The Journal of Systems and Software 85 (2012) 16–27

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Test coverage optimization for large code problems

Ying-Dar Lin[a,*], Chi-Heng Chou[a], Yuan-Cheng Lai[b], Tse-Yau Huang[c], Simon Chung[d], Jui-Tsun Hung[a], Frank C. Lin[e]

[a] Department of Computer Science, National Chiao Tung University, Taiwan
[b] Department of Information Management, National Taiwan University of Science and Technology, Taiwan
[c] Department of Communications Engineering, National Chiao Tung University, Taiwan
[d] Cisco Systems Inc., USA
[e] San Jose State University, USA

ARTICLE INFO

Article history:
Received 1 May 2010
Received in revised form 12 May 2011
Accepted 12 May 2011
Available online 20 May 2011

Keywords:
Regression testing
Test case selection
Test coverage
Test intensity
Software maintenance

ABSTRACT

Software developers frequently conduct regression testing on a series of major, minor, or bug-fix software or firmware releases. However, retesting all test cases for each release is time-consuming. For example, it takes about 36 test-bed-days to thoroughly exercise a test suite made up of 2320 test cases for the MPLS testing area that contains 57,758 functions in Cisco IOS. The cost is infeasible for a series of regression testing on the MPLS area. Thus, the test suite needs to be reduced intelligently, not just randomly, and its fault detection capability must be kept as much as possible. The mode of *safe* regression test selection approach is adopted for seeking a subset of modification-traversing test cases to substitute for fault-revealing test cases. The algorithms, CW-NumMin, CW-CostMin, and CW-CostCov-B, apply the *safe-mode* approach in selecting test cases for achieving full-modified function coverage. It is assumed that modified functions are *fault-prone*, and the fault distribution of the testing area is *Pareto-like*. Moreover, we also assume that once a subject program is getting more mature, its fault concentration will become stronger. Only function coverage criterion is adopted because of the scalability of a software system with large code. The metrics of *test's function reachability* and *function's test intensity* are defined in this study for algorithms. Both CW-CovMax and CW-CostMin algorithms are not safe-mode, but the approaches they use still attempt to obtain a test suite with a maximal amount of function coverage under certain constraints, i.e. the effective-confidence level and time restriction. We conclude that the most effective algorithm in this study can significantly reduce the cost (time) of regression testing on the MPLS testing area to 1.10%, on the average. Approaches proposed here can be effectively and efficiently applied to the regression testing on bug-fix releases of a software system with large code, especially to the releases having very few modified functions with low test intensities.

1. Introduction

Over the lifetime of a large software product, the number of test cases could drastically increase as new versions of software are released. Because the cost of repeatedly retesting all test cases may be too high, software testers tend to remove redundant or trivial test cases to construct a reduced test suite for regression testing at a reasonable cost. Generally, test cases are selected under certain criteria such as coverage criteria, resources constraints, or fault detection capability. In literature, researchers have developed an array of selection algorithms for regression testing, based on a variety of models of code coverage or fault detection capability. However, many algorithms demand a long execution time, and a huge number of test cases exist for a large body of code.

Bearing in mind factors of scalability and practicability, code coverage information in this study is investigated at the *function-level* granularity, rather than the *statement-level* one, e.g. condition/decision or branches. In particular, *test cases* and *functions* form an *interlaced net*; therefore, test case selection can depend on function's attributes, and vice versa. The interlaced net correlation leads to two metrics – function's *test intensity* and test's *function reachability*. The former indicates the percentage of test cases covering a function, while the latter denotes the percentage of functions reached by a test case.

Leung and White (1989) indicated that the test suite reduction problem has two subproblems – *test selection problem* and *test plan update problem*. Solutions to the former problem focus on how to select test cases to construct a reduced test suite, which can still effectively reveal faults. Yet solutions to the latter problem must cope with the management of test plans for a software system that

* Corresponding author.
E-mail addresses: ydlin@cs.nctu.edu.tw (Y.-D. Lin), payton.chou@gmail.com (C.-H. Chou), laiyc@cs.ntust.edu.tw (Y.-C. Lai), jason.hung.sb@gmail.com (J.-T. Hung).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.05.021

had experienced several releases of modifications. Cisco did not provide test plan updates information; hence, only test selection problems can be dealt with when the automated regression test system is applied.

Recently Yoo and Harman (2010) showed a survey of regression testing on three problems – *test suite minimization, regression test-case selection (RTS), and test case prioritization*. All share a common thread of optimization when a test suite reduction is exercised from an existing pool of test cases. In this survey, regression testing is described as "*Regression testing is performed when changes are made to existing software; the purpose of regression testing is to provide confidence that the newly introduced changes do not obstruct the behaviors of the existing, unchanged part of the software*." These problems are restated as follows.

1.1. Test suite minimization problem

Given: A test suite of test cases where a set of testing requirements must be satisfied to provide the desired test coverage of the program, and subsets of the test suite where each subset is associated with *one* of the testing requirements such that any test case in the subset satisfies the testing requirement.

Problem: Find a minimal representative subset of the test suite that satisfies all the testing *requirements*.

Test suite minimization problem is well known as the minimal hitting-set problem, or the minimal set-cover problem (Garey and Johnson, 1979). Approaches to this problem typically emphasize on how to identify redundant test cases to be removed, so that a minimal test suite can be constructed. Because this problem is NP-complete, heuristics methods (Wong et al., 1998; Leung and White, 1989) are encouraged. In literature, the greedy methods, (Harrold et al., 1993; Jeffrey and Gupta, 2005, 2007; Chen and Lau, 1998a), genetic methods (Whitten, 1998; Ma et al., 2005; Mansour and El-Fakih, 1999), and linear programming methods (Black et al., 2004) are commonly applied. In addition, the hitting set algorithm (Harrold et al., 1993) categorizes test cases according to the degree of "essentialness," and selects test cases in order from the most "essential" to the least "essential." The heuristic G/GE/GRE algorithms (Chen and Lau, 1998a) are developed depending on the essential, the 1-to-1 redundant, and the greedy strategies (G: greedy strategy, E: essential strategy, and R: 1-to-1 redundant strategy).

Other approaches include modeling the cost-benefits for regression testing (Malishevsky et al., 2002), measuring the impact of test case reduction on fault detection capability (Wong et al., 1998, 1999; Rothermel et al., 1998, 2002), and analyzing fault detection capability, especially with the branch coverage technique (Harrold et al., 1993; Jeffrey and Gupta, 2005, 2007). The performance of several test suite reduction techniques are examined by experiments or simulations (Zhong et al., 2006; Chen and Lau, 1998b). Because the algorithms in (Chen and Lau, 1998a,b) do not exactly meet our requirements, G algorithms is revised and applied to test case selection, as shown in section 3.

1.2. Test case selection problem

Given: A subject program with a corresponding test suite, and a modified version of this subject program.

Problem: Find a reduced test suite for the modified version of the subject program.

In literature (Yoo and Harman, 2010), approaches to test case selection problems (Rothermel and Harrold, 1996) and to test suite minimization problems are different in how they use *changes,* or modified code, while selecting test cases. Approaches to test suite minimization problems are based on a single release of a subject program while those to regression test case selection problem are based on changes between a previous and the current version of a subject program. Hence, the approaches to test case selection problems are modification-aware (Yoo and Harman, 2010) for emphasizing the coverage of *code changes*. Moreover, Rothermel and Harrold introduced the concept of *modification-revealing* test case in (Rothermel and Harrold, 1994a) and assumed that identifying *fault-revealing* test cases for new release program is possible through the modification-revealing test cases between the original and new release of a subject program. Rothermel also adopted a weaker criterion that selects all *the modification-traversing* test cases. A test case is modification-traversing if and only if it executes new or modified code in the new release of a program, or it executes former code yet deleted in the new release. This led to a premise that selecting a subset of modification-traversing test cases and remove test cases that are guaranteed not to reveal faults in the new release of a program is possible. Thus, an approach to *safe* regression test selection problem was introduced in Rothermel and Harrold (1997), though it is still not safe for detecting all possible faults, but providing a safe sense of always selecting modification-traversing test cases into a reduced test suite. In Section 3, an algorithm that selects test cases for a test suite with full-modified function coverage is of *safe-mode* and considered a safe regression test selection. For instance, CW-NumMin, CW-CostMin, and CW-CostCov-B algorithms are of safe mode while CW-CovMax and CW-CostMin-C algorithms are not because these two algorithms do not intend to achieve full modified function coverage.

Other approaches to test case selection problems employ distinct techniques such as data flow analysis (Harrold and Soffa, 1989), the graph-walk approach (Rothermel and Harrold, 1993, 1997, 1994b), the modification-based technique (Chen et al., 1994), the firewall approach (Leung and White, 1990; White and Leung, 1992; Zheng et al., 2007) and so on. Strengths and weaknesses of these approaches can be found in (Yoo and Harman, 2010).

1.3. Test case prioritization problem

Given: A test suite and a set of permutations of the test suite.

Problem: Find a test suite where test cases are exercised in order, and a specified maximal gain is achieved under certain constraints.

The approach to this problem was first proposed by Wong et al. (1998), and extended by Harrold (1999). Empirical Studies can be found in Rothermel et al. (1999, 2001). The CW-CovMax and CW-CostMin-C algorithms in Section 3 are the variants of approaches to test case prioritization problems, except that these algorithms merely emphasize on selecting test cases, instead of exercising test cases in order.

In this work, six algorithms are implemented by a database-driven method to reduce the size of test suites, and experiments are conducted by an automated production system for regression testing on the MPLS test area of Cisco IOS. The automated system provides information on code coverage traces and execution time for each test case, while a source control system imports a history of code modification for analyzing faults detected from the newly modified code.

Faults detected in regression testing are real, compared to the faults that are hand-seeded in small subject programs when examining the test suite minimization problem. They are probably fixed if detected in a series of releases of a subject program during regression testing. Thus, unless retesting all test cases, we cannot acquire the total number of faults that can be detected. Even nobody can know the total number of real faults that exist in a subject program. Therefore, it is infeasible to calculate the fault detection effectiveness for regression testing on an industrial software system.

PDF-SA algorithm applies the function's test intensity as a metric in reducing the function space by removing infrastructure func-