Theoretical study of test suite reduction techniques
Shrutakeerti Behura and Ambika Prasad Mishra
CSE Department, SOA University, I.T.E.R., Bhubaneswar, India.

ABSTRACT
As the software undergoes changes, new test cases are added to the existing one. In this way test suite size grows. Test suites should be maintained throughout. The Test-Suite Minimization technique aims at reducing the test suite using various techniques such as genetic algorithm, test case prioritization & selection technique based on some of the coverage criteria. This technique should let testers to compute an optimal minimal test suite that satisfies those criteria keeping an eye to maximize coverage and fault detection capability with minimal running time and setup cost.

© 2011 Elixir All rights reserved.

ARTICLE INFO
Article history:
Received: 13 April 2011;
Received in revised form: 21 May 2011;
Accepted: 28 May 2011;

Keywords
Test Suite Reduction,
Genetic algorithm,
Prioritization,
Redundancy.

Introduction
It is a common approach to build and maintain a regression test suite while developing and evolving a software system. Regression test suites are an important artifact of the software-development process and, just like other artifacts, must be maintained throughout the lifetime of a software product. In particular, testers often add to such suites test cases that exercise new behaviors or target newly-discovered faults. As a result, during maintenance, test suites tend to grow in size, to the point that they may become too large to be run in their entirety. In some scenario, the size of a test suite is not an issue. This is the case, for instance, when all test cases can be run quickly and in a completely automated way. In other scenarios, however, having too many test cases to run can make regression testing impractical. For example, for a test suite that requires human intervention (e.g., to check the outcome of the tests case or setup some machinery), executing all test cases could be prohibitively expensive. Another example is the case of cooperative environments where developers run automated regression test suites before committing their changes to a repository. In these cases, reducing the number of test cases to rerun may result in early availability of updated code and improve the overall efficiency of the development process.

Test Suite Reduction Problem
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 requirements that must be satisfied in order to provide desirable coverage of the program entities and each subset \( \{T_1, T_2, \ldots, T_n\} \) from \( T \) are related to one of \( r_i \) 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 \( r_i \) covered by original suite \( T \).

Literature Survey
The test-suite prioritization algorithm [1] created by James A. Jones et al., bases its contribution computation on MC/DC pairs (a pair of truth vectors) and utilizes an additional approach that recomputes the contribution of test cases after each test case is selected. However, instead of the test-case evaluation being based on the uniqueness of program-entity coverage, this algorithm uses a simpler evaluation based on additional MC/DC pairs covered.

The results of studies in this paper [3] are encouraging in that Xue-ying et al. have shown the potential for substantial test-suite size reduction and cost reduction, and genetic algorithm is more effective than Greedy algorithm both in size and cost reduction. Given a test suite \( T = \{t_1, t_2, \ldots, t_n\} \) consisting of the test case and the sequence of blocks of a tested program
\[ BS = \{b_1, b_2, \ldots, b_k\} \]
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 block, which represents the relative importance of \( b_i \) with respect to the correct behavior of program or to the regression testing. For example, we can assign bigger weight to the modified blocks or modification affected blocks of the new version program.

Let \( T \) be an arbitrary set of the test cases, \( T' \subseteq T \). The cost of this test set is defined as the sum of the costs of the test cases that belong to \( T \):
\[ c(T) = \sum_{T \subseteq T} C(t) \]

Let \( cov(T) \) denote the coverage of the test set \( T \),
\[ cov(T) = \sum_{T \subseteq T} wt.Cov(t) \]

Shin Yoo et al. shows that Pareto efficient multiobjective optimization to the problem of test suite minimization [4] described the benefits of Pareto efficient multi-objective optimization, and presented an empirical study that investigated the relative effectiveness of two algorithms for Pareto efficient multiobjective test suite minimization. Primary contribution of this paper is as follows:
1. The paper introduces a multi-objective formulation of the regression test suite minimization problem and instantiates this with two objectives: A two-objective formulation that caters for coverage and cost and a three-objective formulation that caters for coverage, cost and fault-history. The formulations facilitate a theoretical treatment of the optimality of the greedy algorithm and make it possible to establish a relationship between the multi-objective problems of test case prioritisation and test suite minimisation.
2. The paper presents two algorithms for solving the two and three objective instances of the test suite minimisation problem: a re-formulation of the single-objective greedy algorithm, and a hybrid variant of NSGA-II of Deb et al. (1997), which we call HNSGA-II. The hybrid nature of HNSGA-II is based on the known fact that the greedy algorithm produces a good approximation to the set-cover problem, which forms the basis of the test suite minimisation problem.

3. The paper presents the results for these algorithms, when applied to the two-objective version of the problem using, as subjects, five non-trivial real world programs from Software architecture Infrastructure Repository, SIR (Do et al., 2005). The results confirm the theoretical analysis, revealing cases where the search based algorithms out-perform the greedy approach. More importantly, the results show that the hybrid approach is capable of filling in large gaps in the Pareto fronts approximated by the greedy algorithm.

4. The paper also presents results from an empirical study of the algorithms applied to the three-objective formulation of the problem. These results also show that the hybrid approaches can out-perform the greedy approach.

A. Askarunisa et al. implemented the greedy approach [5] that selects the next set (test case) that maximizes the ratio of additional requirement coverage to cost, until no sets provide any additional requirement coverage.

The technique presented by Praveen Ranjan Srivastava et al. implemented a new regression test suite prioritization algorithm [2][9] that prioritizes the test cases with the goal of maximizing the number of faults that are likely to be found during the constrained execution. The algorithm is as follows. Input: Test suite T, number of faults detected by a test case f, and cost to run each test case Tcost.

Output: Prioritized Test suite T'.

1: begin
2: set T' empty
3: for each test case t ∈ T do
4: calculate average faults found per minute as f/Tcost
5: end for
6: sort T in descending order based on the on the value of each test case
7: let T' be T
8: end

Lilly Raamesh sows the knowledge mining system [10] reduces the size of test suite by comparing with original test cases. Data mining sits at the interface between statistics, computer science, artificial intelligence, machine learning, database management and data visualization. It is the process of identifying valid, novel, potentially useful, and ultimately comprehensible knowledge from data that is used to help by crucial decision-making. The search for an optimal solution in the test case generation problem has a great computational cost and for this reason these techniques try to obtain near optimal solutions.

Conclusions

Due to the computational complexity of multi-objective minimization, however, most existing techniques target a much simpler version of the problem: generating a test suite that achieves the same coverage as the original test suite with the minimal number of test cases. Previous research has shown, for instance, that the error-revealing power of a minimized test suite can be considerably less than that of the original test suite. In this hybridized approach we will club genetic algorithm with test case prioritization and removing redundant test cases to reduce the number of test cases. But we have to check error detection capability. The whole work will be carried out by using MATLAB (GA tool), The challenge has to be met to minimize test suite and maximize error detection capability.

References


