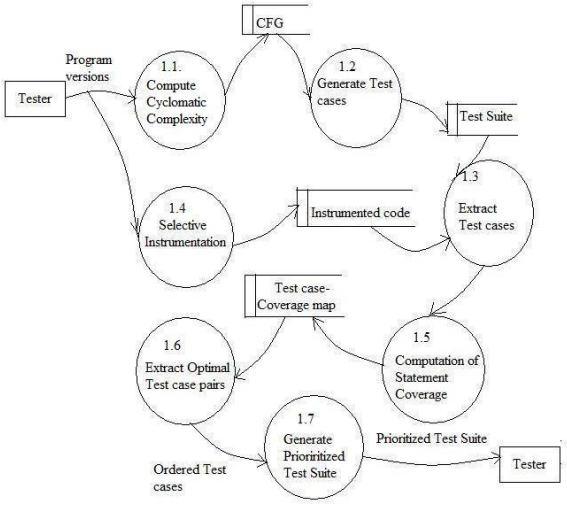


**Figure 1: Nodes Vs Probability of Structural Attack**

**2.3 Test Plan**

The Test Plan is derived from the Functional Specifications, and detailed Design Specifications. The Test Plan identifies the details of the test approach, identifying the associated Test case areas within the specific product for this release cycle.

**2.4 Entity Mapping:**

Test suite reduction system compares the original and modified versions and extracts equivalent statements present in both the versions. It then maps the entities with line numbers whose statement matches between the original and the modified versions.

**2.5 Computation of Cyclomatic Complexity**

Control Flow Graph is generated from the program versions. The number of nodes, edges and the Cyclomatic complexity of the module of the program version are calculated.

**Cyclomatic complexity= E-N+2**

E-Edge, N-Node

**2.6 Instrumentation**

Test suite reduction system performs instrumentation of the changed and affected entities. The changed entities are extracted from the modified version and the entities reachable from the changed entities are also extracted as affected entities. The changed and affected entities are integrated into instrumented code.

**2.7 Reactive GRASP for Test Case Prioritization:**

The Test cases are ordered in the decreasing order of Instrumented Statement Coverage. The Test case pair that covers more number of different statements is given the highest priority.

**Algorithm :**

1: initialize probabilities associated with α (all equal to 1 n)
2: for i = 1 to max iterations do
3: α ← select α (αSet);
4: solution ← run construction phase(α);
5: solution ← run local search phase(solution);
6: update solution(solution, best solution);
7: end;
8: return best solution.

**2.8 Statement Coverage Vs Block Coverage:**

The coverage matrix is used to indicate the statement coverage for each test case. Let the sample coverage matrix for a program set.

**2.9 Statement Coverage:**

This metric reports whether each executable statement is encountered. Control-flow statements, such as if, for, and switch are covered if the expression controlling the flow is covered as well as all the contained statements. Implicit statements, such as an omitted return, are not subject to statement coverage.

**2.10 Block coverage:**

Block coverage is the same as statement coverage except the unit of code measured is each sequence of non-branching statements.

**Table.3 Block Coverage Vs Statement Coverage**

| Program Set | Number of Instrumented Code in LOC (Exsisting System-Block Coverage) | Number of Instrumented Code in LOC (Proposed System-Statement Coverage) |
|---|---|---|
| Program Set-1 | 28 | 35 |
| Program Set-2 | 45 | 56 |
| Program Set-3 | 26 | 36 |
| Program Set-4 | 12 | 23 |
| Program Set-5 | 14 | 26 |
| Program Set-6 | 26 | 52 |
| Program Set-7 | 46 | 86 |
| Program Set-8 | 35 | 46 |
| Program Set-9 | 9 | 13 |
| Program Set-10 | 36 | 59 |

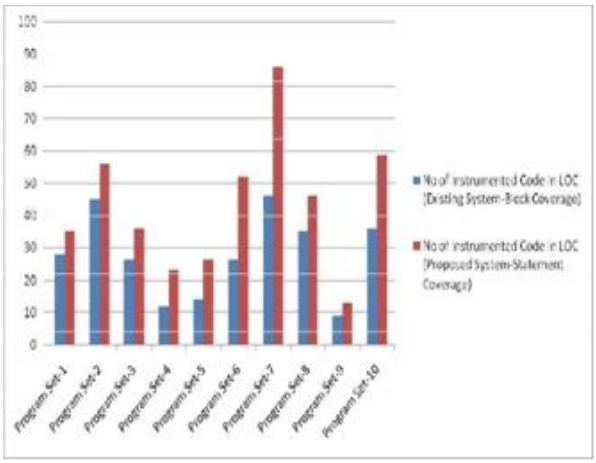


**Figure 2: Graph for Statement Vs Block**

Average Coverage Percentage of test case=

No of Statement Covered by a Test case

Total no of Statements

= 8/12

= 66.67%

**2.11Hill Climbing Based Prioritization :**

The Test cases are generated from the independent Control Flow paths of the program. The Testcases are prioritized based on random search within the Restricted Candidate List.

**Algorithm:**

1: initial solution c=∅
2: initialize the candidate set C with random test cases from the pool of test cases;
3: s ← test case from the RCL at random;
4: while s not locally optimal do
5: Find s ∈ Neighbour (s) with f (s") < f(s);
6: solution ←solution ∪{s};
7: end;
8: return s;

**2.12 Tabu Search:**

The Tabu Search examines a trajectory sequence of solutions and moves to the best neighbor of the current solution. To avoid cycling, solutions that were recently examined are forbidden, or tabu, for a number of iterations.

**Algorithm:**

Input: Problem set P , with |P | = n Test cases
Input: Number of initial solutions required, I
(1) identify I ⊂ P , a randomly identified subset with I nodes from problem set ;
(2) foreach I ∈ I do
(3) P 0 ← P \ {I };
(4) find initial solution s by executing Initial solution heuristic with P0 ;
(5) re-insert I into initial solution to create Ti
(6) end

## 3. RESULTS

The results obtained after performing testing on different program versions, the instrumentation coding test case prioritation technique is efficient for small programs. And Hillclimbing technique is gives only feasible solution for any program testing, so retesting the program becomes must. The best algorithm which suits for small and large programs also and provides best optimum solution.

## 4. CONCLUSION

Regression Test Selection technique that provides updated coverage data for a modified program without re-running all test cases in the test suite that was developed for the original program. The selective instrumentation process instruments only the affected statements, and thus, reduces the amount of instrumentation. By running the test cases selected only for affected statements, the technique updates coverage data for test cases that exercise the change. Using the mapping information provided by the computation of entity map, the technique updates coverage data for test cases that do not exercisechanges. The RTS technique is safe and selects important test cases for the modified version and omits unimportant test cases for the statements that exercise change in the modified version. This reduction results in a savings in the time to run the test cases selected, and thus, reduces the overall regression testing time. The phase I of the project focuses on reducing the test cases for testing the modified version.

## REFERENCES

[1] Gregg Rothermel and Mary Jean Harrold. "Analyzing regression test selection techniques", IEEE Transactions on Software Engineering, Vol.22, No.8, pp 529-551, 1996.

[2] J.-M. Kim, A. Porter, and G. Rothermel. "An empirical study of regression test application frequency", In Proceedings of the 22nd International Conference on Software Engineering, pp 126-135, 2000.

[3] F. Vokolos and P. Frankl, "Pythia: A RegressionTest Selection Tool Based on Text Differencing," Proc. IEEE International Conference on Reliability, Quality and Safety of Software Intensive Systems, pp. 3-21, 1997.

[4] S. Elbaum, D. Gable, and G. Rothermel, "The Impact of Software Evolution on Code Coverage Information." Proc. IEEE International Conference on Software Maintenance, pp. 170-179, 2001.

[5] G. Rothermel, M. J. Harrold, and J. Dedhia. "Regression test selection for C++ software", Journal of Software Testing, Verification, and Reliability, Vol. 10, No.6, pp.77-109,2000.

[6] L. J. White and K. Abdullah." A firewall approach for regression testing of objectoriented software", In Proceedings of 10thAnnual Software Quality Week, 1997.

[7] H.K.N. Leung and L. White. "Insights into Regression Testing." In Proceedings of Conference on Software Maintenance. pp. 60– 69, 2005.

[8] G. Rothermel and M.J. Harrold." Analyzin regression test selection techniques." IEEE Transactions on Software Engineering, Vol.22, No.8, pp.529–551, 1996.

[9] Kewen Li, Zhixia Yang," An Improved AETG Test Suite Optimization Method Based on Regressing Test Model."IEEE Computer Society, 2008.

[10] A. 1Orso, N. Shi, and M.J. Harrold, "Scaling Regression Testing to Large Software Systems", Proceedings of 12th ACM SIGSOFT Symposium on Foundations of Software Engineering. pp. 241-252, 2004.

[11] Mary Jean Harrold and James A. Jones," Regression Test Selection for Java Software" Proc. of the ACM Conf. on OO Programming, Systems, Languages, and Applications ACM Copyright,pp.120-130, 2006.

[12] Guoqing Xu (2006),' A Regression Tests Selection Technique for Aspect-Oriented Programs' Proc. of the ACM Conf. on Software Engineering, ACM Copyright, pp.120-130.