An Efficient Technique to Test Suite Minimization using Hierarchical Clustering Approach

Fayaz Ahmad Khan, Anil Kumar Gupta, Dibya Jyoti Bora

Abstract:- Software testing is a pervasive activity in software development. Testing is widely used to reveal bugs in real software development and is also an expensive task. Testing is expensive due to the fact that it takes very long time to execute the whole test suite. The initial test suite is very large in size and has redundant test cases. So it is necessary to apply some selective techniques in order to reduce the large size of the initial test suite to a manageable size and make it feasible for practical execution. In this study, Hierarchical clustering approach is presented and implemented on the initially generated test suite in order to reduce its size and partition it, into a fixed number of clusters. Here, a branch coverage criterion is selected as the code coverage criteria and for the determination of the number of clusters.

Keywords: Software Testing, Test suite Minimization, Data Clustering, Hierarchical Clustering.

I. INTRODUCTION

To reveal bugs, software testing and retesting occurs continuously during and after software development. Typically, a test suite should be prepared before initial testing starts. As software evolves and grows new test cases are added to the test suite. Over time, some test cases in the test suite become redundant as the requirements covered by them are also covered by the other test cases. To reduce the cost, time and effort for testing, it is very important to develop some techniques that will help in keeping the size of the test suite to a manageable number and will eliminate the redundant test cases from it. The reduction or minimization can be achieved at the time generating the test cases or at the time after acquiring an initial test suite. In literature [1] [2] many efforts have been proposed to address the issue of redundancy in the test suites by taking only one testing criterion. Various types of test criterion exists like: statement coverage criteria, decision coverage, branch coverage [2] [3], path coverage and requirement coverage criteria [1] [4]. A tester can set any of the aforesaid criteria as testing criteria to ensure the complete testing of the application. Software testing plays a very important role in assuring the quality and reliability of the software under test. As the size and complexity of the under developed product grows, the time and effort required for effective testing also increases. Literature on software testing indicates that more than 50% of the cost of software development is devoted to testing [5]. One of the important issues in software testing is the test case generation. Test case design and generation are both time consuming and labour intensive tasks. There are usually two approaches followed while designing test cases; One by manually and second by using automation techniques. The manual way of designing is very time consuming and error prone. So to save time, automated techniques are used for test case generation. But due to automation in the test case generation, large and redundant test cases are generated which take longer time for execution. Thus, it is very important to develop techniques that will help in tackling the aforesaid problem in order to reduce the time and cost devoted in testing.

Revised Version Manuscript Received on September 02, 2015.

Fayaz Ahmad Khan, Department of Computer Science and Applications, Barkatullah University Bhopal, (M.P) India.
Anil Kumar Gupta, Department of Computer Science and Applications, Barkatullah University Bhopal, (M.P) India.
Dibya Jyoti Bora, Department of Computer Science and Applications, Barkatullah University Bhopal, (M.P) India.

II. TEST SUITE MINIMIZATION

Test objectives or requirements are usually defined before the software is tested are very different from each other and may also vary in granularity. A test case requirement can be defined with a small granularity like the coverage of the every statement and on the other hand, it can be defined with large granularity, such as the coverage of every user requirement. As one test case is not sufficient to satisfy all the user requirements, it usually requires large number of test cases to satisfy as many as possible test requirements. Thus in practice, the test suite undergoes the process of expansion due to the addition of new test cases as and when the software is modified. The other reason behind the large size of test suite is that the input domain of program variables is very large and exhaustive. So, exhaustive testing with an initial test suite is not adequate for practical execution. Thus, it has long been identified as a research problem to find a minimized subset of test cases from the test suite for an effective testing. This problem is usually referred to as test suite minimization problem.

In literature various techniques have been proposed in the area of test suite minimization. Simran et al, in [6], proposed a delayed greedy algorithm using concept analysis. Reduction in test suite is also minimized by reducing the requirement set using graph retraction techniques [7]. In [1], call tree construction approach is proposed to address the test suite minimization for white box testing. But the construction of call tree is very cumbersome process. Also in [8], an approach based for embedded nondeterministic systems based on testing in context is proposed. Mutation analysis proposed in [9], is an important technique for test suite minimization. But generation of mutants is also a very difficult task. The other useful techniques each with pros and cons are genetic algorithm based [10] and Integer linear programming based [4].

In this study, we have presented and implemented a very useful technique for test suite minimization using a Hierarchical clustering approach. The proposed technique partitions the test suite into a fixed number of clusters or partitions based on the branch coverage criteria. A coverage criterion is an important constraint that is used to stop testing. It guides the tester during the testing process.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

III. DATA CLUSTERING

The aim of cluster analysis is to partition the given set of data or objects into clusters (subsets, groups or classes). Each obtained cluster or partition should have the following two characteristics:

a). Homogeneity within the cluster that is data that belong to the same cluster should be as similar as possible.

b). Heterogeneity between clusters that is data belong to different clusters should be as different as possible.

Data clustering or cluster analysis is one of the fundamental data mining techniques and has its wide applications to customer segmentation, data summarization and target marketing [11]. There are two approaches followed for cluster analysis [12]: (1) Hard clustering and (2) Soft clustering. Hard clustering partitions the data or objects into fixed number of clusters and each object in a cluster cannot share the properties with objects in other clusters. While in case of soft clustering, every object is assigned a membership value calculated using fuzzy logic. The membership values are used for cluster assignment and indicate that the objects may belong to more than one cluster. Soft clustering approach in some situations may give more promising results compared to hard cluster approach. We have selected hard clustering approach in our study because soft clustering approach is computationally very hard. In hard clustering, two commonly used techniques for cluster analysis are:

(i). Partitional clustering technique.

(ii) Hierarchical clustering techniques.

In Partitional clustering, the initial data set is treated as a single cluster and gradually the data set is divided into a fixed number of clusters. In case of hierarchical clustering, the data set is decomposed into a hierarchy of groups and the result is displayed by a tree like structure known as dendrogram, whose root node represents the whole data set and each node is the single object of the data set. In our previous study [13] we have used Partitional clustering technique for test suite minimization and in this study we are applying hierarchical agglomerative clustering approach. In agglomerative hierarchical clustering the clusters are formed from branches to the root as depicted in the Fig.1. Hierarchical clustering algorithms usually use proximity matrix or similarity matrix for the cluster assignment and merging. There are number of cluster proximities used in hierarchical clustering approach. The few are single link, complete link, and group average and wards method.

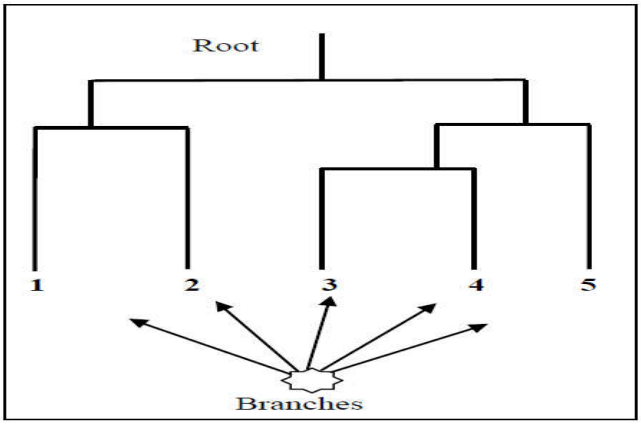

Fig.1

In [14] [15], the basic steps used in agglomerative clustering approach are:

[1] Compute the proximity graph or matrix.

[2] Merge the most similar clusters.

[3] Update the proximity matrix by measuring the proximity between newly formed or merged cluster with all the remaining clusters.

[4] Repeat steps [2] and [3], until only one cluster remains.

The following are the proximity or distance measures used for merging the clusters or objects;

I. Single Link or MIN

The distance between two clusters C1 and C2 is the minimum n1 n2 distances between ant two points in the different clusters. Single link is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

$\mathbf{d}(C1,C2) = \text{Min}(\mathbf{d}_{rs})$ with r Є C1 and s Є C2.

II. Complete Link or MAX

The distance between two groups is defined as the maximum of n1 n2 distances between each observation of group C1 and group C2.

$\mathbf{d}(C1,C2) = \text{Min}(\mathbf{d}_{rs})$ with r Є C1 and s Є C2.

III. Average Link or Group Average

In average linkage, the distance is defined as the arithmetic average of n1 n2 distances between the observations in both C1 and C2 groups.

$$\mathbf{d}(C1,C2) = 1/n1n2 * \sum_{r=1}^{n1} * \sum_{s=1}^{n2} \mathbf{d}_{rs}$$

Where r Є C1 and s Є C2

Average link cluster proximity is an intermediate approach between MIN and MAX.

IV. IMPLEMENTATION OF PROPOSED APPROACH

Agglomerative hierarchical clustering technique with average linkage proximity measure is implemented on the test suite of a sample program. The sample program accepts three input positive integers that represent the coefficients of quadratic equation. The sample code or module that is to be tested is shown in **Fig.2**. We have used Worst test technique for the generation of the test cases. In addition to specific range of positive integers we have randomly added some additional test cases to see the behavior of the sample code. Total of 99 test cases are present in the initial test suite which is presented in Fig.4. It is clearly observed from this test suite that, some test cases are redundant as they are satisfying the same requirements multiple times. The size and redundancy of a test suite are two important problems in testing and many proposed technique exist that in some way prove beneficial in certain circumstances. But there is no concrete solution to these problems, because finding the optimal representative set from the test suite has been identified as an NP-Complete problem [16].

In this study, branch coverage criterion is selected as the testing criteria. For 100% branch coverage of the sample code, only four test cases are required. The control flow graph of the sample code presented in **Fig.3** is used for the

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

determination of number of branches. Also in our sample module, four test requirements exist that are:

1. If (d>0): Real Roots
2. If (d=0): Equal Roots
3. If (d<0): Imaginary Roots
4. If (Not Valid Input): Invalid Values

Thus for an effective testing of the sample code, all the test requirements and branches must be satisfied and covered by the test cases. Based on the number of test requirements and branches, we have partitioned the test suite into four (4) clusters as depicted in **Fig.4**. Hence with our proposed approach we have reduced both the size and redundancy of the test suite. The initial size of the test suite was 99 and now it has been reduced to only 4, that means now only four test cases are required for 100% requirement and branch coverage of the desired sample code. The redundancy of the test suite is addressed by grouping the multiple test cases in their appropriate cluster based on the requirements covered by them. Also, a single test case that is selected from each cluster will represent the other test cases in the same cluster. The cluster analysis is performed in Weka, which is an important tool used for data mining. The resulted partitioned test suite is pictured in **Fig. 5**. The final cluster assignment of test cases is pictured in **Fig.7** The two dimensional scatter plot matrix given in **Fig.8**, is the visual representation of the manipulated data set for selection and analysis. The matrix of plots also represents the different attributes within the data set plotted against the other attributes. The graphical representation of the proposed technique is pictured in **Fig.6**

```
import java.util.*;

class Qaud
{
 public static void main(String[] args)
{
  int a,b,c;
 float d;
 Scanner s=new Scanner(System.in);

 System.out.println("Enter input");

  a=s.nextInt();
  b=s.nextInt();
  c=s.nextInt();

 if(((a>=0)&&(a<=100))&&
((b>=0)&&(b<=100))&&((c>=0)&&(c
<=100)))
  {
  d = (b*b)-4(a*c);
  If (d>0)
    {
     System.out.println("Real roots");
    }

 Else If (d==0)
 {
  System.out.println("Equal roots");
 }

 Else if (d<0)
    {
 System.out.println("Imaginary roots");
    }
}

 Else
 {
 System.out.println("NotValid input");
 }
}
}
```

Fig.2 Sample Module

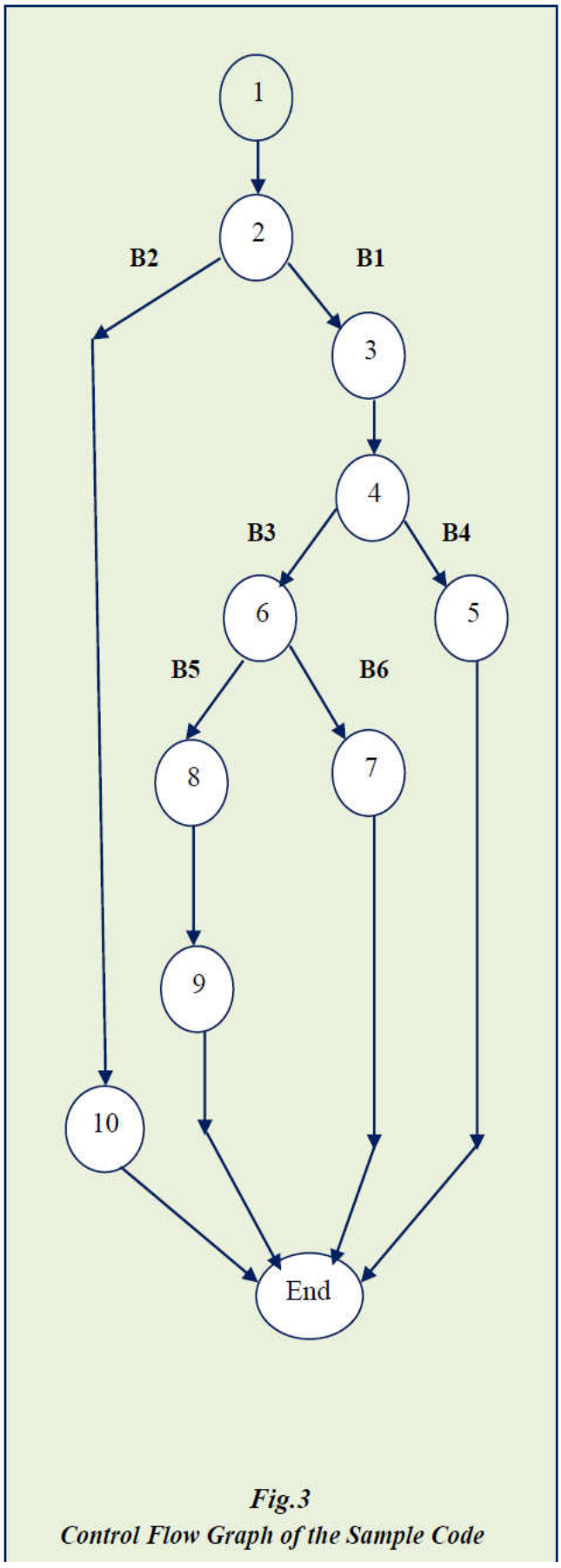

Fig.3
Control Flow Graph of the Sample Code

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

Microsoft Excel - FinaltestCases Quadratic

TEST CASE ID	A	B	C	Expected Result
T1	100	0	1	Imaginary
T2	100	0	50	Imaginary
T3	100	0	99	Imaginary
T4	100	0	100	Imaginary
T5	100	1	0	Real Roots
T6	100	1	1	Imaginary
T7	100	1	50	Imaginary
T8	100	1	99	Imaginary
T9	100	1	100	Imaginary
T10	100	50	0	Real Roots
T11	100	50	1	Real Roots
T12	100	50	50	Imaginary
T13	100	50	99	Imaginary
T14	100	50	100	Imaginary
T15	100	99	0	Real Roots
T16	100	99	1	Real Roots
T17	100	99	50	Imaginary
T18	100	99	99	Imaginary
T19	100	99	100	Imaginary
T20	100	100	0	Real Roots
T21	100	100	1	Real Roots
T22	100	100	50	Imaginary
T23	100	100	99	Imaginary
T24	100	100	100	Imaginary
T25	99	0	0	EqualRoots
T26	99	0	1	Imaginary
T27	99	0	50	Imaginary
T28	99	0	99	Imaginary
T29	99	0	100	Imaginary
T30	99	1	0	Real Roots
T31	99	1	1	Imaginary
T32	99	1	50	Imaginary
T33	99	1	99	Imaginary
T34	99	1	100	Imaginary
T35	99	50	0	Real Roots
T36	99	50	1	Real Roots
T37	99	50	50	Imaginary
T38	99	50	99	Imaginary
T39	99	50	100	Imaginary
T40	99	99	0	Real Roots
T41	99	99	1	Real Roots
T42	99	99	50	Imaginary
T43	99	99	99	Imaginary
T44	99	99	100	Imaginary
T45	99	100	0	Real Roots
T46	99	100	50	Real Roots
T47	99	100	99	Imaginary
T48	99	100	100	Imaginary
T49	50	0	0	Imaginary
T50	50	0	1	EqualRoots
T51	50	0	50	Imaginary
T52	50	0	99	Imaginary
T53	50	0	100	Imaginary
T54	50	1	0	Imaginary
T55	50	1	1	Real Roots
T56	50	1	50	Imaginary
T57	50	1	99	Imaginary
T58	50	1	100	Imaginary
T59	50	50	0	Imaginary
T60	50	50	1	Real Roots
T61	50	50	50	Real Roots
T62	50	50	99	Imaginary
T63	50	50	100	Imaginary
T64	50	99	0	Imaginary
T65	50	99	1	Real Roots
T66	50	99	50	Real Roots
T67	50	99	99	Imaginary
T68	50	99	100	Imaginary
T69	50	100	0	Imaginary
T70	50	100	1	Real Roots
T71	50	100	50	Real Roots
T72	50	100	99	EqualRoots
T73	50	100	100	Imaginary
T74	1	0	0	Imaginary
T75	1	0	1	Imaginary
T76	1	0	50	Imaginary
T77	1	0	99	Imaginary
T78	1	0	100	Imaginary
T79	1	1	0	Imaginary
T80	1	1	1	Imaginary
T81	1	1	50	Imaginary
T82	1	1	99	Imaginary
T83	1	1	100	Imaginary
T84	1	50	0	Imaginary
T85	1	50	1	Real Roots
T86	1	50	50	Real Roots
T87	1	50	99	Real Roots
T88	1	50	100	Real Roots
T89	1	99	0	Real Roots
T90	1	99	1	Real Roots
T91	1	99	50	Real Roots
T92	1	99	99	Real Roots
T93	1	99	100	Real Roots
T94	1	100	0	Real Roots
T95	1	100	1	Real Roots
T96	1	100	50	Real Roots
T97	1	100	99	Real Roots
T98	dd	xx	ss	not Valid
T99	110	110	200	not Valid

Fig.4 The initial test suite

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

TEST CASE ID	INPUT DOMAIN	EXPECTED RESULTS	CLUSTER NUMBER	
T1	100.0,0.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T2	100.0,0.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T3	100.0,0.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T4	100.0,0.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T6	100.0,1.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T7	100.0,1.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T8	100.0,1.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T9	100.0,1.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T12	100.0,50.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T13	100.0,50.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T14	100.0,50.0,100.0	IMAGINARY ROOTS	CLUSTER 1	C1
T17	100.0,99.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T18	100.0,99.0,99.0	IMAGINARY ROOTS	CLUSTER 1	A single test case from this cluster will represent the whole cluster and other test cases in the same cluster.
T19	100.0,99.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T22	100.0,100.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T23	100.0,100.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T26	100.0,100.0,100	IMAGINARY ROOTS	CLUSTER 1	
T27	99.0,0.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T28	99.0,0.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T29	99.0,0.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T31	99.0,0.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T32	99.0,1.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T33	99.0,1.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T34	99.0,1.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T37	99.0,1.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T38	99.0,50.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T39	99.0,50.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T42	99.0,50.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T43	99.0,99.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T44	99.0,99.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T47	99.0,99.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T48	99.0,100.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T49	99.0,100.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T51	50.0,0.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T52	50.0,0.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T53	50.0,0.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T54	50.0,0.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T55	50.0,1.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T56	50.0,1.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T57	50.0,1.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T58	50.0,1.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T59	50.0,50.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T62	50.0,50.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T63	50.0,50.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T64	50.0,99.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T67	50.0,99.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T68	50.0,99.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T69	50.0,99.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T70	50.0,100.0,0.0'	IMAGINARY ROOTS	CLUSTER 1	
T73	50.0,100.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T74	1.0,0.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T75	1.0,0.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T76	1.0,0.0,50.0	IMAGINARY ROOTS	CLUSTER 1	
T77	1.0,0.0,99.0	IMAGINARY ROOTS	CLUSTER 1	

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

T78	1.0,0.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T81	1.0,1.0,0.0	IMAGINARY ROOTS	CLUSTER 1	
T82	1.0,1.0,1.0	IMAGINARY ROOTS	CLUSTER 1	
T83	1.0,1.0,99.0	IMAGINARY ROOTS	CLUSTER 1	
T84	1.0,1.0,100.0	IMAGINARY ROOTS	CLUSTER 1	
T5	100.0,1.0,0.0	REAL ROOTS	CLUSTER 2	
T10	100.0,50.0,0.0	REAL ROOTS	CLUSTER 2	
T11	100.0,50.0,1.0	REAL ROOTS	CLUSTER 2	
T15	100.0,99.0,0.0	REAL ROOTS	CLUSTER 2	
T16	100.0,99.0,1.0	REAL ROOTS	CLUSTER 2	
T20	100.0,100.0,0.0	REAL ROOTS	CLUSTER 2	
T21	100.0,100.0,1.0	REAL ROOTS	CLUSTER 2	
T30	99.0,1.0,0.0	REAL ROOTS	CLUSTER 2	
T35	99.0,50.0,0.0	REAL ROOTS	CLUSTER 2	
T36	99.0,50.0,1.0	REAL ROOTS	CLUSTER 2	
T40	99.0,99.0,0.0	REAL ROOTS	CLUSTER 2	
T41	99.0,99.0,1.0	REAL ROOTS	CLUSTER 2	C2
T45	99.0,100.0,0.0	REAL ROOTS	CLUSTER 2	Similarly A
T46	99.0,100.0,50.0	REAL ROOTS	CLUSTER 2	single test
T55	50.0,1.0,1.0	REAL ROOTS	CLUSTER 2	case from this
T60	50.0,50.0,1.0	REAL ROOTS	CLUSTER 2	cluster.
T61	50.0,50.0,50.0	REAL ROOTS	CLUSTER 2	
T65	50.0,99.0,1.0	REAL ROOTS	CLUSTER 2	
T66	50.0,99.0,50.0	REAL ROOTS	CLUSTER 2	
T70	50.0,100.0,1.0	REAL ROOTS	CLUSTER 2	
T71	50.0,100.0,50.0	REAL ROOTS	CLUSTER 2	
T85	1.0,50.0,1.0	REAL ROOTS	CLUSTER 2	
T86	1.0,50.0,50.0	REAL ROOTS	CLUSTER 2	
T87	1.0,50.0,99.0	REAL ROOTS	CLUSTER 2	
T88	1.0,50.0,100.0	REAL ROOTS	CLUSTER 2	
T89	1.0,99.0,0.0	REAL ROOTS	CLUSTER 2	
T90	1.0,99.0,1.0	REAL ROOTS	CLUSTER 2	
T91	1.0,99.0,50.0	REAL ROOTS	CLUSTER 2	
T92	1.0,99.0,99.0	REAL ROOTS	CLUSTER 2	
T93	1.0,99.0,100.0	REAL ROOTS	CLUSTER 2	
T94	1.0,100.0,0.0	REAL ROOTS	CLUSTER 2	
T95	1.0,100.0,1.0	REAL ROOTS	CLUSTER 2	
T96	1.0,100.0,50.0	REAL ROOTS	CLUSTER 2	
T97	1.0,100.0,99.0	REAL ROOTS	CLUSTER 2	
T24	99.0,0.0,0.0	EQUAL ROOTS	CLUSTER 3	C3
T49	50.0,0.0,1.0	EQUAL ROOTS	CLUSTER 3	One test case
T71	50.0,100.0,99.0	EQUAL ROOTS	CLUSTER 3	
T98	dd, xx, ss	NOT VALID INPUTS	CLUSTER 4	C4
T99	110.0,110.0,200	NOT VALID INPUTS	CLUSTER 4	One test case

Fig.5 The resulted partitioned test suite.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

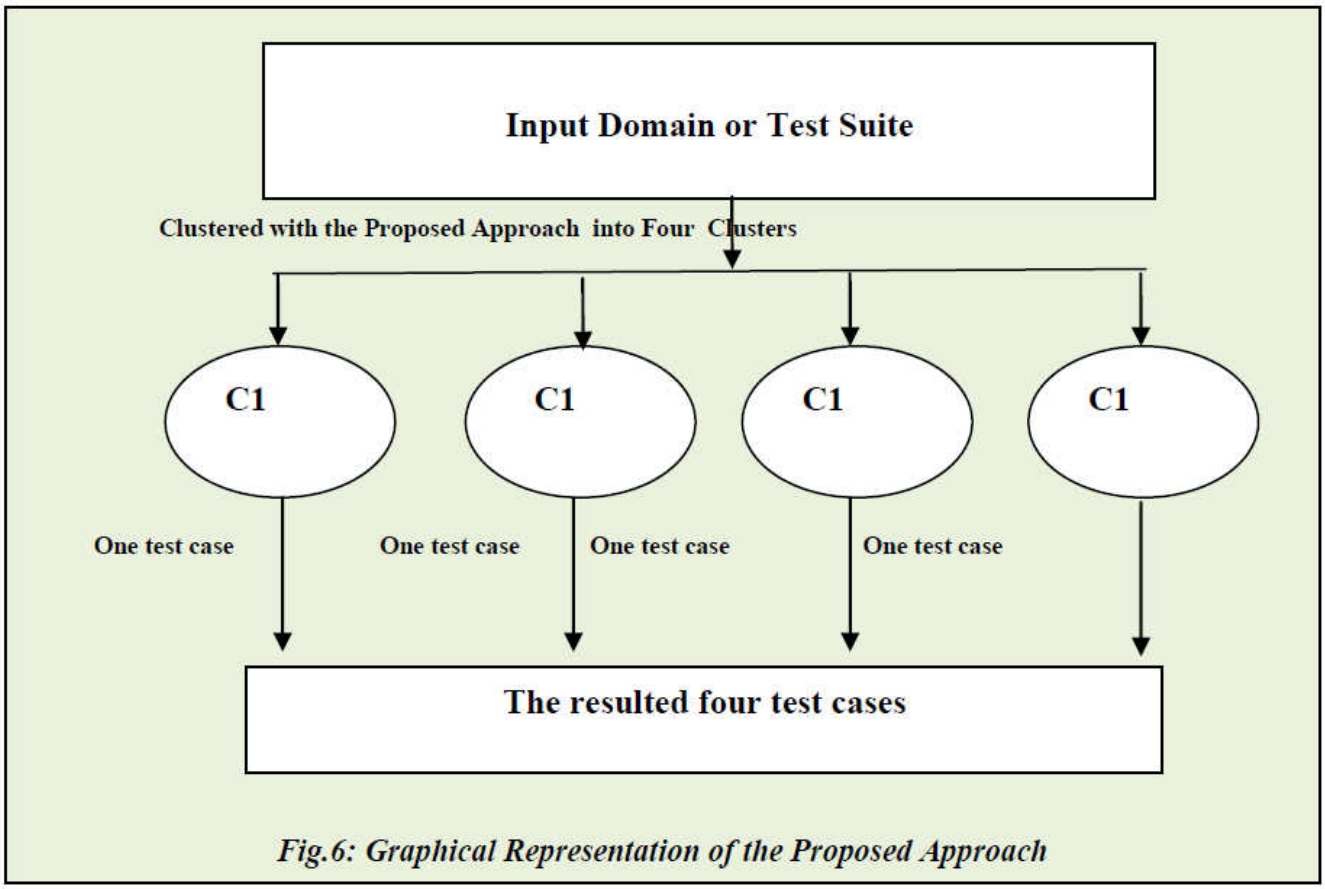

Fig.6: Graphical Representation of the Proposed Approach

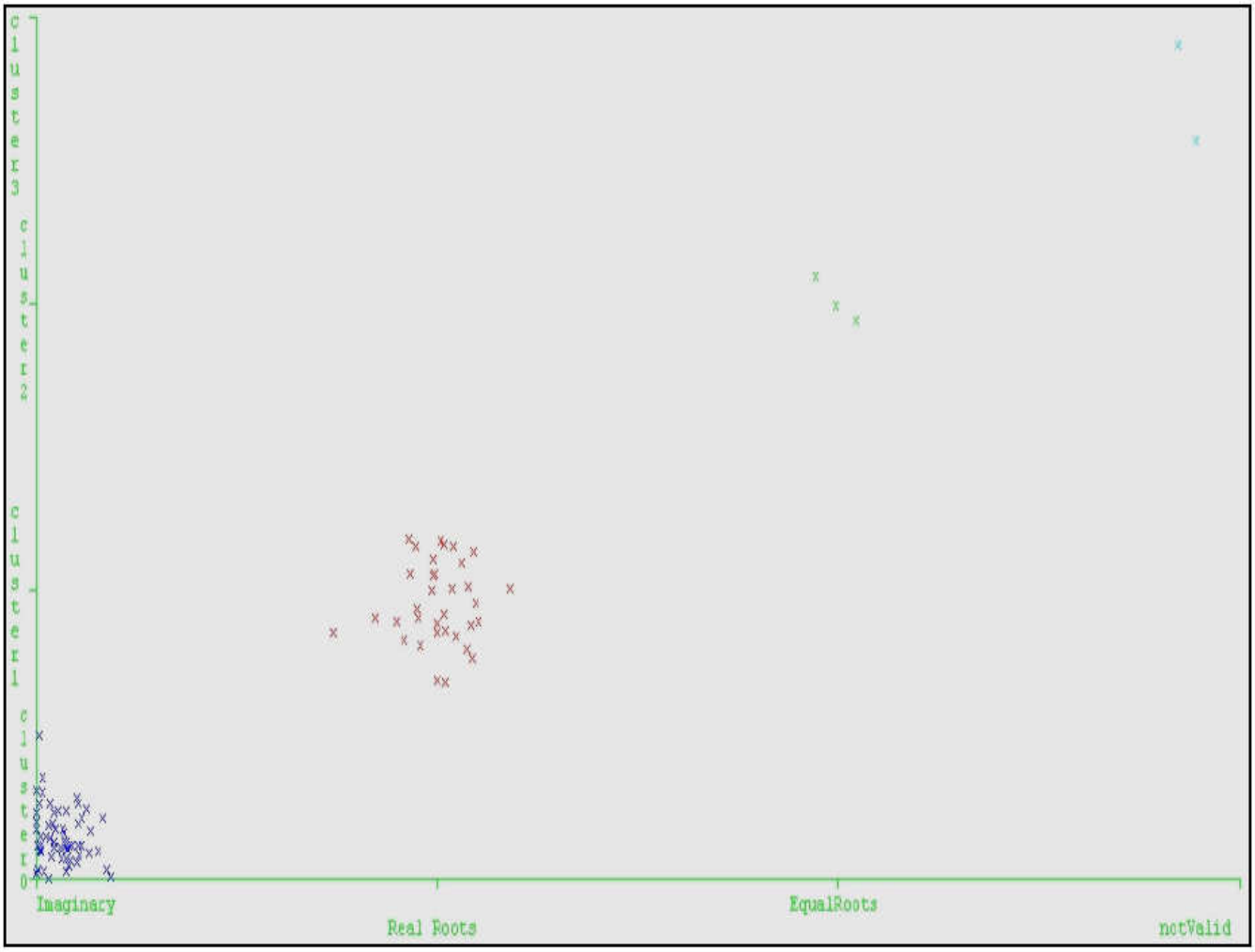

Fig.7 The Final Cluster Assignment of Test Cases.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

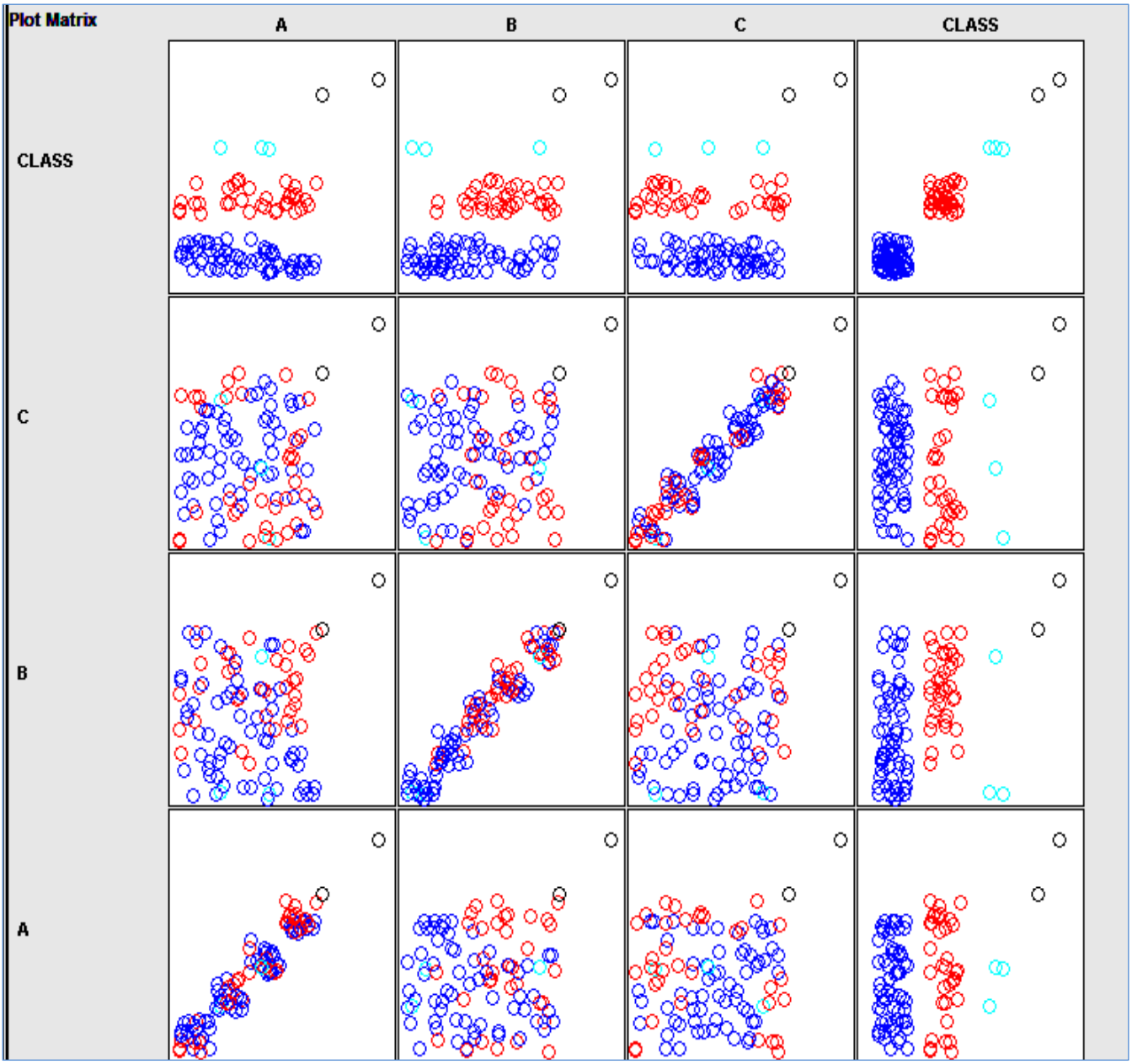

Fig.8: Plot Matrix

V. Conclusion and future work

Software testing is very important and challenging activity. In past, lack of effective testing resulted in many software troubles and has actually brought many social and financial losses. Testing techniques should find the possible number of faults or errors with manageable amount cost and time with finite number of test cases. But software test case design and generation algorithms are exhaustive with respect to the coverage goal defined. Therefore, if a coverage criterion is not properly chosen, the process can generate too many test cases that are infeasible to be considered for practical execution. Also, most of the test cases can be redundant in the sense of exercising common features of the Code under test and revealing common sets of defects. Therefore, other than structural coverage criteria, test-case generation may need to be combined with selection strategies that will minimize redundancy in test suites; and limit the size of test suites. In this study, hierarchical clustering approach has been implemented on the initial test suite from the size and redundancy perspective. With the proposed approach, we have reduced and partitioned the initial test suite into a fixed number of clusters with respect to requirement and branch coverage criteria. Also based on the requirement coverage the redundant test cases are grouped into their appropriate cluster. So with the proposed approach, considerable amount of reduction (95%) has been achieved by the elimination of unnecessary or redundant test cases. The future scope of this study will be the application of soft clustering approach. Also in future, a comparative study between hard and soft clustering technique from test suite minimization perspective will be carried out.

REFERENCES

1. Smith, A., Geiger, J., Kapfhammer, M. and Soffa, M. (2007), Test suite reduction and prioritization with call trees, in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE '07), 2007, pp. 539-540.
2. Parsa, S., and Khalilian, A., (2010), On the optimization approach towards test suite minimization, International Journal of Software Engineering and its applications, Vol. 4, No. 1, January 2010.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

3. Selvakumar, S. and Ramaraj, N., (2011), Regression test suite minimization using dynamic interaction patterns with improved FDE, European Journal of Scientific Research, 2011, Vol. 49, No. 3, pp. 332-353.
4. Hsu, H. and Orso, A., (2009), MINTS: A general framework and tool for supporting test suite minimization, International Conference on Software Engineering (ICSE '09), 2009.
5. R.V Binder. "Testing Object-Oriented Systems Models, Patterns, and Tools". Object Technology Series. Addision Wesley, Reading, Massachusetts, October 1999.
6. Tallam, S. and Gupta, N. (2005), A concept analysis inspired greedy algorithm for test suite minimization,in Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE '05), 2005, pp. 35-42.
7. Chen, Z., Xu, B., Zhang, X. and Nie, C., (2008), A novel approach for test suite reduction based on requirement relation contraction, in Proceedings of the 2008 ACM symposium on Applied computing (SAC '08), 2008, pp. 390-394.
8. Yevtushenko, N., Cavalli, A., and Anido, R., (1999), Test Suite Minimization for Embedded Nondeterministic Finite State Machines, in Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems: Method and Applications, 1999, pp. 237-250.
9. Usaola, M., Mateo, P., and Lamancha, B., Reduction of test suites using mutation, in Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering (FASE'12), 2012, pp. 425-438
10. N. Mansour, K. El-Fakih, Simulated annealing and genetic algorithms for optimal regression testing, Journal of Software Maintenance 11 (1) (1999) 19–34.
11. Jain. Data clustering: 50 years beyond k-means. Pattern Recognition Letters, 31(8):651– 666, 2010.
12. Dibya Jyoti Bora, Anil Kumar Gupta," A Comparative study Between Fuzzy Clustering Algorithm and Hard clustering Algorithm", International Journal of Computer Trends and Technology (IJCTT) ,volume 10 number 2 – Apr 2014,pp. 108-113.
13. A. Khan, A.K.Gupta, D.J.Bora, Profiling of Test Cases with Clustering Methodology, .International Journal of Computer Applications 106(14):32-37.
14. Richard C. Dubes and Anil K. Jain, (1988), Algorithms for Clustering Data, Prentice Hall.
15. L. Kaufman and P. J. Rousseeuw, (1990), Finding Groups in Data: an Introduction to Cluster Analysis, John Wiley and Sons.
16. D.S. Johnson, Approximation algorithms for combinatorial problems Journal of Computer and System Sciences 9 (3) (1974) 256–278.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.