A New Algorithm to Test Suite Reduction Based on Cluster Analysis

S. Parsa, A. Khalilian and Y. Fazlalizadeh
Computer Engineering Department, Iran University of Science and Technology
Tehran, Iran
{parsa}@iust.ac.ir, {khalilian, ya_alizadeh}@comp.iust.ac.ir

Abstract—Regression testing is a significant activity which occurs during the maintenance level of a software lifecycle. However, it requires a large amount of test cases to test new or modified parts of the software. To address the issue, Test Suite Reduction techniques have been presented. An appropriate technique should generate a minimized test suite which exercises different execution paths within a program while retaining the fault detection capability of the suite admissible. To achieve this, a heuristic algorithm is proposed in this paper. The new algorithm clusters test cases based on the similarity of their execution profiles and sample some representatives to form the reduced test suite. The results of applying the new algorithm to the Siemens suite and comparing to the H algorithm manifest interesting insights into the effectiveness of the proposed algorithm.

Keywords- software regression testing; testing criteria; test suite minimization; test suite reduction; fault detection

1. INTRODUCTION

Software regression testing is a significant activity to the development and maintenance of evolving software. A major difficulty with the regression testing is the excessive number of test cases accumulated while generating new test cases to test any new or modified functionality within the program [1]. To resolve the difficulty and to reduce the excessive cost of regression testing, various coverage-based and distribution-based techniques have been proposed [2, 3, 4, 5, 6, 7, 8]. These techniques attempt to permanently discard redundant test cases and retain the most effective ones to reduce the excessive cost of regression testing [6]. An effective technique for regression testing is to find a minimal subset of test cases which exercises all the test requirements as the original set does [9]. A suitable subset could be found during the test case generation or after creating the test suite. Apparently the less the number of test cases the less time it takes to test the program. This consequently improves the effectiveness of the test process.

The technique is commonly known as test suite reduction or test suite minimization in the literature and the resulting suite is called representative set [3].

Almost all the previous test suite reduction techniques could significantly reduce the size of the test suites. But an important issue deals with how well these reduced suites can be compared with their corresponding un-reduced suites using other criteria rather than the suite size criterion. Since the purpose of test case execution is to detect faults in the software, one measure of the suite quality is its fault detection capability. In fact, a potential drawback in test suite reduction studies is that permanently discarding some test cases of the test suite may cause high decrease in fault detection effectiveness of the reduced suite. Thus, the tradeoff between the time required to execute and manage test suites and their fault detection effectiveness should be considered when applying test suite reduction techniques [2].

Coverage-based techniques attempt to maximize the code coverage of a program in order to ensure that all different execution paths are exercised and the majority of faults are revealed. The reason is that any uncovered parts of the program, is likely to be faulty. However, code coverage solely is not a sufficient criterion for selecting representative test cases from the original test suite. This is because high code coverage can be achieved by selecting simple test cases that do not reflect the program execution in real situations [10] and the resultant reduced suite may not be efficient in detecting faults.

On the other hand, execution profiles which are used in distribution-based techniques [8] reflect additional information about test cases that can be helpful in determining effective test cases when reducing the original test suite. Cluster filtering [11] and failure-pursuit [12] are examples of distribution-based techniques. Cluster filtering uses automatic cluster analysis to partition the test pool in terms of its test cases profile similarity. Then, a number of test cases are sampled from each partition. One-per-cluster sampling is a method which randomly selects a test case from each cluster. The other method is failure-pursuit sampling in which k nearest neighbors of a failed test case found by auditing samples from one-per-cluster method are selected. However, these methods perform rather well, but do not necessarily make full coverage [8]. Moreover, empirical studies [8] show that their functionality varies with the size of the test suite. These studies also indicate that coverage-based and distribution-based techniques often do not select similar test cases and are complementary because they find different faults.

To address the limitations of the coverage-based and distribution-based approaches, in this paper a heuristic algorithm combining the ideas of the both approaches, is proposed. Based on this algorithm, test cases are clustered according to their execution profile similarities, and then test cases are sampled from each cluster as long as a same coverage of the original suite is achieved. While sampling from each cluster, the algorithm attempts to select the test case which covers the maximum number of unmarked requirements.

In order to evaluate the applicability of the proposed approach, we conducted experiments on the Siemens suite.
We also implemented the well-known H algorithm [3], to compare the results of reducing suites using our distribution-based heuristic algorithm with those of minimizing test suites using the H algorithm.

The rest of the paper is organized as follows: Section 2 discusses the background of the test suite reduction techniques. Section 3 contains the outline of the proposed approach. Section 4 describes the empirical studies and the obtained results. Finally, conclusions are mentioned in section 5.

II. BACKGROUND AND RELATED WORK

A. Test Suite Reduction

The first formal definition of test suite reduction problem introduced in 1993 by Harrold et al. [3] as follows:

Given: \{t_1, t_2, \ldots, t_m\} is test suite \(T\) from \(m\) test cases and \(\{r_1, r_2, \ldots, r_n\}\) is set of test requirement that must be covered in order to provide desirable coverage of the program entities and each subsets \(\{t_i, r_1, \ldots, t_j\}\) from \(T\) are related to one of \(r_j\) such that each test case \(t_j\) belonging to \(T\) satisfies \(r_j\).

Problem: Find minimal test suite \(T'\) from \(T\) which satisfies all \(r\)s covered by original suite \(T\).

Generally the problem of finding the minimal subset \(T'\), \(T' \subseteq T\) which satisfies all requirements of \(T\), is NP-complete [13], because we can reduce the minimum set-cover problem to the problem of test suite minimization in polynomial time. Thus, researchers use heuristic approaches to solve this problem. One heuristic method proposed by Harrold et al. [3], tries to find the smallest representative set that provides the same coverage as the entire test suite does.

Related work in the context of test suite reduction can be classified into two main categories: The works in which a new technique is presented [3, 4, 5, 14, 9, 6, 7], and empirical studies on the previous techniques [2, 13, 15, 1]. The works which propose a new approach commonly include heuristic algorithms [14], genetic algorithm-based techniques [16] and approaches based on integer linear programming [17].

Testing criteria are defined in order to help the selection of subsets of the input domain to be covered during testing. For example, a code coverage criterion provides test suite adequacy with respect to coverage of the program entities and also provides a check on its quality. Assuming testing criteria \(C\) which satisfies by the test suite \(T\), a test case, \(t\), is redundant if the suite \(T - \{t\}\) also satisfies \(C\) [5]. Therefore, removing those test cases which are redundant with respect to some specific criteria preserves test suite’s adequacy with respect to it. In previous empirical studies [13] researchers commonly apply various code coverage criteria in their reduction techniques. The results of empirical studies [2, 15] show that the more percentage of the test suite size is reduced the more percentage of faults will be lost.

B. Coverage-based Techniques and the H Algorithm

The main purpose of traditional software testing is to achieve the maximum code coverage [8] and the main idea is that the test suite is capable of exercising the whole program under test. In other words, there must be some test cases per program element (including statement, branch, def-use pair and so on) which can cover it. This is due to unreliability of the uncovered parts of the program during testing. But as mentioned above, code coverage alone is not sufficient for selecting test cases. If the code coverage is the main factor in determining the quality of a test suite, then it is desirable to have the minimal subset of the test suite that covers the program elements the same as the original suite [8]. Even if the code coverage is not main factor in test suite quality, it is possible to use it as the first step of other reduction approaches.

One of the famous and efficient coverage-based heuristics to test suite reduction problem is H algorithm which has been proposed by Harrold et al. [3]. This algorithm performs as follows: Given test suite \(T\) and testing requirements \(r_1, r_2\) which should be covered in order to achieve proper testing coverage, this algorithm considers the subsets \(T_i\) \(T_j\) from \(T\) where each test case \(t_i\) in \(T_i\) can be used to satisfy \(r_i\). First, it selects all test cases which are in \(T_0\) of cardinality one, insert them into the reduced suite and marks all the corresponding \(T_0\). Then, it considers the \(T_0\) of cardinality two. Repeatedly, those test cases that are in the maximum number of \(T_0\) of cardinality two are selected and added to the reduced suite. All unmarked \(T_0\) containing these test cases are marked. This process continues for \(T_0\) of cardinality 3, 4, . . . , \(max\), where \(max\) is the maximum cardinality of \(T_0\). Whenever tie occurs in selecting test cases from \(T_0\) of cardinality \(m\), \(T_0\) of higher cardinalities are checked and finally a test case is selected randomly.

C. Distribution-based Techniques and Cluster Analysis

Distribution-based techniques select test cases according to how their profiles are distributed by dissimilarity metric in the multi-dimension profile space [8]. Dissimilarity metric is a function that produces a real number for each pair of profiles which shows the degree of their dissimilarity. Distribution-based methods can be described using two closely related techniques: cluster filtering and failure-pursuit. Cluster filtering uses automatic cluster analysis to partition the test pool [8]. The purpose of cluster analysis is to partition the population such that objects with similar attributes are in the same cluster. After clustering, test cases are sampled from each cluster. For example, one-per-cluster sampling selects one test case from each cluster. The features of this type of sampling can be described as follows: 1) redundancy is minimized in the selected subset because one sample is selected from each cluster containing similar test cases, 2) unusual executions are selected because they are in one cluster solely, 3) selected test cases test distinct behaviors of the program.

The distribution-based techniques used for filtering test cases and reducing test suites, determine some characteristics of the distribution profile that are likely to detect faults, and apply them during the selection of test cases. For example, clusters that have similar profiles contain redundant test cases and selecting one or a few representatives are sufficient. Also isolated profiles contain test cases that indicate unusual conditions and are likely to detect faults. Test case cluster
filtering is defined by selecting clustering algorithm, dissimilarity metric, number of clusters and sampling method [8].

III. THE PROPOSED APPROACH

As mentioned in previous section, one of the main purposes of software testing is that selected test cases can exercise the whole program under test in order to increase reliability of the software. Hence, coverage-based techniques have been widely applied to reduce test suites. On the other hand, the distribution of execution profiles in distribution-based approaches, reflect more information about test cases which are useful in test suite reduction process. For example, using distribution of test cases, one can determine unique test cases from similar (or redundant) ones. Thus, combining the ideas of coverage-based and distribution-based techniques, we present a heuristic algorithm to reduce regression test suites. Our approach requires that the execution profiles of the test cases are computed according to some criteria. Then, using a clustering algorithm and the execution profiles, test cases are clustered according to the similarity of their execution profiles. In this way, test cases with similar execution profiles are placed in the same cluster, and are likely to cover similar program elements. After that, we apply a heuristic method for sampling test cases from clusters so long as the coverage of the reduced test suite equals to the coverage of the original suite. The proposed algorithm is shown in the Fig. 1. Initially, the heuristic sorts the clusters based on the number of test cases within each cluster ascending. In next step, the algorithm repeatedly considers the first cluster to the last one until the unmarked requirements are satisfied. In each iteration, a sample test case is selected from each cluster. At first, all test cases of the current cluster are added to the list. If the list contains one test case which covers some unmarked requirements, it is added to the reduced suite. Otherwise, the next cluster is considered. If the list contains more than one test case, the function SelectTest is called to select one test case which is then added to the reduced suite. The function SelectTest selects a test case from a certain cluster. This function first computes the number of unmarked requirements that each test case in testCaseSet covers. Then, test cases with maximum value for numUnmarked are selected in testList1. If testList1 contains one element, it will be selected and returned. Otherwise, for each test case in testList1, the number of covering requirements by the test case is computed and test cases with maximum values are selected in testList2. If the cardinality of testList2 is 1, the test case will be selected. Otherwise a test case is randomly selected.

IV. EMPIRICAL STUDIES

A. Subject Programs

To evaluate and compare our results with prior experimental studies in test suite reduction area, we conducted the experiments with seven C programs as subjects, the Siemens suite. These programs are associated with faulty versions besides test suites with different coverage criteria. Each faulty version of each program in the Siemens suite contains a single fault seeded in it. Like many studies in this area [2], we used branch coverage adequate test suites. A test suite is branch coverage adequate if its coverage with respect to program branches is the same as test pool's branch coverage.

B. Measures

To investigate the effectiveness of our approach and compare it to the coverage-based H algorithm, we measured the following from our experiments:

- The percentage suite size reduction $= \frac{|F| - |F_{Real}|}{|F|} \times 100$

where $|F|$ the number of test cases in the original is test suite and $|F_{Real}|$ is the number of test cases in the reduced test suite.

- The percentage fault detection loss $= \frac{|F| - |F_{Real}|}{|F|} \times 100$

where $|F|$ is the number of distinct faults exposed by original test suite and $|F_{Real}|$ is the number of distinct faults detected by the reduced suite.

C. Clustering and Analysis Tools

In order to cluster execution profiles of test cases, Weka 3.5.8 [19] has been used. Weka is a java-based tool that provides a uniform interface to many different learning algorithms. Besides, we used SAS 9.1.3 [20] to create box plots. Box plot diagrams are commonly used to visualize the empirical results in test suite reduction studies.

D. Clustering Algorithm and Number of Clusters

Our experiments include a large number of test cases each of which covers a lot of requirements (here requirements are different branches of a program). Thus, a large number of high dimensional data should be clustered. However, fast and effective clustering of such data is extremely difficult because of their large volume and high dimensionality [21].

On the other hand, clustering algorithms need to set the number of clusters. Selecting the number of clusters affect the separation of test cases. The CLOPE algorithm [21] is a fast and efficient method for clustering large and high dimensional data. In this algorithm, a value called Repulsion is set which controls level of intra-cluster similarity. The number of clusters changes by varying this value. We felt that as a first step for exploration, using the CLOPE algorithm would lead to a good compromise between fast and effective clustering and a reasonable number of clusters.

E. Experiment Setup and Results

Our experiments follow a setup similar to that used by Rothermel et al [2]. For each program, we created branch coverage adequate test suites for six different suite ranges named as B, B1, B2, B3, B4 and B5. For each suite range, we first selected $X \times LOC$ test cases randomly from the test pool and added to the test suite, where $X$ is 0, 0.1, 0.2, 0.3, 0.4 and 0.5 respectively and LOC is the number of lines of
In order to evaluate the effectiveness of our approach, we implemented our algorithm and applied it to the generated suites. We also implemented the H algorithm and conducted some similar experiments to compare both algorithms. The results of this experiment are shown in Fig. 2 from A to F. The box plots in these figures show the distribution of the percentage of size reduction (SR) and percentage fault detection loss (FL) of suites for all suite size ranges from B to B5 for each program. In each figure, box plots are paired such that for each suite size range, white pair of plots shows percentage of size reduction and gray pair of plots shows the percentage of fault detection loss. In each pair, left side box indicates for our algorithm and the right side one indicates for the H algorithm. As can be seen, when the percentage of size reduction of both algorithms are closely equal, the percentage of fault detection for our approach is less than the H algorithm. When the percentage of fault loss is nearly equal, our algorithm have greater suite size reduction. These results demonstrate the effectiveness of our approach over the H algorithm both in size reduction and fault detection loss.

F. Threats to Validity

In this section, we describe the potential threats to validity of our study. In our study, the measurement for the percent of fault loss assumes simple model for cost which treats all faults as equally severe. But in practice, faults have wide range of severity. Another issue is composition of the test suites. However, we utilized the process of creating suites which was employed in previous studies [2]. The other threat is the subjects we used. Siemens programs are widely used in software testing studies. But, they are not real programs and the faults are hand-seeded.

V. Conclusions

In this paper, a new heuristic algorithm to reduce regression test suites was presented. The proposed algorithm combines the ideas of coverage-based and distribution-based approaches to test suite reduction. Unlike distribution-based approaches, the reduced suite is coverage adequate with respect to the original suite in addition to containing test cases with higher fault detection effectiveness. The results of experiments on the wide range of suite sizes for the Siemens suite demonstrate the effectiveness of the proposed approach in size reduction and decreasing fault loss.

REFERENCES


