Received: March 15, 2018

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A New Similarity-Based Greedy Approach for Generating Effective Test Suite

Shilpi Singh [1]* **Raj Shree [1]**

[1]*Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, India*
*Corresponding author's Email: shilpi.singh.it@gmail.com

Abstract: Software regression testing is one of the most critical phases of software development life cycle, used by developers with the intent of detecting new faults to validate modified software prior to delivery to the customer. To validate updated features, new test cases are generated by the testers which increment the test suite size automatically. The resulting test suite may contain obsolete, redundant, and ambiguous test cases. Therefore, there is a strong requirement of an intelligent testing approach to reduce the test suite size by removing those unessential test cases economically. This paper proposed an interesting approach, which involves the combination of regression testing techniques: minimization, and prioritization both. The main focus is on multiple regression activities with multiple criteria rather than using only single activity to produce an optimal solution. In this paper clustering approach is also considered, which could simplify and enhance the minimization and prioritization task. To evaluate the effectiveness of the strategy, we performed an experimental investigation together with an eminent heuristic Harrold Gupta and Soffa (HGS), considering the testing measures of the minimized test suite size and fault coverage. The results show that, similarity-based greedy approach with multiple coverage criteria can be quite effective in terms of fault detection loss of reduced test suite without much affecting the percentage of suite size reduction.

Keywords: Regression testing, Test suite, Test cases, Test suite reduction, Test case prioritization, Clustering.

1. Introduction

Software testing is the most commonly used but expensive method for developing quality software by validating the software program [1]. The goal of software testing is to execute the software system, identify the faults that cause failures, and improve the software quality by removing the identified faults. It is a very expensive process as well as the important task of the software development life cycle (SDLC), through which we add some value to the software program [2].

Inadequate testing is one of the major cost factors. Testing efforts, often consume more than half of the overall development resources. Early detection of faults and failure reduces maintenance costs as well as requires fewer corrections [3]. According to the IEEE definition [4], a test case is a collection of input data given to the program and expected output results created to evaluate a software function or test requirement. It is hard for a single test case to satisfy the coverage of entirely given test requirements. That is why; a number of test cases are generated and collected in a test suite [5]. Because of the extensive use of testing to measure the quality of the software, one of the challenges faced by organizations is the test suite optimization [6].

Software regression testing processed continuously during the software development and maintenance of evolving software. Maintenance requires some modifications, which leads to growth in software and it results in an increment in test suite size. Over time, some test cases in a constructed test suite may become redundant, because the test cases created specifically for some selected testing criteria may also satisfy other requirements, and a requirement may still satisfy by some of the proper subsets of the test suite. Two test cases are termed as duplicate or redundant if their satisfied testing objectives are same. On the other hand, some of the test cases are termed as essential if their testing objective is unique. So, the prime objective is to remove the duplicate test cases and extract the essential or diverse test cases to generate the optimal test suite.

With the aim of optimizing the test suite, many researchers proposed different regression testing techniques; such as test case selection, minimization, and prioritization techniques by using different approaches. Majority of the existing minimization tools and framework consider code coverage information of the software to be tested as a base to determine the minimized test suite [7-9] and apply any of these techniques: Greedy [10], GE (Greedy Essential) [11], GRE (Greedy Redundant Essential) [12], and HGS (Harrold Gupta and Soffa) [13]. Greedy approach repeatedly chooses the test case that covers the maximum number of uncovered test requirements. GE is based on essential concept. GRE is based on three rules: the greedy heuristic, the 1-to-1 redundancy rule, and the essentials rule. Whereas, HGS approach is based on selection of test cases according to their degree of essentialness, i.e., the order of test cases is most essential to least essential in the reduced test suite. On the other hand, Shounak et al. [20] proposed an algorithm i.e. GTAP) using TAP (Test cases which Already included in Pool-based Measure) measure and greedy search algorithm to reduce the number of test cases. Optimizing the test suite by ordering the test cases Megala et al. proposed a cost-cognizant history based prioritization approach using immune algorithm that makes use of the historical information of the test cases [22].Coverage based test suite reduction techniques have gained wide consideration but they do not always give a satisfactory output. Empirical studies, however, reveals that code coverage may not the strong criteria for test suite effectiveness [14]. To address this problem, a number of techniques and framework have been proposed to make the reduction process more effective which is based on test case classification according to similarity degree measured by a distance function [2, 15-16]. Diversity and similarity-based test case selection and prioritization are one of the new approaches with favourable output [17-19]. But the above techniques did not provide any systematic approach for minimization and prioritization both. They have optimized the test suite only on the basis of the calculated similarity degrees between test case pairs. So, the study suggests that using the greedy technique with similarity-based approach could be a better option for effective regression testing.

In this paper, we proposed a similarity based greedy approach for test suite reduction. The main idea is to analyse the similarity degree among test case pairs and systematically remove them by applying enhanced greedy algorithm while maintaining test requirements coverage. In this paper clustering approach is also considered, which could simplify the further optimization process. We divided the proposed approach into different phases, and each phase helps to get an optimal representative test set. However, rather than using single coverage criterion, here we used multiple criteria i.e. branch, control flow, def-use, and data flow to compute the similarity (distance) values for each test case pair. With the help of distance values, cluster of test cases are generated accordingly. And, on each cluster combination of regression testing techniques i.e. test case minimization with prioritization is applied (testers will decide order to execute these techniques).

In order to evaluate the quality and effectiveness of the proposed approach, we performed experiments on a standard case study. We also implemented the well-known standard HGS algorithm [17]; to compare the results of optimized test suites using our similarity-based test suite optimization algorithm with those of minimizing test suites using the HGS algorithm.

The rest of the paper is organized as follows: Section 2 provides a background of test suite optimization and an overview of related work. Section 3 presents the proposed similarity based greedy approach for test suite optimization. Section 4 presents an experimental study to validate the proposed approach, while, Section 5 discusses the result and analysis of the proposed work. Finally, the conclusion is presented in Section 6.

2. Background

Different approaches including test case selection, minimization, and prioritization aimed at finding the optimal representative test set that reduces the cost and effort required for regression testing. A complete survey of these approaches and critical investigation on different TSR (Test Suite Reduction) framework and tools are discussed in detail by Yoo and Harman [22] and Khan et al. [23].

In this paper, we presented similarity based greedy approach, which involves the combination of different regression testing techniques such as test case selection, minimization, and prioritization and employs agglomerative hierarchical clustering approach as well to produce an optimal solution. The main aim is to find an optimized representative test set by concentrating on multiple regression testing techniques rather than using the only single technique. Khan et al. combines the reduction and prioritization to get minimal and ordered test suite. However, other previous work did not consider any potential combination of regression testing techniques.

Despite the above-discussed techniques researchers are also working on similarity-based approach. The main aim of using similarity-based approach is to increase the diversity of test suite for regression testing. Different distance measures were used for this purpose. With the help of any distance measure, we can evaluate how much the two test cases are different or similar to each other. Hemmati et al. [24] analysed different similarity measures for similarity based test case selection. Cartaxo et al. [25] presented similarity based selection in the context of MBT (Model-Based Testing). Whereas, Coutinho et al. [26] proposed test suite reduction strategy based on similarity degree between test case pair. The key idea behind their approach is to remove most similar test cases according to some defined coverage requirements. Chunrong et al. [27] proposed different similarity based prioritization techniques to order the test cases according to their importance with the help of edit distances. Wang et al. [19] proposed global similarity based prioritization to overcome the problems that are not effectively solved by coverage based prioritization techniques. All the above techniques have their common objective i.e. to optimize the test suite. But, they have used only single coverage criterion to compute the similarity degree between test case pairs. And the removal of such test cases by their similarity degree based on certain single coverage criterion may suffer from the quality of the test suite generated and overall fault detection ability may also reduce. The proposed approach presented a systematic way to get the desired output by considering different coverage criteria with a similarity-based approach to minimize and prioritize the test cases.

3. The proposed approach

The proposed approach consisting of three phases: (1) Test case analysis (2) Clustering and (3) Optimization. Where, Fig. 1 shows the flow diagram of proposed similarity based greedy approach. In the following, we briefly discuss all the three phases.

3.1 Test case analysis

The required inputs for this phase are - Program source code, Test suite, and Test case coverage metrics. The test case coverage metrics are used to compare any pair of test cases in a quantifiable manner. Four metrics used in this work are Block coverage equivalence, Control-flow divergence, def-use equivalence and data flow divergence. A first metric measures the block testing overlap between two test cases of a test suite. Second metric control-flow divergence measures the similarity of two test cases that test the same blocks that have conditional path within them. Third metric DU equivalence measures def-use path testing overlaps between two test cases in a test suite. And, the last metric Data divergence measures the similarity or diversity of test cases by data values used by test cases for code variables. In this phase, all the four metrics values are calculated. After getting these values, similarity and diversity values of test case pairs are calculated. Algorithm 1 illustrates the procedure to analyse the test cases. In this phase, test cases are analysed and selected based on their similarity and diversity values. With the help of calculated similarity values, clustering approach will be applied to categorize the test cases accordingly.

3.2 Test case clustering

For clustering, we applied an agglomerative hierarchical clustering approach which is based on the distance between test case pairs [28]. Test cases are grouped into a cluster with the help of their similarity value as measured in the analysis phase. For making a group of similar and diverse test cases, threshold values of divergence and equivalence as 0.05 and 0.90 are assumed. Only those test cases having divergence value less than (0.05) and equivalence value greater than (0.90) comes under the similar or duplicate group and the remaining test cases are selected for a diverse group.

3.3 Test case optimization approach

On each cluster, a combination of optimization technique (Reduction and Prioritization) is applied to get the refined result. As discussed earlier, users are free to select the execution order of combination. If they use the combination of both the techniques, it helps them to determine the execution order of test cases as well as they get improved fault detection rate. Note that we have employed Enhanced Greedy Algorithm (CMIMX technique) [29] for test suite reduction followed by prioritizing the test cases by their total number of the unique path coverage. After applying the optimization technique, testing team will get the minimized as well as an ordered test suite, which is termed as an optimized test suite.

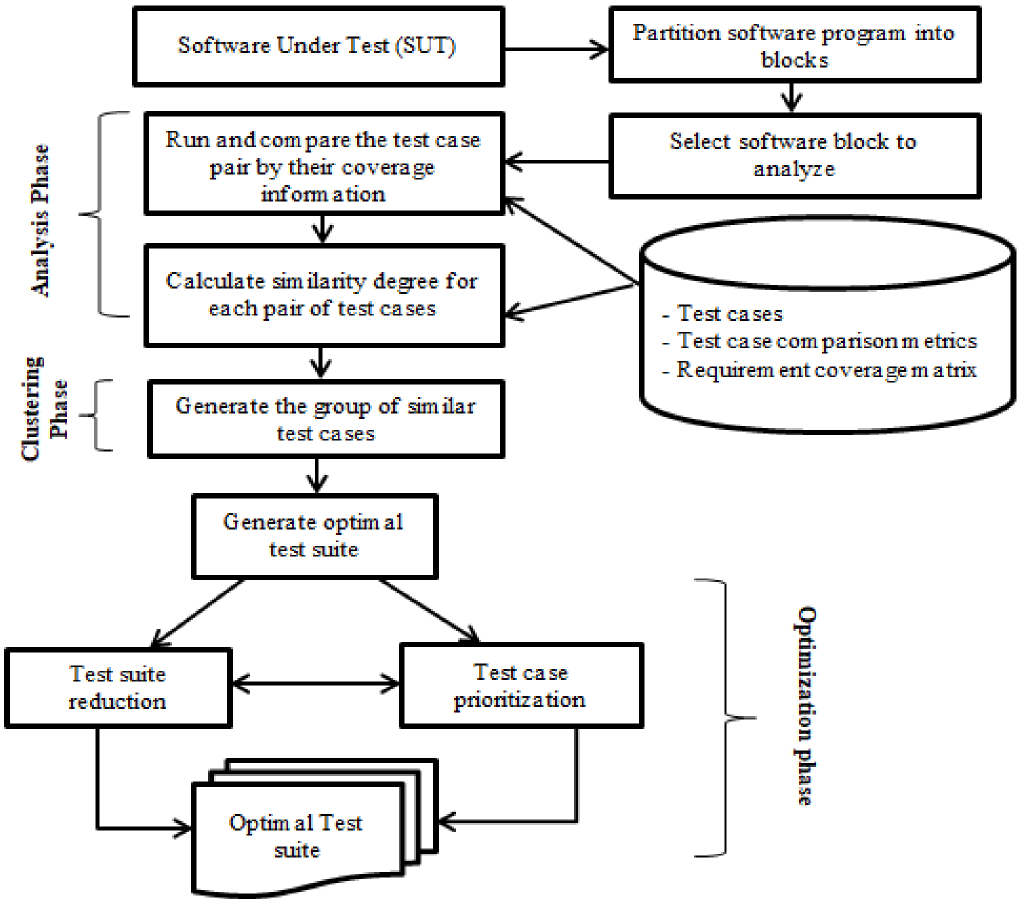
Software Under Test (SUT)
Partition software program into blocks
Select software block to analyze
Run and compare the test case pair by their coverage information
Calculate similarity degree for each pair of test cases
- Test cases
- Test case comparison metrics
- Requirement coverage matrix
Analysis Phase
Clustering Phase
Generate the group of similar test cases
Generate optimal test suite
Test suite reduction
Test case prioritization
Optimization phase
Optimal Test suite

Figure.1 Effective test suite generation process

4. Empirical study

To evaluate the effectiveness of the proposed approach, we implemented our algorithm and applied it to the benchmark program of prime number (see Table. 1). We have extracted some of the test cases for the given program to validate the proposed work (see Table 2).

The complete work of this paper is carried out in three essential phases as discussed above: where the first phase is achieved by TCCA algorithm and combination of second and third phase is achieved by SBGA algorithm. For each test case pair, the coverage metrics are calculated with the help of TCCA algorithm, where the values are shown in Table 3. Here, M_B, M_{CF}, M_{DU} and M_{DF} represents coverage metric value for block, control-flow, def-use and data flow coverage. After applying TCCA algorithm, the Commonality (Comm.) and Divergence (Div.) signature values are calculated for each pair of test cases (see Table 4).

We apply Similarity Based Greedy Algorithm (SBGA) on collected coverage metrics for each pair of test cases. According to the agglomerative clustering approach, we have created four groups i.e. diverse, relaxed, sensitive, and duplicate. To remove some conflicts, we merge them with each other and creates only two groups i.e. diverse and similar group.

Table 1. Sample program

Original Program	Fault No.	Injected Faults
#include <iostream.h>		
int main()		
int num i;		
cout << “enter a number \n”;		
cin >> num;		
if(num%2==0)	F1	if(num%2>=0)
cout<< “even\n”; else		
cout<< “odd\n”;		
if(num == 1)	F2	if(num<1)
cout<< “prime\n”; else		
{for (i=2; i ≤ num/2; ++i)	F3	{for (i=2; i ≤ num/2; --i)
if(num%i==0)	F4	if(num%i==1)
{cout<< “not prime\n”;		
goto lb;}		
cout<< “prime\n”;}		
lb; return 0; }		

Algorithm 1: Test Case Coverage Analyzer (TCCA)

```
Input: N: Total number of test cases
       $R_{criteria}$: Requirement coverage information in terms of the selected criteria i.e. blocks($R_{Br}$), Control-Flow($R_{CF}$), Def-Use($R_{DU}$), Data Flow ($R_{DF}$) coverage, and Path ($R_P$) coverage.
Output: Similarity Matrix
Begin
/* Calculate block coverage commonality */
1.  X ← Number of common blocks between each pair of test cases;
2.  Y ← Number of unique blocks tested by both test case pair;
3.  for each test case ($T_i \le N$) do
4.  for each test case ($T_{i+1} \le N$) do
5.  $Block\ Coverage\ Metrics\ M_B\ (T_i\ , T_{i+1})\ =\ X/Y$ ;
6.   end for
7.   end for
/* Control Flow (CF) is calculated for each test case that executes block B with a conditional branch. */
8.  $n(T)\ \leftarrow$ Number of times true branch is executed
9.  $n(F)\ \leftarrow$ Number of times false branch is executed
/* Calculate the total number of times each branch is executed */
10. for each test case ($T_i \le N$) do
11. $Sum(T,F)\ \leftarrow\ n(T) + n(F)$ ;
/* Calculate the average number of times each branch is executed */
12. $Avg(T,F)\ \leftarrow\ Sum(T,F)/Total(Br)$ ;
13. $CF_B(T_i)\ \leftarrow\ ((n(T) - Avg(T,F)) + (\ Avg(T,F) - n(F)))/Sum(T,F)$ ; end for
/* Variance of CF values is calculated for each test case pair that executes common blocks with conditional statements */
14. for each test case ($T_i \le N$) do
15. for each test case ($T_{i+1} \le N$) do
16. $n\ (common(T_i, T_{i+1}))$ = Number of common shared block for each test case pair;
17. $M = Mean(CF_B(T_i), CF_B(T_{i+1})$ ;
/* Calculate the variance of CF (Control Flow) value for a common block B for each test case pair */
18. $\Delta\, B(T_i, T_{i+1})\ \leftarrow\ (CF_B(T_i) - M)^2 +\ (CF_B(T_{i+1}) - M)^{\ 2}$ ;
19. $Control\ Flow\ Coverage\ Metrics\ M_{CF}\ (T_i\ , T_{i+1})\ =\ \sum \Delta\, B(T_i, T_{i+1})/n\ (common(T_i, T_{i+1})$
20.  end for
21. end for
/* Calculate def-use coverage equivalence */
22. for each test case ($T_i \le N$) do
23. for each test case ($T_{i+1} \le N$) do
24. W= No. of common def-use chain tested by test case pairs
25. Z = No. of unique def-use chain tested by both test case pairs
26. $def - use\ Coverage\ Metrics\ M_{DU}\ (T_i\ , T_{i+1})\ =\ \ W/Z$ ;
27.  end for
28.  end for
/* For each test case that executes a block B, a data flow (DF) diversity value is calculated */
29. for each test case ($T_i \le N$) do
30. $DF(T_i, B)\ \leftarrow\ \ (n(T) - n(F))/2$ ; end for
/* Calculate the variance of DF value for a common block B with loop statements */
31. for each test case ($T_i \le N$) do
32. for each test case ($T_{i+1} \le N$) do
33. $n\ (common(T_i, T_{i+1}))$ = Number of common shared block for each test case pair;
34. $M = Mean(DF_B(T_i), DF_B(T_{i+1})$ ;
35. $\Delta\, B(T_i, T_{i+1})\ \leftarrow\ (DF_B(T_i) - M)^2 +\ (DF_B(T_{i+1}) - M)^{\ 2}$ ;
36. $Data\ Flow\ Coverage\ Metrics\ M_{DF}\ (T_i\ , T_{i+1})\ \leftarrow\ \ \sum \Delta\, B(T_i, T_{i+1})/n\ (common(T_i, T_{i+1})$ ;
37. end for
38. end for
/* Generate Test Case Distance matrix to compare the pair of test cases */
39. for each test case ($T_i \le N$) do
40. for each test case ($T_{i+1} \le N$) do
41. $M_{Dist}(Commanality\ Value[T_i\ , T_{i+1}])\ \leftarrow\ \ (M_B + M_{DU})/2$ ;
```

```
42.    $M_{Dist}(Diversity\ Value[T_i, T_{i+1}]) \leftarrow (M_{CF} + M_{DF})/2$ ;
43.    **end for**
44.    **end for**
45.    **end TCCA**
```

Algorithm 2: SBGA (Similarity Based Greedy Algorithm)

```
1.     Apply agglomerative clustering;
2.     **if** $(selection == 1)$ **then**
3.     On each cluster, apply enhanced greedy algorithm (CMIMX);
4.     **for** each test cases $T_i$ **do**
5.     Calculate number of unique coverage for each test cases: $unqcov_p(T_i)$;
6.     Ranked the test cases based on highest $unqcov_p(T_i)$ to lowest; **end for**
7.     **else if** $(selection == 2)$ then
8.     Apply the step 4, step 5, step 6 and then step 3 respectively;
9.     **else** apply step 3 onwards; **endif**
10.    **endif**
11.    end SBGA
```

Table 2. Test cases

Test Cases	Test cases inputs	Expected output
T1	7	Prime
T2	2	Prime
T3	6	Not prime
T4	15	Not prime
T5	1	Prime
T6	10	Not prime

Table 3. Coverage matrix for test case pairs

Test cases		Coverage metrics values					
		T1	T2	T3	T4	T5	T6
T1	M_B	0.00	0.66	0.42	0.66	0.50	0.42
	M_{CF}	0.00	0.88	0.22	0.22	–	0.22
	M_{DU}	0.00	1.00	0.75	0.75	0.66	0.75
	M_{DF}	0.00	0.50	0.50	0.12	–	0.50
T2	M_B	0.66	0.00	0.66	0.42	0.28	0.66
	M_{CF}	0.88	0.00	2.00	2.00	–	2.00
	M_{DU}	1.00	0.00	0.75	0.75	0.66	0.75
	M_{DF}	0.50	0.00	0.00	0.12	–	0.00
T3	M_B	0.42	0.66	0.00	0.66	0.28	1.00
	M_{CF}	0.22	2.00	0.00	0	–	0
	M_{DU}	0.75	0.75	0.00	1.00	0.50	1.00
	M_{DF}	0.50	0.00	0.00	0.12	–	0.00
T4	M_B	0.66	0.42	0.66	0.00	0.50	0.42
	M_{CF}	0.22	2.00	0.00	0.00	–	0.00
	M_{DU}	0.75	0.75	1.00	0.00	0.50	1.00
	M_{DF}	0.12	0.12	0.12	0.00	–	0.12
T5	M_B	0.50	0.28	0.28	0.50	0.00	0.28
	M_{CF}	–	–	–	–	0.00	–
	M_{DU}	0.66	0.66	0.50	0.50	0.00	0.50
	M_{DF}	–	–	–	–	0.00	–
T6	M_B	0.42	0.66	1.00	0.42	0.28	0.00
	M_{CF}	0.22	2.00	0	0.00	–	0.00
	M_{DU}	0.75	0.75	1.00	1.00	0.50	0.00
	M_{DF}	0.50	0.00	0.00	0.12	–	0.00

Table 4. Overall distance matrix for test case pairs

Test cases	Distances	Signature values					
		T1	T2	T3	T4	T5	T6
T1	Div.		0.69	0.36	0.17	0.00	0.36
	Comm.		0.83	0.58	0.70	0.58	0.58
T2	Div.			1.00	1.06	0.00	1.00
	Comm.			0.70	0.58	0.47	0.70
T3	Div.				0.06	0.00	0.00
	Comm.				0.83	0.39	1.00
T4	Div.					0.00	0.06
	Comm.					0.50	0.71
T5	Div.						0.00
	Comm.						0.39

4.1 Reduction and then prioritization

On each generated cluster, we apply CMIMIX procedure to minimize the cluster size by removing obsolete or redundant test cases. Control flow graph of the source code is given in Fig. 2 and the paths covered by the test cases are shown in Table 5. One (1) and zero (0) represents paths covered and not covered respectively. With the help of Table 6 further prioritization is processed. Finally, we get the optimized (minimized and ordered) clusters C1 (T2, T1, T5) and C2 (T4, T3).

4.2 Prioritization and then reduction

In this optimization approach, first of all, we prioritize the test cases of the given cluster, and then we apply the minimization technique on them. After prioritization, in cluster one the value of PRTest becomes {T2, T1, and T5}. Subsequently, the minimization technique (CMIMX) is processed to get the representative test suite in cluster one.

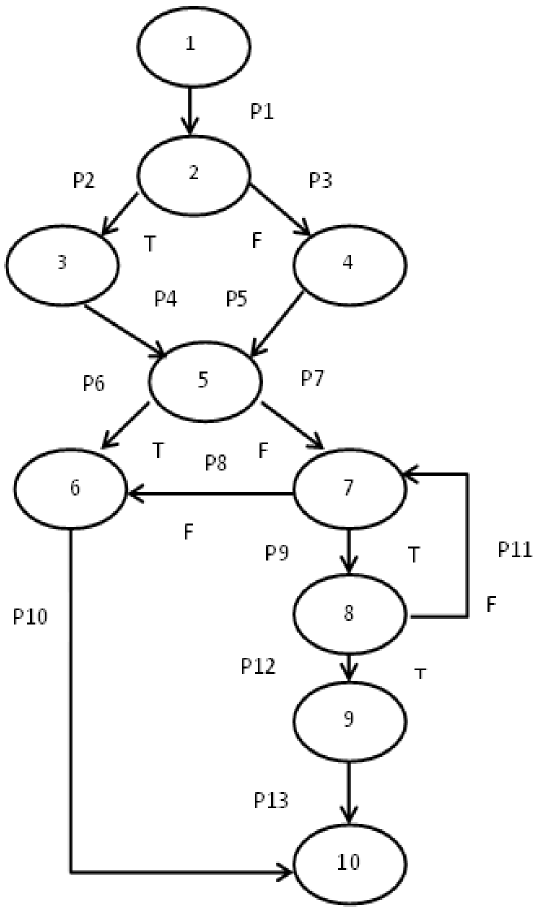

Figure.2 Control flow graph of the source program with their path coverage.

Keep in mind that, always prefer the test case have the highest priority while applying minimization here. While considering cluster two, after prioritization we get the PRTest as {T4, T3, T6}, and because the test case pair T3 and T6 are duplicate the minimized test suite or cluster becomes {T4, T3}. Then, we have combined the results of generated clusters and get the optimized test suite i.e. T2, T1, T5, T4, and T3 respectively. The result of this technique is similar to previous one. But, if we have taken a larger set of test cases, results may defer accordingly.

Table 5. Path coverage of test cases

Test Cases	**Path Coverage (P)**												
	1	2	3	4	5	6	7	8	9	10	11	12	13
T1	1	0	1	0	1	0	1	1	1	1	1	0	0
T2	1	1	0	1	0	0	1	1	0	0	0	0	0
T3	1	1	0	1	0	0	1	0	1	0	0	1	1
T4	1	0	1	0	1	0	1	0	1	0	1	1	1
T5	1	0	1	0	1	1	0	0	0	1	0	0	0
T6	1	0	1	0	1	0	1	1	1	1	1	0	0

Table 6. Unique paths covered by test cases of cluster C1

Test Cases	**Path Covered (cov(t))**	**│cov(t)│**	**│unqcov(t)│**
T1	1, 3, 5, 7, 8, 9, 10, 11	8	2 {9, 11}
T2	1, 2, 4, 7, 8, 12, 13	7	4 {2, 4, 12, 13}
T5	1, 3, 5, 6, 10	5	1 {6}

Table 7. Coverage criteria

Coverage Criteria	**Description**
def-use Coverage	Measures a logic execution sequence in a block that defines and uses a variable [2]
Block Coverage	Measures the sequence of consecutive groups of statements [2]
Control Flow Coverage	Measures the same block that has a conditional path between them [15]
Data Flow Coverage	Measures the coverage with respect to data values used for code variables [15]
Path Coverage	Measure the coverage of each possible routes in each function [16]

5. Experimental results and discussion

The performance evaluation of the proposed algorithms and state-of-the-art algorithm HGS [13] has been performed on a benchmark program with different coverage criteria (Table 7). The four coverage criteria i.e. Block, Control flow, def-use and data flow are used to assess the level of similarity between test case pair. Alternatively, path coverage is used for optimizing test suite. Hand seeded faults were used for the subject program. The representative optimized test set obtained for the subject program is analyzed for test suite size and fault detection effectiveness. We have compared the proposed approach with the standard minimization technique i.e. HGS approach. For the HGS algorithm, we have taken each coverage criteria independently for measuring their adequacy. Moreover, for the proposed algorithm we have taken multi-coverage criteria. The results of this experiment are presented via Fig. 4, which denotes the percentage of suite size reduction (SSR) and percentage fault detection loss (FDL) of minimized test suites.

To show the importance of combining prioritization with minimization, we have compared the proposed prioritization approach with random prioritization that are presented via Fig. 4 and Fig. 5.

5.1 Performance metrics

The following metrics are used for performance evaluation of the proposed and state-of-the-art algorithms and are described as follows:
The first test metric implies the percentage reduction in test suite i.e. Suite Size Reduction (SSR). This is calculated as in Eq. (1)

$$SSR = 100 \times \left(1 - \frac{T_{rs}}{T}\right) \quad (1)$$

On the other hand, second metric Fault detection loss (FDL) percentage signifies the total number of faults revealed by the minimized test suite. It is calculated as in Eq. (2)

$$FDL = 100 \times \left(1 - \frac{F_m}{F}\right) \quad (2)$$

Where F represents the total number of distinct faults revealed by original test suite and F_m is the number of distinct faults exposed by minimized test suite.

5.2 Result and analysis

As can be seen in Fig. 3, for each coverage criteria the percentage of size reduction and fault detection loss are different for HGS algorithm. The main drawback of this algorithm is that they achieve high test suite size reduction with the sacrifice of significant loss in fault detection ability. However, the prime objective of any testing scheme is to reveal maximum faults. We observe from Fig. 3 that suite size reduction with multiple coverage criteria by the proposed algorithm is lower than the HGS algorithm as expected. However, the proposed algorithm achieves zero fault dete ction loss since the reduced test suite retain all coverage (multi-coverage). Where the loss is very less than the corresponding HGS fault detection loss values. In the proposed approach we have mainly concentrated on fault detection ability of the reduced test suite.

In this work, the test cases are checked for all the selected coverage criterion for redundancy rather than considering only single criterion as HGS algorithm. Because while removing any duplicate test cases from the test suite, there might be a possibility that the removed test cases may not become duplicate with respect to another criterion. The user can also use a different set of coverage criteria to evaluate similarity or difference between test case pair depending on their requirement. Fig. 4 and Fig. 5 illustrates the fault coverage percentage of ordered test cases based on random prioritization and proposed prioritization approach. And result reveals that our proposed approach is quite effective in terms of early fault detection as compared to random based approach. As we can clearly see in Fig. 4 that 100% fault coverage is obtained at last while execution of test case T5. Despite that in Fig. 5, 100% coverage is obtained at an early stage while executing the test cases on a priority basis. By comparing the proposed approach with the conventional approach (HGS), our approach identifies maximum faults as early as possible and show good improvement in fault detection loss (see Fig. 3).

Therefore analysis reveals that, identification of duplicate test cases based on any one single criteria and throwing them away is not an smart approach. Our experimental results clearly reveals that use of more than one criteria improves the quality of reduced and prioritized test suite in terms of coverage and fault detection ability. Our approach also provides the combinatorial approach for regression testing techniques i.e. the possible combination of minimization and prioritization. Overall the proposed similarity based greedy algorithm performs better than the existing coverage based technique HGS. If we have taken larger data set for validation of the proposed work, results would be quite different in terms of suite size reduction and fault detection loss.

6. Conclusion

In this paper, a similarity-based greedy approach is presented to get an optimal test suite. The optimized test suite is used by testers for effective regression testing. The goal is to apply similarity-based strategy with multiple criteria to identify the difference between a pair of test cases and compare them to similarity level. This paper concentrates on the combination of regression testing techniques i.e. minimization and prioritization with different coverage criterion to optimize the test suite size. The main idea is to analyze the test cases first to know the difference and similarity value of the test case pairs and further apply the greedy and clustering approach to optimize the test cases accordingly. We have proposed two algorithms TCCA and SBGA for optimizing the test suite. The proposed approach can be very helpful when the fault detection effectiveness is more important as compared to code coverage. The paper also presents the results of an experimental study conducted on the small case study to evaluate the performance of the proposed approach. To evaluate the effectiveness of the

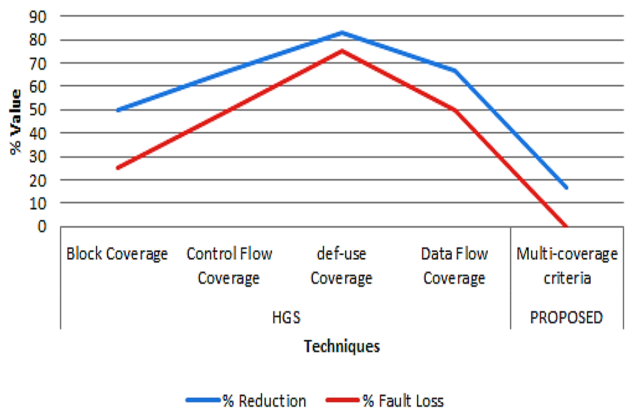

Figure.3 Test suite size reduction and fault detection loss using HGS and proposed algorithm with different coverage criteria

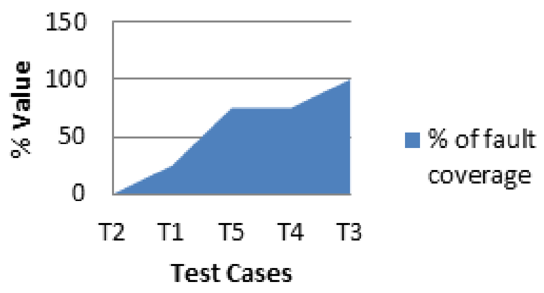

Figure.4 Fault coverage graph for random based prioritization of test cases

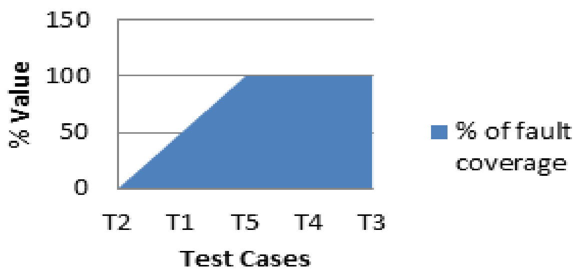

Figure.5 Fault coverage graph for the proposed prioritization approach

proposed work two performance metrics were used i.e. SSR and FDL. And the experimental results show that the fault detection ability is highly improved by the proposed technique as compared to existing technique.

As future work, we are going to experiment with the larger size of data set that may produce different and more precise results. Moreover, experimental evaluation of other similarity functions with a different combination of coverage criteria needs to be conducted for getting more optimal test cases.

References

[1] S.U.R. Khan, A. Nadeem, and A. Awais, "TestFilter: A Statement Coverage based Test Case Reduction Technique", In: *Proc. of 10th IEEE International Multitopic Conference (INMIC'06)*, pp. 275-280, 2006.

[2] S. Singh, C. Sharma, and U. Singh, "A Simple Technique to Find Diverse Test Cases", In: *Proc. of 9th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security, and Robustness*, pp. 1-4, 2013.

[3] R. Gupta and M. L. Soffa, "Employing static information in the generation of test case", *Software Testing, Verification and Reliability* Vol.3, No.1, pp. 29-48, 1993.

[4] D. Binkley, "Semantics guided regression test cost reduction", *IEEE Transactions on Software Eng*ineering, Vol.23, No.8, pp.498–516, 1997.

[5] H. Zhong, L. Zhang, and H. Mei, "An experimental study of four typical test suite reduction techniques", *Information and Software Technolo*gy, Vol.50, No. 6, pp. 534–546, 2008.

[6] A. Ilkhani and G. Abaee, "Extracting test cases by using data mining; reducing the cost of testing", In: *Proc. of International Conference on Computer Information Systems and Industrial Management Applications*, pp.730–737, 2011.

[7] 8. J. R. Horgan and S. London, "A data flow coverage testing tool for C", In: *Proc. of the Second IEEE Symposium on Assessment of Quality Software Development Tools*, pp. 2-10, 1992.

[8] H. S. Chae, G. Woo, T. Y. Kim, J. H. Bae, and W. Y. Kim, "An automated approach to reducing test suites for testing retargeted C compilers for embedded systems", *Journal of Systems and Software*, Vol.84, No.12, pp. 2053-2064, 2011.

[9] X. Zhang, Q. Gu, X. Chen, J. QI, and D. Chen, "A study of relative redundancy in test-suite reduction while retaining or improving fault-localization effectiveness", In: *Proc. of the ACM Symposium on Applied Computing (SAC'10)*, Sierre, Switzerland, pp. 2229-2236, 2010.

[10] V. Chvatal, "A greedy heuristic for the set-covering problem", *Mathematics of operations research*, Vol.4, No.3, pp.233-235, 1979.

[11] T. Y. Chen and M. F. Lau, "A simulation study on some heuristics for test suite reduction", *Information and Software Technology*, Vol.40, No.13, pp. 777-787, 1998.

[12] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction", *Information and Software Technology*, Vol.40, No.(5-6), pp.347-354, 1998.

[13] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol.2, No.3, pp. 270-285, 1993.

[14] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness", In: *Proc. of the 36th International Conference on Software Engineering*, pp. 435-445, 2014.

[15] C. Sharma and S. Singh, "Mechanism for identification of duplicate test cases", In: *Proc. of Recent Advances and Innovations in Engineering (ICRAIE)*, pp. 1-5, 2014.

[16] S. Singh and R. Shree, "A combined approach to optimize the test suite size in regression testing", *CSI transactions on ICT*, Vol.4, No.(2-4), pp. 73-8, 2016.

[17] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. L. Machado, "Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing", *Software Quality Journal,*Vol.24, No.2, pp.407-445, 2016.

[18] H. Hemmati, A. Arcuri, and L. C. Briand, "Achieving scalable model-based testing through test case diversity", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol.22, No.1, pp. 1–42, 2013.

[19] R. Wang, S. Jiang, and D. Chen, "Similarity-based regression test case prioritization", In: *Proc. of the 27th International Conference on Software Engineering and Knowledge Engineering*, pp. 358-363, 2015.

[20] S. R. Sugave, S. H. Patil, and B. E. Reddy, "A Cost-Aware Test Suite Minimization Approach Using TAP Measure and Greedy Search Algorithm", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.4, pp.60-69, 2017.

[21] M. Tulasiraman and V. Kalimuthu, "Cost Cognizant History Based Prioritization of Test Case for Regression Testing Using Immune Algorithm", *International Journal of Intelligent Engineering and Systems*, Vol.11, No.1, pp.221-228, 2018.

[22] S. Yoo and M. Harman, "Regression testing minimization, selection, and prioritization: a survey", *Softw. Test., Verif. Reliable*, Vol.22, No.2, pp. 67–120, 2012.

[23] S. U. R. Khan, S. P. Lee, R. W. Ahmad, A. Akhunzada, and V. Chang, "A survey on Test Suite Reduction frameworks and tools", *International Journal of Information Management*, Vol.36, No.6, pp.963-975, 2016.

[24] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection", In: *Proc. of IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 141-150, 2010.

[25] E. G. Cartaxo, P. D. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing", *Software Testing, Verification and Reliability*, Vol.21, No.2, pp.75-100, 2011.

[26] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. L. Machado, "Test suite reduction based on similarity of test cases", In: *Proc. of 7th Brazilian workshop on systematic and automated software testing—CBSoft,* 2013.

[27] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities", *Software Quality Journal*, Vol.22, No.2, pp.335-361, 2014.

[28] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritization incorporating expert knowledge", In: *Proc. of the 18th international symposium on Software testing and analysis*, pp. 201-212. ACM, 2009

[29] A. P. Mathur, "Foundations of Software Testing", Pearson Education, 1st Edition, 2008.