International Journal of Allied Practice, Research and Review

Website: www.ijaprr.com (ISSN 2350-1294)

An Efficient Approach to Test Suite Minimization for 100% Decision Coverage Criteria using K-Means Clustering Approach

Fayaz Ahmad Khan, Dr. Anil Kumar Gupta, Dibya Jyoti Bora

Department of Computer Science and Applications, Barkatullah University Bhopal, (M.P)

Abstract— Test case generation is one of the labour- intensive processes in softer testing. Test case generation has a very strong impact on the effectiveness and efficiency of the software under test. Thus, it is imperative to reduce cost and improve the effectiveness of software testing by automatic the test case generation techniques. But, both manual and automation test case generation techniques generate large and redundant test cases that are infeasible to be considered for practical execution. Thus an efficient approach is needed which can limit the size and redundancy of test suite based on certain code coverage criteria like statement coverage, branch/ decision coverage and dataflow criteria. In this study, K-Means clustering approach is implemented on a test suite which is large in size and contains redundant test cases, in order to reduce the size of test cases to an effective number of test cases for 100% decision coverage of a code.

Keywords—Software testing; Test cases; Test Suite Minimization; Data Clustering; K-means Algorithm

I. Introduction

Software testing is an important phase of a product under development. Software testing plays an important role in evaluating or assessing the quality of software during its development and after its delivery. Therefore, probably, the intension of testing increases exponentially with the product size and its desired level of reliability required [1] .Testing as an important domain of software engineering is executed in different levels during the software development life cycle. Unit testing is accomplished at the time of module development. After unit testing, at the time of module integration, integration testing is done. Similarly, when the system exists as a complete

entity, System testing is applied. Finally, acceptance phase of testing begins for the verification of all the requirements specified in requirement specification document.

Early integration of testing activities not only supports effective test design, which is a critically important activity, but also provides early exposure of errors and prevents movement of errors from requirement specification to design, and thence from design into code. This approach of error prevention reduces cost, minimizes rework, and saves time. The earlier in the development cycle those errors are revealed, the easier and less costly they are to fix. Cost is measured in terms of the amount of time and resources required to correct the defect after it is detected. A defect found at an early stage is relatively easy to fix, has no operational impact, and requires few resources. In contrast, a defect discovered during the operational phase can involve several organizations, can require a wider range of retesting, and can cause operational downtime.

Software test case generation is an important activity in software testing as both test case design and test case execution are time consuming and labour intensive processes. Test cases are designed both by manually and by automatically with some automation tools. But the manual approach is time consuming and error prone way of designing test cases and usually automatic tools are used to generate test cases. The main goal of automatic test-case generation is to generate all possible combinations of test cases from a given test model according to coverage criteria. The common techniques for automatic test cases generation include Symbolic execution, Model based, combinatorial testing, Adaptive random testing and Search based techniques. Although, automatic way of design is better approach in terms of time and cost perspective, but the generated test suite is inefficient and impractical to be considered for practical execution. Also, most of the test cases can be redundant in the sense of exercising common features of the SUT (for instance, the same lines of code) and revealing common sets of defects. Many automatic tools have also been developed based on the above mentioned techniques like PEX as an add-in to Visual Basics for white Box testing implementing Dynamic Symbolic execution[2], CodePro AnalytiX for java[3],]. COVER for real time systems based on model based techniques [4]. CodeProJUnit is a plug-in for Eclipse used for both test case generation and code coverage criteria [5].

II. Test Suite Minimization

The input domain of a program or a module is infinite and cannot be used as a test data, because, Automatic test-case algorithms are usually exhaustive with respect to the coverage goal defined. Therefore, depending on the magnitude of the model, these algorithms may produce a considerable number of test cases. 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. Therefore, other than structural coverage criteria, test-case generation may need to be combined with selection strategies that may help to focus test generation at particular functionalities of interest; minimize redundancy in test suites; and limit the size of test suites.

Thus, it has long been identified as a research problem to find a minimized subset of test cases from the large input domain of a program for the effective testing. The problem of finding a

minimized test suite is known as test suite minimization. The approaches for test suite minimization and selection based on coverage criteria must be done in systematic and care full manner so that reasonable sizes of test cases are selected. Many efforts have also been proposed on how to reduce the size of a test suite while maintaining its effectiveness. Typical techniques include heuresistic approaches [6][7][8][9], the genetic algorithm based[10] and ILP based [11][12]. In [13], where it is stated that, testing a program P consisting of different components or modules like p1,p2,p3,p4….p(n), with a few or small number of test cases t1,t2,t3 from the input domain (D), will exercise only few components or modules of P, and the faults in other modules will go undetected. So both test case selection and reduction must be performed with respect to coverage criteria defined in order to exercise all the structure components of software under test.

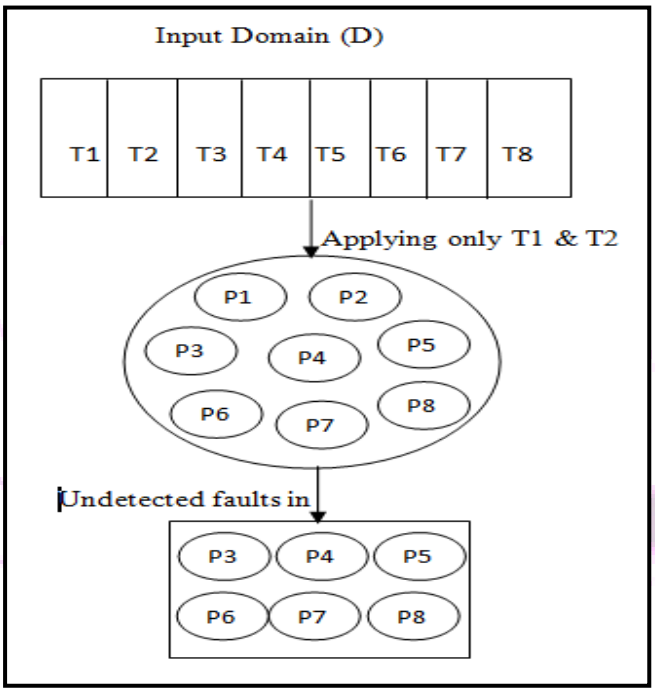

Figure.1 (Inspired from [13])

We have implemented the K-means Clustering approach for test suite minimization of a sample program or module in Fig.2, which determines the largest of three numbers. The test suite is generated using random testing technique and is augmented with some additional test cases in order to consider all the types of input parameters including valid, invalid, and out of range values. There are three input variables/ parameters and all of these are declared as integer type, so each will take any values in between 2^{-128} to 2^{127}. In order to keep the test cases in a reasonable size, we have only generated test cases up to 100 for all the three variables. The generated test suite depicted in Fig.3 is not effective in terms of size and has lot of redundant test cases. Thus, it is required that the test suite needs to be minimized and a subset of test cases may be selected so that the size and redundancy in test suite is properly eliminated.

```
import java.util.*;
class test11
{
 public static void main(String[] args)
{
  int a,b,c;
 Scanner s=new Scanner(System.in);
 System.out.println(" Input values for A : B and C ");
  a=s.nextInt();
  b=s.nextInt();
  c=s.nextInt();
 if(((a>=0)&&(a<=100))&& ((b>=0)&&(b<=100))&&((c>=0)&&(c<=100)))
  {
   System.out.println("Valid input");
   If (a>b)
 {
   If (a>c)
   System.out.println("A is greater");
 }
Else if (b>c)
    {
      System.out.println("B is greater");
    }
  Else
   {
     System.out.println("C is greater");
   }}

 Else

 {  System.out.println("NotValid input");

 }}}
```

Figure.2 (A sample Module)

Test Case ID	A	B	C	EXPECTED OUTCOME
t1	10	12	5	B
t2	15	10	9	A
t3	25	35	45	C IS BIGER
t4	A	C	D	WRONGINPUT
t5	-1	-4	5	WRONGINPUT
t6	40	45	50	C IS BIGER
t7	50	55	45	B
t8	60	55	59	A
t9	70	75	80	C IS BIGER
t10	80	70	75	A
t11	90	80	60	A
t12	70	90	80	B
t13	0	0	0	WRONGINPUT
t14	95	85	75	A
t15	75	85	95	C IS BIGER
t16	85	95	80	B
t17	100	95	90	A
t18	80	100	90	B
t19	85	99	100	C IS BIGER
t20	101	110	120	WRONGINPUT

Figure.3 (Test Suite)

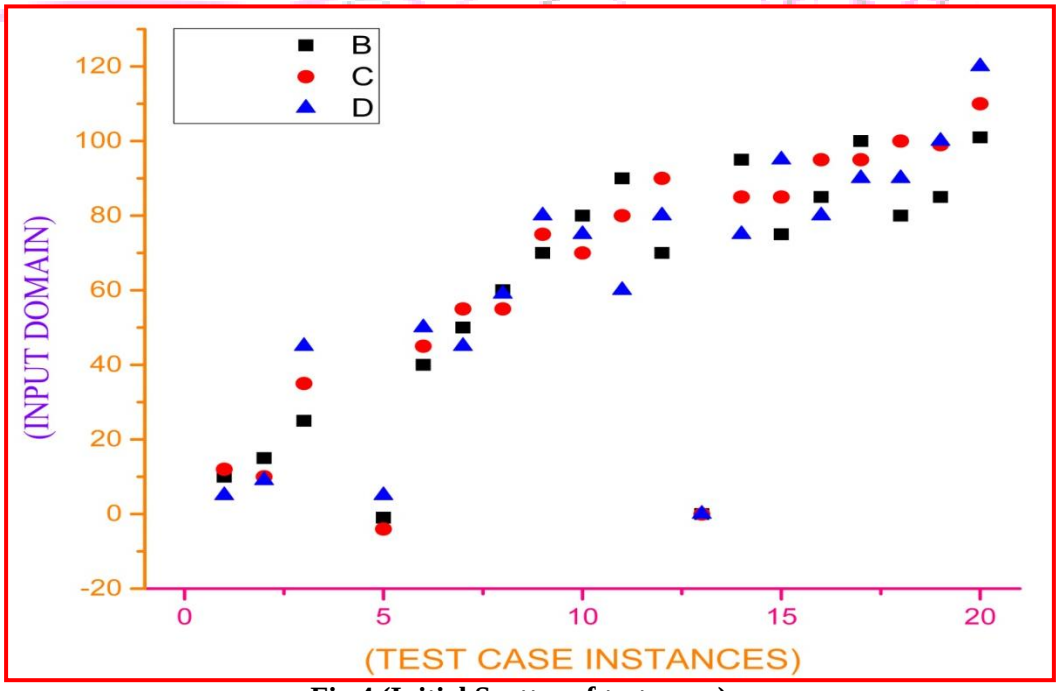

Fig.4 (Initial Scatter of test cases)

III. Implementation of proposed Approach

Cluster analysis an important domain of data mining, groups the data into meaningful groups/ clusters based on the information found in the data. The aim is that the objects in a cluster are related to one other and separate or different from the objects in other groups or clusters. The assignment or inclusion of objects or observations to a cluster is done on the basis of similarity or dissimilarity matrix which is commonly known as proximity matrix [14]. The proximity matrix is an n by m matrix containing all the pair wise dissimilarities or similarities between the observations being considered. The most commonly used distance measures include Minkowski metric for ratio scaled data. The other distance measures based on Minkowski metric are City block (Manhattan, taxicab, L1 norm) distance, Euclidean distance and Supermun (Lmax norm, L∞ norm) distance [15]. For measuring similarity between binary vectors, the common measures include SMC (Simple Similarity Measure) and JC (Jaccard coefficient). Based on these distance measures, different numbers of clustering techniques have been proposed over the years. The most commonly used techniques are Hierarchical clustering (or nested) and Partitional clustering (or unnested) techniques. Hierarchical techniques produce a nested sequence of partitions, with a single inclusive cluster at the top and singleton clusters of individual points at the bottom. While Partitional techniques create a one level (Unnested) partitions of the data points into a desired number of clusters at once. Here, in this study, we have implemented K-Means Partitional clustering techniques to group the objects or test cases in the desired number of clusters. The *k*-Means method is considered one of the simplest and most classical methods for data clustering [17] and is also perhaps one of the most widely used methods in practical implementations because of its simplicity. The decision of K is very important in K-means Algorithm because they yield different results based on the location of K initial centroids [16]. The K-Means approach initially treats a data set as a single cluster and proceeds by partitioning it into the 'K' specified number of clusters. The algorithm seeks to minimize the sum of the squared distance of a point from the center of its cluster. Finally the algorithm stops when the centroids do not change.

We have implemented the K-Means Algorithm to partition the input domain or test suite of sample module into a desired number (K) of partitions. The intuition behind the approach is that, the given test suite in Fig.3 is to be partitioned into a specific number of clusters in order to reduce the size and eliminate the redundant test cases. Here, we have specified the value of K=4 in order to group the test cases according to a coverage criteria of the sample code. A coverage criterion is an important factor while applying any reduction techniques because it gives the measure of structural components executed by a test suite. The test suite reduction is guided by the code coverage criteria defined in order to select a test suite that will give 100% code coverage. There are many ways to measure code coverage; like statement coverage, branch coverage, decision coverage, multiple decisions / condition coverage, loop coverage and path coverage. So it very important to consider any of the above code coverage criteria while reducing the test cases. A selected test suite after reduction should be able to give 100% code coverage; otherwise the faults will remain hidden in other parts or components of software under test.

In this study, we have implemented the proposed clustering approach with k=4 (Groups/ Clusters) to the suite in Fig.3 in order to group the similar test cases into an appropriate number of groups based on the Euclidean similarity distance measure. After successful implementation, the input domain or test suite is clustered or grouped into a four different partitions or clusters (C1, C2, C3, C4) as shown in **Figure.5.** Thus each partition or clusters contain test cases which are similar within the partition and dissimilar with other test cases in rest of the clusters. Now, instead of executing the sample code with twenty (20) test cases, only four (4) test cases are required for the 100% branch coverage. The percentage of reduction achieved with the proposed technique is calculated as:

$$\% \text{ of reduction} = \frac{[20-4]*100}{[20]} = 80\%$$

Thus from our proposed approach we have achieved 80% reduction in test cases with 100% branch coverage. The **Figure.4** is the random initial scatter of test cases before reduction and **Figure.5** depicts the final cluster assignment of test cases in their appropriate cluster numbers after reduction. So from **Figure.5,** it is clearly observed each cluster (C1, C2, C3, C4) is unique and contain no overlapping test cases. A single test case from each cluster as shown in **Figure.6,** would an effective subset of test cases required for the practical execution of code. But due to fuzzy nature of the data, few test cases are assigned to a wrong groups or clusters, and future scope of this study would be to remove this inconsistency.

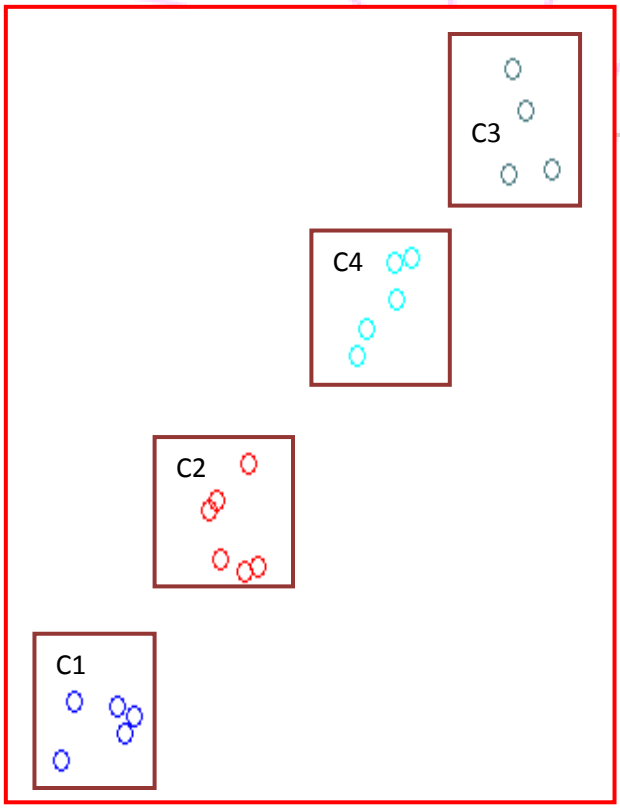

Fig.5 (Final Scatter with Cluster Assignment)

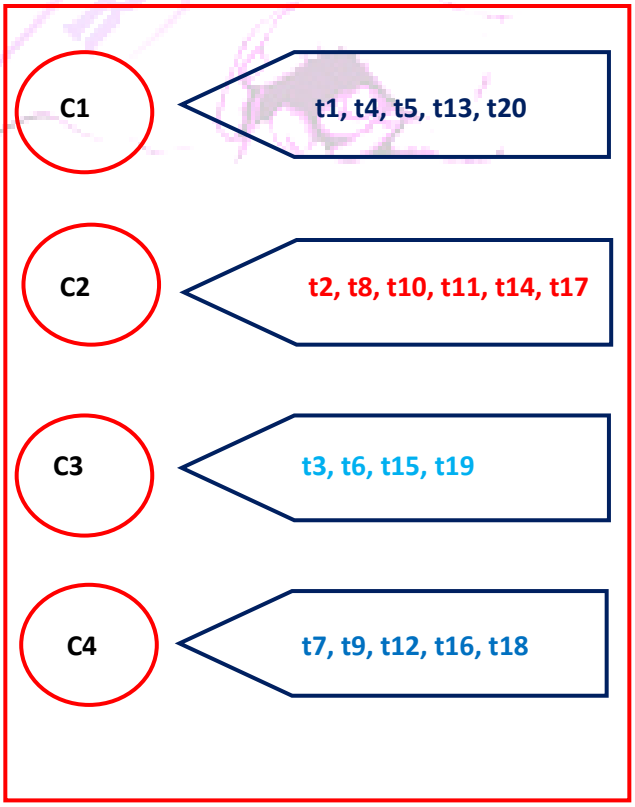

Fig.6 (Actual test cases possessed by each cluster)

IV. Conclusion and Future Work

In this study, we have proposed an efficient approach for test suite minimization for an effective and an efficient testing of a sample code which has multiple decision statements. Test suite minimization is important activity during software testing because the available initial test suite is not ample for testing. The reasons are that the generated test suite is both large and contains redundant test cases. There are numerous reasons for the large size of test suite, one is that the input domains of program variables are large and the second is that the test suite usually undergoes expansion due to the addition of test cases when ever requirements undergo certain changes. With the initial test suite (a test suite of 20 test cases), the decision statements present in the sample code are traversed multiple times by the test cases. Thus this test suite will take much time for the execution and also will be an inefficient approach for resource utilization. Through the proposed approach the test cases are clustered into four groups (C1, C2, C3, C4) and a single test case from each group will form a test suite of size four (4).Thus through our proposed technique we have achieved 80% reduction or minimization in test cases with 100% decision Coverage.

V. REFERENCES

[1]. John A. N. Lee e Xudong He. A methodology for test selection. Technical report, Blacksburg, VA, USA, 1988.

[2]. N. Tillmann and J. D. Halleux, "Pex–White Box Test Generation for .NET", LNCS 4966, B. Beckert and R. H. ahnle, eds., pp. 134–153: Springer-Verlag Berlin Heidelberg, **(2008)**.

[3]. "CodePro Analytix User Guide", 13 April, 2014; https://developers.google.com/java-devtools/codepro/doc/.

[4]. A. Hessel and P. Pettersson, "COVER – A Real-Time Test Case Generation Tool".

[5]. "JUnit Test Case Generation," April 7, 2014; https://developers.google.com/java-dev-tools/codepro/doc/features/junit/test_case_generation.

[6]. J. Offutt, J. Pan, J. Voas, "Procedures for reducing the size of coverage based test sets", in:Proceedings of 12th International Conference on Testing Computer Software, 1995, pp. 111–123.

[7]. J. Horgan, S. London, ATAC: "A data flow coverage testing tool for C", In: Proc. Symposium of Quality Software Development Tools, 1992,pp. 2–10.

[8]. T.Y. Chen, M. Lau, "A new heuristic for test suite reduction", Information and Software Technology 40 (5-6) (1998) 347–354

[9]. P. Saraph, M. Last and A. Kandel, "Test case generation and reduction by automated input-output analysis",

vol. 1, pp. 768-773.

[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]. J. Hartmann, D. Robson, "Revalidation during the software maintenance Phase", Proceedings of International Conference on SoftwareMaintenance (1989) 0–80.

[12]. J. Black, E. Melachrinoudis, D. Kaeli, "Bi-criteria models for all-uses test suite reduction", in: Proceedings of 26th International Conference on Software Engineering, IEEE ComputerSociety, Washington,DC, USA, 2004, pp. 106–115.

[13]. F.A.Khan., A.K.Gupta. D.J.Bora."Test Suite Minimization for statement Coverage testing using K-Means Clustering Approach" International Conference on Emerging trends in information technology, pages 26-31, March-2015.

[14]. L. Kaufman and P. J. Rousseeuw, (1990), *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley and Sons.

[15]. Richard C. Dubes and Anil K. Jain, (1988), *Algorithms for Clustering Data*, Prentice Hall.

[16]. Data Mining: Concepts and Techniques, 2nd Edition, Jiawei Han and Micheline Kamber, Morgan Kauffman,2006.

[17]. A. Jain. Data clustering: 50 years beyond *k*-means. *Pattern Recognition Letters*,31(8):651– 666, 2010.