Test suite minimization with a greedy approach

Shrutakeerti Behura and Ajit Kumar Nayak
Department Computer Science and Engineering, SOA University Institute of Technical Education & Research, Bhubaneswar, India.

ABSTRACT

Regression testing leads to running many tests many times. Hence it requires more cost. The most robust and straightforward technique for regression testing is to accumulate all integration tests and rerun them whenever new components are integrated into the system. This requires developers to keep all tests up-to-date, to evolve them as the subsystem interfaces change and to add new integration tests as new services or new subsystems are added. As regression testing can become time consuming, test suite minimization (also known as Test Suite Reduction) technique is best suited to tackle it. In this paper we have explained the heuristic approach to solve this optimization problem.

Keywords
Regression Testing,
Test Suite Minimization,
Heuristics,
Greedy.

Introduction

Test Suite is a collection of written test cases and Regression testing requires large amounts of test cases to test any new or modified functionality within the program [1]. The components of a test suite is shown below.

![Components of a Test Suite](image)

Given. \[\{t_1, t_2, ..., t_m\}\] is test suite \(T\) from \(m\) test cases and \[\{r_1, r_2, ..., r_n\}\] is set of test requirements that must be satisfied in order to provide desirable coverage of the program entities and each subset \(\{T_1, T_2, ..., T_n\}\) from \(T\) are related to one of the requirements such that each test case \(t_j\) belonging to \(T_i\) satisfies \(r_i\) Problem. Find minimal test suite \(T'\) from \(T\) which satisfies all \(r_i\)'s covered by original suite \(T\).

Generally the problem of finding the minimal subset \(T', T\) belongs to \(T\) which satisfies all requirements of \(T\), is NP-complete [10], because we can reduce the minimum set-cover problem to the problem of test suite minimization in polynomial time.

Related Work

The classical greedy heuristic for solving the set-cover problem was presented by Chvatal [6]. The approach greedily selects the next set (test case) that maximizes the ratio of additional requirement coverage to cost, until no sets provide any additional requirement coverage. Another heuristic presented by Harrold et al. [3] (the HGS algorithm) greedily selects the next test case exercising the most additional requirements that are satisfied by the fewest number of tests. Chen and Lau [5] described two strategies for dividing a test suite into \(k\) smaller sub problems (sub suites) such that if optimal solutions can be found for each of the \(k\) sub problems, then these solutions can be combined to form an optimally reduced suite. However, these two dividing strategies cannot be applied to every suite. Agrawal [7] developed a technique using global dominator graphs to derive implications among testing requirements such that satisfying one requirement implies satisfying one or more of the other requirements. These implications can be used to achieve higher coverage with smaller suites by targeting those requirements implying the most coverage of the other requirements. Tallam and Gupta [1] developed another heuristic called Delayed-greedy that exploits both the implications among test cases and the implications among the requirements to remove the implied rows and columns in the table mapping test cases to the requirements covered by them. It delays the application of the greedy heuristic until after the table cannot be reduced any further and after the
essential tests are selected. Selecting a test case using the greedy heuristic and removing the corresponding row and the columns from the table exposes new implications among test cases and the implications among the requirements, which enables further reduction of the table. All the above heuristics to generate a minimal suite have polynomial time worst-case runtime complexity.

An Empirical Study

Given a test suite TS = \{t1,t2…tn\} consisting of the test case and the sequence of blocks of a tested program. \(R=\{r_1, r_2, \ldots, r_m\}\) we have a positive cost, \(c_j\) assigned to each test case measuring the amount of resources its execution needs. A positive weight, \(w_i\) is assigned to each requirement, which represents the relative importance of \(r_i\) with respect to the correct behavior of program or to the regression testing. For example, we can assign bigger weight to the recently modified correct behavior of program or to the regression testing. For

Modified Greedy Algorithm

The greedy algorithm takes the change in the coverage when choosing a test case to add to the reduced test-suite. We calculate the marginal coverage of each test case, i.e., the change in the coverage as a consequence of the change in reduced test-suite. We then compare it with the change in cost, and choose the test case that proves to be the best.

**Modified Greedy Algorithm (MGrA):**

Step1: Let \(T=\{\}\);

Step2: For each \(t_i \in TS-T\), calculate the increase in coverage and cost if it is added to \(T\):

\[\text{Cov}(t_i)=\text{Cov}(T\cup\{t_i\})-\text{Cov}(T),\]

\[\text{Cost}(t_i)=\text{Cost}(T\cup\{t_i\})-\text{Cost}(T)\]

Step3: Find a test cast \(t_i\) in \(TS-T\) for which \(\text{Cov}(t_i)/\text{Cost}(t_i)\) is minimal. If there are more, then choose the one with the lowest index. Let \(T=T\cup\{t_i\}\);

Step4: If \(\text{Cov}(T)\geq K\), then STOP, otherwise go to Step 2.

The above algorithm is being implemented using MATLAB. The resulted graph is shown in the following figure. The graph has been plotted by taking original test suite size along X axis and reduced suite size along Y axis. We have also implemented genetic algorithm (GA) to minimize the test suite. We have found that greedy is giving better result than that of genetic algorithm.

Another experiment have also been done to verify time taken by genetic algorithm and greedy algorithm. The result is shown below.

**Fig 3: Original test suite size Vs Time Elapsed**

Greedy algorithm is also showing better result in case of time elapsed during the reduction of test suite size.

**Conclusion**

The tests which have been performed to verify the performance of genetic algorithm and greedy algorithm further need to be minutely examined. A question also comes about fault detection effectiveness, which should be revealed and should also be same as original one. More experiments need to be done to verify the same. The above minimal cost problem is a single objective one. We are aiming at a multi objective problem which will consist of minimal cost problem and maximal fault detection effectiveness problem.

**Future work**

Future work includes performing the experiments on different sets of well known test suites as well as with more applications and larger test sets. We are also investigating a solution to the maximal fault detection effectiveness problem for more accuracy.

**Reference**


Test Cases”, Proceedings of the 14th International Symposium on Software