The Journal of Systems and Software 105 (2015) 91–106

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Input-based adaptive randomized test case prioritization: A local beam search approach ☆

Bo Jiang [a], W.K. Chan [b,*]

[a] School of Computer Science and Engineering, Beihang University, Beijing, China
[b] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong

ARTICLE INFO

Article history:
Received 19 January 2014
Revised 1 March 2015
Accepted 22 March 2015
Available online 30 March 2015

Keywords:
Regression testing
Adaptive test case prioritization
Randomized algorithm

ABSTRACT

Test case prioritization assigns the execution priorities of the test cases in a given test suite. Many existing test case prioritization techniques assume the full-fledged availability of code coverage data, fault history, or test specification, which are seldom well-maintained in real-world software development projects. This paper proposes a novel family of input-based local-beam-search adaptive-randomized techniques. They make adaptive tree-based randomized explorations with a randomized candidate test set strategy to even out the search space explorations among the branches of the exploration trees constructed by the test inputs in the test suite. We report a validation experiment on a suite of four medium-size benchmarks. The results show that our techniques achieve either higher APFD values than or the same mean APFD values as the existing code-coverage-based greedy or search-based prioritization techniques, including Genetic, Greedy and ART, in both our controlled experiment and case study. Our techniques are also significantly more efficient than the Genetic and Greedy, but are less efficient than ART.

1. Introduction

Regression testing (Yoo and Harman, 2012) is a widely-practiced activity in real-world software development projects (Onoma et al., 1998), in which a better testing infrastructure has a potential to recover the economic loss resulting from software failures by one third (Tassey, 2002). During a session of a regression test, a changed program P is executed over a regression test suite T. Many companies executed the whole test suite to ensure the quality of their software (Onoma et al., 1998). Moreover, each nightly build of many open-source software projects such as *MySQL* (MySQL, 2013) and *FireFox* (FireFox, 2013) always apply the whole test suite to verify the version built.

If the time spent to complete the execution of a program over an individual test case is non-trivial, the time cost to execute the whole test suite T may be large (Jiang et al., 2011). For instance, profiling an execution trace of a C/C++ program at the memory access level using a *pintool* may easily incur tens to one hundred fold of slowdown (Luk et al., 2005). On the other hand, programmers may want to know the test results as early as possible at low cost.

Test case prioritization (Elbaum et al., 2002; Wong et al., 1997) is a *safe* aspect of regression testing. In essence, test case prioritization reorders the test cases in a test suite T and does not discard any test case in T for execution toward a chosen testing goal (denoted by G).

A vast majority of existing test case prioritization research studies (Yoo and Harman, 2012) propose to collect data from the executions of a previous version (denoted by Q) of P over a test suite T_{old} to guide the prioritization on T. For ease of presentation, we denote the set of program execution traces of Q over T_{old} by $Q(T_{old})$ and that of P over T by $P(T)$.

Numerous types of such data (such as the fault history (Kim and Porter, 2002), the change history (Elbaum et al., 2004), and the execution profiles (Elbaum et al., 2002)) obtained from these executions of Q have been empirically evaluated in diverse contexts with respect to the differences in their effects on regression testing results towards the selected goal G of regression testing techniques. For instance, a vast majority of empirical studies on test case prioritization validate on how quickly the permutations of T generated by such test case prioritization techniques detect faults in P by assuming that $T = T_{old}$. A recent trend is to replace the rate of fault detection by the rate of program element coverage (Li et al., 2007) or to incorporate the results of change impact analysis (Li et al., 2013) in their problem or solution

☆ This research is supported in part by the Early Career Scheme of Research Grants Council of Hong Kong SAR (project nos. 111313 and 11201114), the National Natural Science Foundation of China (project no. 61202077).

* Corresponding author. Tel.: +852 3442 9684.
E-mail addresses: jiangbo@buaa.edu.cn (B. Jiang), wkchan@cityu.edu.hk (W.K. Chan).

http://dx.doi.org/10.1016/j.jss.2015.03.066

formulations. Still, the essential assumption of inferring T based on T_{old} remains unchanged.

In this paper, we propose a new family of novel input-based randomized local beam search (LBS) techniques for test case prioritization. This family of techniques targets to be applied in the general (more practical) scenario, where T may be different from T_{old} and Q may be different from P without taking any approximation (i.e., not assuming either T inferable from T_{old} or Q and P similar). Because both T_{old} and Q are irrelevant to this family of techniques, these LBS techniques can be applied in both regression testing and functional testing. In this way, practitioners need not care about whether a testing technique is applicable to functional testing scenarios only or to regression testing scenarios only, or both.

Given a test suite T, each LBS technique starts with a set of k partial solutions, each being a sequence of single test case taken from T. For each partial solution S, it randomly selects a number of test cases from $T \setminus S$ to construct a candidate set C and evaluates each extended partial solution S^{t}, where $t \in C$, according to a chosen distance measure. It marks the overall best k extended partial and randomized solutions as the current set X of partial solutions. It then goes into the next iteration to extend each partial solution in the current set X in the same manner until all test cases in T have been included in each partial solution X. It addresses the search space exploration cost problem by controlling the number of branches in the exploration tree in each round to a small number. Suppose that at each round, both the size of the candidate set and the number of branches to be explored by an LBS technique are k, and the number of distance comparisons between test cases in each node of the tree being explored is capped to be m, and then there are at most $mk^2|T|$ comparisons.

We have validated our LBS techniques on four medium-sized UNIX utility programs in a controlled experiment setting to evaluate their overall performance. We have further performed a case study on the comparison of our LBS search algorithm to the algorithms of Greedy (Elbaum et al., 2002), ART (Jiang et al., 2009), and Genetic (Li et al., 2007) by encoding test cases using input information and using the even-spread of test cases as the guidance heuristic to determine whether the performance of our techniques is merely attributed to the LBS algorithm. In both validations, we measured their effectiveness in terms of the average rate of fault detection (i.e., APFD (Elbaum et al., 2002)) and the time spent in generating a resultant prioritized test suite.

The empirical results from both the controlled experiment and the case study show that LBS achieves either higher mean APFD values than or similar mean APFD values as Greedy, ART, and GA. LBS also is *significantly* more efficient than GA but less efficient than ART at the 5% significance level. The result further shows that the effectiveness of LBS is not much affected by different parameter values needed to initialize the LBS algorithm. In the case study, we have the following observations: (1) the effectiveness of the studied LBS techniques was mainly contributed by our LBS algorithm, and (2) the use of input information for test case encoding and our heuristic also contributed to the significant reduction of the test case prioritization cost.

This work significantly extends its preliminary version (Jiang and Chan, 2013): (1) It generalizes the family of LBS techniques by presenting five more new techniques and evaluates the family against more existing techniques for benchmarking. (2) It reports a new experiment that investigates the impact of candidate set size and beam width on the effectiveness of the family. (3) It presents a new case study on comparing this family with several adapted classical search-based test case prioritization algorithms (Greedy, ART, and Genetic).

The main contribution of the paper together with its preliminary version (Jiang and Chan, 2013) is twofold. (1) This paper is the *first* work that presents a family of novel input-based randomized test case prioritization techniques. (2) It presents the *first* series of experiments to validate input-based search-based test case prioritization techniques.

We organize the rest of paper as follows: we review the preliminaries of this work in Section 2. Then, we describe our family of LBS techniques with their design rationales in Section 3. After that, we present validation experiments in Section 4 and Section 5. Section 6 discusses other issues relevant to our LBS techniques. Finally, we present the related work followed by the conclusion of the paper in Section 7 and Section 8, respectively.

2. Preliminaries

2.1. Problem formulation

Elbaum et al. (2002) described the test case prioritization problem as follows:

Given: T, a test suite; PT, the set of permutations of T; g, a goal function from PT to real numbers.

Problem: To find $T' \in PT$ such that $\forall T'' \in PT, g(T') \geq g(T'')$.

In this problem formulation, PT represents a set of all possible prioritizations (orderings) of T and g is a goal function that calculates an award value for that ordering.

For a test suite containing N test cases, the size $|PT|$ is $N!$, which is intractable if N is large. In practice, the set PT in the universal quantification under the above problem formation is replaced by an enumerated subset of PT.

Moreover, a goal g, such as the rate of code coverage (Li et al., 2007), can be measurable before the execution of P over the prioritized test suite T'. Such a goal can be used by a search-based optimization technique for test case prioritization.

There are however other types of goals, such as the rate of fault detection (Elbaum et al., 2002), which cannot be measured directly before the execution of the test cases. A recent attempt is to use a heuristic (e.g., code coverage, even spreading of test cases) to make an approximation. Our techniques try to spread the test cases in T as evenly as possible within the input domain in each iteration using a randomized local beam search approach.

2.2. Critical review on assumptions of test case prioritization techniques

In this section, we revisit the assumptions made in the typical test case prioritization research work.

In general, T may be different from T_{old} and P may be different from Q. The dataset or execution profile of $Q(T_{old})$ is also unlikely to be the same as these of $Q(T)$, $P(T_{old})$, or $P(T)$. Moreover, if *either* a test case reduction/selection technique (Do et al., 2008; Yoo and Harman, 2012) *or* an impact analysis technique (Li et al., 2013) has been applied on T_{old} to construct a proper subset $T_{old'}$ of T_{old} and the test cases in $T_{old} \setminus T_{old'}$ have not been executed by Q, we have $Q(T_{old'}) \subset Q(T_{old})$. In this connection, if a test case prioritization technique relies on $Q(T_{old'})$ in prioritizing test cases in T, it is a threat.

We further categorize our observations on the limitations due to such assumptions into five classes:

First, assuming the historic data of T_{old} always available is restrictive. For instances, the execution profile data are seldom maintained in the repositories of real-world software development projects such as *MySQL* (MySQL, 2013), *Eclipse* (Eclipse, 2013), and *FireFox* (FireFox, 2013). In many projects, such as numerous Android applications (e.g., *Foursquared* (Foursquared, 2012)) available in Google Code (Google Code, 2013), their bug repositories only keep few bug reports.

One way to alleviate this problem is to run T on an older version Q. However, the correctness criteria (e.g., the assertion statements in JUnit test cases) may require manual determination. Both the executions of the test cases and the determination of their correctness lead to non-trivial overheads.

Collecting code coverage data requires profiling program executions, which may be impractical in some industrial environments. For instance, in safety-critical software like avionics control (where enumerators cannot support the execution of the whole test suite),