Reducing the Cost of Regression Testing by Identifying Irreplaceable Test Cases

Chu-Ti Lin\(^1\), Kai-Wei Tang\(^2\), Cheng-Ding Chen\(^1\), Gregory M. Kapfhammer\(^3\)

\(^1\)Dept. of Computer Sci. and Info. Eng.
National Chiayi University
Chiayi, Taiwan
{chutilin, s0990394}@mail.ncu.edu.tw

\(^2\)Cloud System Software Institute
Institute for Information Industry
Taipei, Taiwan
kwang@iii.org.tw

\(^3\)Dept. of Computer Science
Allegheny College
Meadville, PA
gkapfham@allegheny.edu

Abstract—Test suite reduction techniques decrease the cost of software testing by removing the redundant test cases from the test suite while still producing a reduced set of tests that yields the same level of code coverage as the original suite. Most of the existing approaches to reduction aim to decrease the size of the test suite. Yet, the difference in the execution cost of the tests is often significant and it may be costly to use a test suite consisting of a few long-running test cases. Thus, this paper proposes an algorithm, based on the concept of test irreplaceability, which creates a reduced test suite with a decreased execution cost. Leveraging widely used benchmark programs, the empirical study shows that, in comparison to existing techniques, the presented algorithm is the most effective at reducing the cost of running a test suite.

Keywords—regression testing; test suite reduction; code coverage

I. INTRODUCTION

According to the IEEE definition \([1]\), a test case is a set of input data and expected output results which are designed to exercise a specific software function or test requirement. During testing, the underlying software system will be executed to examine the associated program path or to determine the correctness of a software function. It is difficult for a single test case to satisfy all of the specified test requirements. Hence, a considerable number of test cases are usually generated and collected in a test suite \([2]\).

Evolutionary development, incremental delivery, and software maintenance are common in software development \([3]\). In such development processes, the functionality of a software system may be refined to meet the customer’s needs or may be delivered incrementally. Each time the software developers modify the system, they may also introduce some faults. New tests should be added to ensure the quality of new functions. The existing test cases should also be re-executed in order to detect the faults caused by imperfect debugging. Such an activity is called regression testing \([1]\). In the process of software development, more and more test cases will be included, thus often causing some test requirements to be associated with more than one test case.

If software developers can reduce the test suite by removing the redundant test cases, while still ensuring that all test requirements are satisfied by the reduced test suite, then testing may be more efficient. That is, given the original test suite \(T = \{t_1, t_2, t_3, \ldots, t_n\}\) and a set of test requirements \(R = \{r_1, r_2, r_3, \ldots, r_m\}\), the goal is to find a subset of the test suite \(T\), denoted by a representative set \(RS\), to satisfy all the test requirements satisfied by \(T\). The process of finding the representative set is called test suite reduction \([4]\), \([5]\).

The minimum set cover problem can be reduced to the test suite minimization problem in polynomial time \([6]\). Karp proved that the set cover problem is NP-complete \([7]\). Many techniques have been proposed to obtain the near-optimal solution for the test suite reduction problem. Even though the representative sets produced by these techniques are not guaranteed to be optimal, they can significantly decrease both the size of the test suite and the cost associated with its execution. Yet, to the best of our knowledge, most of the existing reduction algorithms ignore the significant differences in the execution costs of the tests. In response to this limitation, this paper describes and empirically evaluates a technique that uses the concept of test irreplaceability to create a representative set with low execution cost.

II. RELATED WORK

A. Greedy Algorithm

The Greedy algorithm is a commonly-used method for finding the near-optimal solution to the test suite reduction problem \([8]\). This algorithm repeatedly removes the test which covers the most unsatisfied test requirements from the test suite set \(T\) to \(RS\) until all of the requirements are covered. Many existing test suite reduction methods are based on the concept of the Greedy algorithm \([9]\). In other words, many algorithms repetitively choose the “best” test case to obtain the near-optimal solution from the locally optimal solutions.

B. GE and GRE Algorithms

If a test requirement only can be satisfied by a specific test case, then that test can be called the essential test case for that requirement \([10]\). If the essential test cases are not inserted into the representative set early in the reduction process, some of the selected test cases may become redundant with a high probability. However, the Greedy algorithm does not specifically deal with the essential test cases as early as is possible. Because the Greedy algorithm does not ensure that the essential test cases are chosen first, Chen and Lau \([10]\) proposed two algorithms, called GE and GRE, to address this problem. The GE algorithm will choose all of the essential test cases first and then will apply the
A. individual execution costs when choosing the test cases. As such, the cost of a test should be a more important consideration for achieving cost-effective testing than the size of the test suite. Thus, it is necessary to consider individual execution costs when choosing the test cases.

C. HGS Algorithm

The HGS algorithm proposed by Harrold et al. [4] is another approach to test suite reduction that has received considerable attention. Let $T_i$ (for $i = 1, 2, 3, \ldots, m$) represent the subsets of $T$, with each subset $T_i$ containing all of the test cases that satisfy the $i$-th test requirement. The HGS algorithm will determine the representative test cases for each subset and include them in the representative set.

III. Reducing the Execution Cost of a Test Suite

Due to the differences in the execution costs between the test cases, the representative set with the smallest number of tests may not be the one with the minimum execution cost. As such, the cost of a test should be a more important consideration for achieving cost-effective testing than the size of the test suite. Thus, it is necessary to consider individual execution costs when choosing the test cases.

A. Review of the ReduceWithRatio Algorithm

The coverage increase per unit of cost consumption may be an intuitive metric to evaluate a test case. Given that $Coverage(t)$ represents the number of uncovered test requirements satisfied by test case $t$, Ma et al. [11] and Smith and Kapfhammer [12] evaluated the test cases using:

$$\text{Ratio}(t) = \frac{Coverage(t)}{Cost(t)},$$

where $Cost(t)$ represents the execution cost of the test case $t$. A higher value of $\text{Ratio}(t)$ implies that the test case is expected to be more cost-effective; in contrast, a lower value of $\text{Ratio}(t)$ may indicate that the test is less desirable.

The Greedy algorithm repeatedly includes the test case with the maximum $Coverage(t)$ until all of the test requirements are satisfied. Instead of $Coverage(t)$, Smith and Kapfhammer’s algorithm, hereafter called ReduceWithRatio, repeatedly includes the test case with the maximum $\text{Ratio}(t)$ until all of the test requirements are satisfied.

B. Reducing with Irreplaceability

In preliminary studies, we found circumstances in which ReduceWithRatio did not produce the best reduced test suite. As an illustrative example, let’s consider Table I’s test suite that satisfies six requirements with four tests that have varying cost. Table II demonstrates the steps that ReduceWithRatio would take to reduce the test suite from Table I. This table shows that the ratio-based method creates the representative set $RS_1 = \{t_1, t_2, t_3\}$ with an execution cost of 14. However, the subset $\{t_2, t_3\}$, with cost 10, is enough to satisfy all test requirements. That is, the best representative set will exclude $t_1$ in order to minimize the execution cost.

In fact, if the test requirement $r$ that is satisfied by the test case $t$ can also be satisfied by many other test cases, there is a high probability that $r$ can still be satisfied even though $t$ is not included in the representative set. Therefore, we posit that $t$ has a higher replaceability with respect to $r$ in this case. According to our initial observations, we found that:

1. A representative set may not have the lowest execution cost if it includes a test case with high replaceability.

2. Because the ReduceWithRatio algorithm only considers the coverage and execution cost of a test case, the test cases with high replaceability frequently may be selected for inclusion in the representative set.

In prior work, Jones and Harrold [13] pointed out that test suite reduction algorithms may choose a test case according to its contribution, or goodness, based on some characteristics of a program. One measure of the contribution of a test case $t$ to the test suite $T$ can be defined as:

$$\text{ContributionToSuite}(t, r) = \sum_{i=1}^{n} \text{Contribution}(t, r),$$

where $R = \{r_1, r_2, \ldots, r_m\}$ is the test requirements and $\text{Contribution}(t, r) = \begin{cases} 1, & \text{if } t \text{ satisfies } r, \\ 0, & \text{if } r \text{ cannot satisfy } t, \end{cases}$

The number of test cases that satisfy the test requirement $r_i$ in (3) is positively related to the replaceability of $t$ with respect to $r_i$. A higher value of replaceability indicates that more test cases can be used to replace $t$ while maintaining the original test coverage. In contrast, a higher value of irreplaceability means that it is not easy to find other test cases to replace $t$. Consequently, we use (2) to evaluate the irreplaceability of test cases in the test suite. Based on this concept, we combine the execution cost of a test with (2) and evaluate the irreplaceability of test cases by:

$$\text{Irreplaceability}(t) = \sum_{i=1}^{n} \frac{\text{Contribution}(t, r_i)}{\text{Cost}(t)}.$$

As mentioned previously, $Coverage(t)$ and $\text{Ratio}(t)$ have been adopted to evaluate and choose the test cases. Similarly, $\text{Irreplaceability}(t)$ can be used to evaluate test cases and repeatedly choose the test with the maximum value until all of the test requirements are satisfied. Fig. I gives the pseudo code of ReduceWithIrreplaceability and Table III shows the steps associated with applying this algorithm to the example in Table I. In comparison to the sets produced by the other methods, it is evident that the ReduceWithIrreplaceability algorithm creates the representative set $RS^* = \{t_2, t_3\}$ with the lowest overall execution cost.

### Table I. An Example of a Test Suite and Test Requirements

<table>
<thead>
<tr>
<th>Test case</th>
<th>Requirements to be satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>r₁</td>
</tr>
<tr>
<td>t₂</td>
<td>7</td>
</tr>
<tr>
<td>t₃</td>
<td>3</td>
</tr>
<tr>
<td>t₄</td>
<td>4</td>
</tr>
</tbody>
</table>
A. in [5] and [16], we took these steps:

1. Select a set of test requirements from the test pool.
2. Evaluate the execution costs of the test cases with respect to the selected test requirements.
3. Select the test cases that can satisfy one or more unsatisfied test requirements.
4. Repeat Step 3 until all test requirements are satisfied.

B. Experimental Results

Table V furnishes the experimental results for the five algorithms. Here $RS_{Greedy}$, $RS_{GRE}$, $RS_{HGS}$, $RS_{ReduceWithRatio}$ and $RS_{ReduceWithIrreplaceability}$ denote the representative sets obtained by applying the selected algorithms, respectively. From this table, it is clear that all selected algorithms can significantly reduce the execution costs of test suites; the SCR values are greater than 87% in all cases. Among all of the algorithms, $RS_{ReduceWithIrreplaceability}$ achieves the best SCR scores for all of the subject programs. With the exception of schedule, the Greedy algorithm produces the worst representative set for the programs. Moreover, even though $ReduceWithRatio$ considers test execution cost, it performs worse than the traditional algorithms (i.e., GRE and HGS) for printtokens and printtokens2. On the whole, both $ReduceWithIrreplaceability$ and $ReduceWithRatio$ exhibit excellent cost reduction capabilities, but the SCR scores of $ReduceWithRatio$ are not as good as those of $ReduceWithIrreplaceability$. It should be noted that the SCR values shown in Table V are close to each other because the execution cost of the original generated test suite is quite high. However, the differences in the execution costs of the representative sets are considerable.

One threat to the validity of our empirical study includes the measurement of the execution time of each test case. Although we executed each test 1000 times and took the average of the execution costs, the measure for each test case may change if we performed the experiment in another environment. The second threat to validity is related to the fact that the experiments focused on the relatively small Siemens programs. While these programs are a good starting point for experimentation, our findings may not be relevant to larger programs and test suites.
V. CONCLUSION AND FUTURE WORK

Most existing test suite reduction algorithms attempt to minimize the size of a regression test suite. Since the ReduceWithRatio algorithm does not always perform in a satisfactory manner, this paper presents an algorithm that uses irreplaceability to evaluate the importance of tests and ultimately produce reduced test suites with a substantially decreased execution cost. The experimental results indicate that ReduceWithIrreplaceability is the best method for decreasing the cost of test suite execution, according to the SCR metric. In future work, we intend to enhance the cost reduction capabilities of GRE and HGS by incorporating test irreplaceability into their operation. Furthermore, we will conduct additional experiments with larger subject programs.

ACKNOWLEDGMENT

This research was sponsored by the National Science Council of Taiwan (Grant: NSC 100-2221-E-415-007-MY2).

REFERENCES