Available online at www.elixirpublishers.com (Elixir International Journal)

Advanced Engineering Informatics

Elixir Adv. Engg. Info. 40 (2011) 5183-5185

ISSN: 2229-712X

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.

ARTICLE INFO

Article history:

Received: 25 August 2011;

Received in revised form:

17 October 2011;

Accepted: 27 October 2011;

Keywords

Regression Testing,

Test Suite Minimization,

Heuristics,

Greedy.

ABSTRACT

Regression testing leads to running many tests many times. Hence it requires more cost. The most robust and straight forward 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 changes 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.

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.

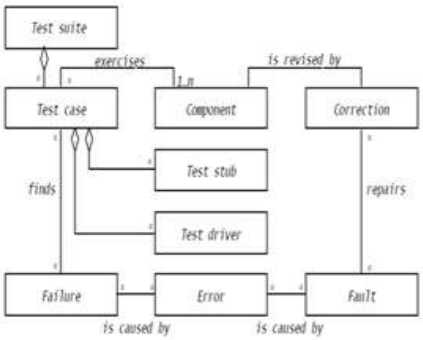

Fig1. Components of a Test Suite

Re-running all existing test cases together with the new ones is often costly and even infeasible due to time and resource constraints. To address this problem, the researchers proposed techniques to optimize regression testing [2], [3], [4], [5], [6], [7], [8]. Re-running test cases that do not exercise any changed or affected parts of the program makes extra cost and gives no benefit. An effective technique is to permanently discard such redundant or obsolete test cases and retain the most effective ones to reduce the excessive cost of regression testing [6]. Such technique attempts to find a minimal subset of test cases which satisfy all the testing requirements as the original set does [9]. This 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. This technique is commonly known as test suite reduction or test suite minimization in the literature and the resulting suite is called representative set [3].

Test suite reduction problem

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

Given. {t1, t2,…, tm} is test suite T from m test cases and {r1, r2,…, rrn} is set of test requirements that must be satisfied in order to provide desirable coverage of the program entities and each subsets {T1, T2,…, Tn} from T are related to one of requirements such that each test case tj belonging to Ti satisfies ri Problem. Find minimal test suite T' from T which satisfies all ris 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

Tele:

E-mail addresses: shrutakeerti.behura@gmail.com

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={r1, r2,…, rrn} , we have a positive cost, cj assigned to each test case measuring the amount of resources its execution needs. A positive weight, wi is assigned to each requirement, which represents the relative importance of ri with respect to the correct behavior of program or to the regression testing. For example, we can assign bigger weight to the recently modified requirement.

Let T be an arbitrary set of the test cases, T⊂TS. The cost of this test set is defined as the sum of the costs of the test cases that belong to T: c(T)=Σt_T C(t).

Let cov(T) denote the coverage of the test set T,

cov(T)=Σt_T wt.Cov(t).

Here the lower bound (K) is the coverage of the original test-suite. In fact, the coverage of the reduced test suite is impossible to be larger than K.

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 ti ∈TS-T, calculate the increase in coverage and cost if it is added to T:

Cov(ti)= Cov(T∪ { ti })- Cov(T),

Cost(ti)= Cost(T∪ { ti })- Cost(T)

Step3: Find a test cast ti in TS-T for whic h Cov(ti)/ Ccost(ti) is minimal. If there are more, then choose the one with the lowest index. Let T=T ∪ { ti };

Step4: If Cov(T)≥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.

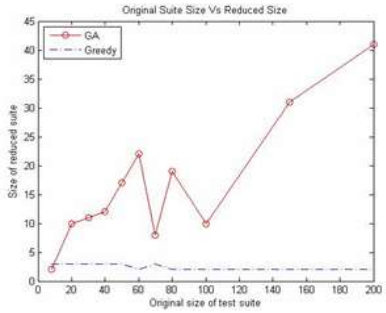

Fig 2: Comparison between Genetic Algorithm and Greedy implementation

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

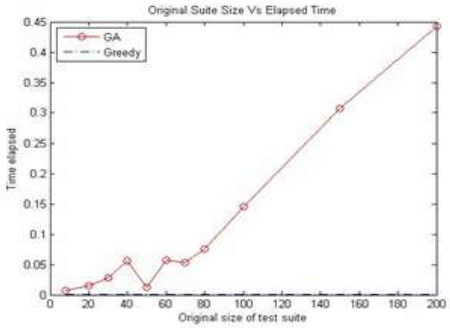

Fig 3: Original test suite size Vs Time Elapsed

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

[1]G. Rothermel, M.J. Harrold, J. von Ronne, C. Hong, "Empirical Studies of Test-Suite Reduction",Journal of Software Testing, Verification, and Reliability, 12(4), 2002, pp. 219-249.

[2]G.Rothermel, M.J. Harrold, J. Ostrin, C. Hong, "An Empirical Study of the Effects of Minimization onthe Fault Detection Capabilities of Test Suites", Proceedings of the International Conference on Software Maintenance, IEEE Computer Society, 1998.

[3]M.J. Harrold, R. Gupta, M.L. Soffa, "A Methodology for Controlling the Size of a Test Suite", ACM Transactions on Software Engineering Methodologies 2, 1993, pp. 270-285.

[4]T.Y. Chen, M. F. Lau, "Heuristics toward the Optimization of the Size of a Test Suite" Proc. 3rd Int'l Conf. on Softw. Quality Management. Vol. 2, Seville, Spain, April 1995, pp. 415-424.

[5]J.A. Jones, M.J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", IEEE Trans. Softw. Eng. 29, 2003, pp. 195-209.

[6]S. McMaster, A. Memon, "Call-Stack Coverage for GUI Test Suite Reduction", IEEE Trans. Softw.Eng. 34, 2008, pp. 99-115.

[7]S.Tallam, N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization",Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. ACM, Lisbon, Portugal, 2005.

[8]D. Leon, A. Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing

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

[9]G. Rothermel and M.J. Harrold, A Safe, Efficient Regression Test Selection Techinique. ACM Trans. Software Eng. And Methods, vol. 6, no. 2, pp. 173-210, Apr. 1997.

[10]G. Rothermel, M.J. Harrold, J. Ostria, and C. Hong, An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites. Proc. Int'l Conf. Software Maintenance, PP. 34-43, Nov. 1998.