Information and Software Technology 80 (2016) 57–72

Contents lists available at ScienceDirect

Information and Software Technology

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

The optimal testing order in the presence of switching cost

Huayao Wu[a], Changhai Nie[a,*], Fei-Ching Kuo[b]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[b] Faculty of Information and Communication Technologies, Swinburne University of Technology, VIC, Australia

ARTICLE INFO

Article history:
Received 23 September 2015
Revised 12 July 2016
Accepted 18 August 2016
Available online 20 August 2016

Keywords:
Test suite prioritization
Combinatorial testing
Switching cost
Combination coverage
Multi-objective optimization

ABSTRACT

Context: Test suite prioritization is a problem of deciding the order of executing test cases to reach the desirable outcome. Many cost-cognisant prioritization approaches decide the order based on the cost of test execution; but few based on the cost of switching test cases. The latter known as switching cost is the effort of re-configuring the environment for running subsequent test cases. Our previous studies show that switching cost can affect the efficiency of testing.

Objective: In this paper, we aim to identify the optimal testing order that can detect interaction triggered faults earlier in the presence switching cost.

Method: We presented a distance based metric to measure the switching cost between test cases. As reducing the switching cost can make the whole test suite run faster and thus achieve full combination coverage earlier, single-objective algorithms were used to minimize the total switching cost. Besides, when determining the next test case to run, there is a trade-off between high combination coverage and low switching cost. Hence, hybrid and multi-objective algorithms were used to achieve a better balance. In order to evaluate different algorithms, we conducted a series of experiments covering 400 different testing scenarios. We also conducted an empirical study with six real world applications.

Results: The heuristic solver for the travelling salesman problem is the best algorithm to minimize the switching cost. It can detect faults earlier than the order with high rate of combination coverage. But in order to further reduce the effort to detect the first fault, the hybrid and multi-objective algorithms are the best methods.

Conclusion: Prioritization based on switching cost can speed up the fault detection to some extent, but prioritization based on both combination coverage and testing cost can deliver the optimal testing order.

1. Introduction

In software testing, time and resources are two key constraints that drive testers' decisions in testing practices. Test suite prioritization is about scheduling test cases to maximize test benefits under these constraints. One example of benefits is the ability of early fault detection [1]. Given a test suite to be executed, as the various testing orders will make a difference on the effectiveness of testing, many prioritization techniques have been widely studied.

To do test suite prioritization, generally a surrogate metric should be determined before the prioritization algorithms can be applied. Currently, researchers have proposed many metrics according to different testing goals, such as coverage based metrics [2–7], human based metrics [8,9], and requirement based metrics [10,11]. Testing cost is also an important metric in prioritization. Elbaum et al. [12] have shown that the varying execution cost of test cases will affect how quickly the fault can be detected. On the other hand, Walcott et al. [13] have proposed a time-aware prioritization method to detect as more faults as possible when the allowable execution time is known in advance. However, as the testing cost usually consists of the execution cost of each test case and the switching cost between adjacent test cases, most of current works all lack a consideration on the latter one.

The switching cost is crucial in practice if it cannot be ignored. Supposing that the execution cost of each test case is fixed, reordering them will not affect the total cost of execution. Whereas, in different testing orders, the time and effort to change parameter settings when switching test cases will affect the efficacy of testing. For example, Srikanth et al. [10] have reported the impact of switching cost on a legacy product. In their study, the software needs to be tested under different configurations, and it will

* Corresponding author. Fax: +862589680915.
E-mail address: changhainie@nju.edu.cn (C. Nie).

http://dx.doi.org/10.1016/j.infsof.2016.08.006
0950-5849/© 2016 Elsevier B.V. All rights reserved.

spend several days to change one configuration into another. In addition, the switching cost may relate to the setting of software compilation parameters [14]. Apparently, if this cost is high and occurs frequently, a lot of time will be spent on switching test cases instead of executing them. Furthermore, modern software systems, especially highly configurable software systems, usually consist of many features or components to be configured. When there exists reasonable switching cost to change some features or components, there is a need to make costly switches as infrequent as possible to reduce the testing cost.

To measure the switching cost between test cases, in this work we consider the context of combinatorial testing (CT). CT is a well-known black box testing method to detect the faults triggered by the interactions of parameters. The test suite of CT is generated by some sophisticated algorithms to cover all the required combinations with as small test suite as possible [15], and the test suite of CT is often reordered by the combination coverage [7,16–18]. When the switching cost is considered in CT, Kimoto et al. [19] and Demiroz et al. [14] have proposed test suite generation methods with two goals in mind: small size and low switching cost. However, according to our previous findings [20], their methods are not competitive in both criteria. And, generating a small test suite is always a key challenge in CT. Only focusing on prioritization can take the advantages of existing generation algorithms. Besides, pure prioritization can also make use of existing test suites, which may be designed to satisfy various needs. Hence, instead of minimizing switching cost in test suite generation, we focus on test suite prioritization for an existing test suite.

Our previous work [20] had proposed the problem of switching cost based prioritization. The switching cost between test cases was measured by a distance based metric, and we found that minimizing switching cost can make the whole test suite run faster and so achieve full combination coverage earlier. This observation motivated the single objective optimization, which aimed to identify a testing order with as low switching cost as possible for a given test suite. However, it was shown that the greedy algorithm is fast but is easy to trap into local optimum, and the dynamic programming can deliver the optimal order but it is only available for very small test suites [20]. In order to explore more effective algorithms for minimizing switching cost, in this work we further applied the following two algorithms: Genetic Algorithm (GA) [21], which is inspired by the successful applications of Search Based Software Engineering (SBSE) [22] on prioritization [23]; and Lin–Kernighan heuristic (LKH) solver [24,25] for travelling salesman problem (TSP) [26], which can be used because the problem of switching cost based prioritization can be reduced to TSP.

However, the single objective optimization is often limited in practice, because software testing is often driven by multiple imperatives [27–29]. In this work, when determining the next case to run, we found that there exists a trade-off between high combination coverage and low switching cost. And, the execution cost of each test case is also crucial in prioritization. So if the aim is to cover more combinations with less time, it could be insufficient to only consider one factor and multi-objective optimization is thus more desirable. Therefore, we reformulated the prioritization problem into a multi-objective version, and applied Hybrid and NSGA-II [30] to balance the two goals: rate of combination coverage and testing cost (execution cost + switching cost). The Hybrid algorithm is based on greedy construction where multiple objectives are combined into a single one, while NSGA-II is a well known multi-objective evolutionary algorithm (MOEA) to simultaneously optimize different objectives.

Moreover, as the effectiveness of prioritization is often dependent on particular testing scenarios, we need to have a better understanding of when and how we should reorder a test suite in the presence of switching cost. To achieve this goal, we conducted experiments to evaluate both single and multi objective algorithms. The experimental subjects are 400 randomly sampled testing scenarios with different distributions of testing cost, and the different testing orders are compared in terms of optimization quality and the ability of early fault detection. Besides, we also conducted an empirical study to evaluate different prioritization algorithms under five mobile applications and one desktop application with realistic faults. As there has been little previous work focusing on the relationships between combination coverage and testing cost, the obtained results can bring new insights about the practical choice of prioritization metrics.

In summary, the main contributions of this work are as follows:

1. Switching cost is an important factor in test suite prioritization. To explore more effective algorithms to minimize switching cost, we applied GA and LKH solver.
2. In order to further take combination coverage and execution cost into consideration, we reformulated the prioritization problem into a multi-objective version and applied Hybrid and NSGA-II to achieve better balance.
3. We conducted experiments to evaluate different algorithms in terms of both optimization quality and the ability of early fault detection. The experimental results establish the following findings:
   (a) The LKH solver for TSP can yield the order with the lowest switching cost. It can make faults be detected earlier than the order with high rate of combination coverage.
   (b) However, to further improve the ability of early fault detection, considering both combination coverage and testing cost is recommended.
4. We conducted an empirical study and reported our findings based on six real world applications.

The rest of this paper is organized as follows. Section 2 introduces the basic concepts of CT and switching cost with a motivating example. Section 3 presents the measurement of switching cost, and discuss its characteristics. Section 4 presents all algorithms that are used in this work. Section 5 gives our experiment and analysis. Section 6 reports real world case studies. Section 7 discusses the threats to validity. Section 8 summarizes the related work, and Section 9 concludes this paper.

2. Motivating example

Mobil apps are reshaping human daily life. Currently, there are more than 1.5 million apps available to download for either Apple or Android users. However, it is not easy to assure their quality and user experience, as the behaviour of apps is often affected by many factors.

For example, let us consider a system build-in app: the voice call. How this app handles a call depends on phone environmental settings and users actions. If the user enables sound, the phone should ring when there are incoming calls, and if the user switches the mode to airplane, no calls can be received or made. Furthermore, the system should allow the user to call numbers from text content such as SMS, notes or web sites. If the user is listening to music while a call arrives, the user would love the system to resume the music as soon as the call is finished.

To test such functionalities by CT, firstly we need to define the testing model. In this paper we suppose that the SUT (Software Under Test) contains n independent parameters, and each parameter p_i has v_i discrete values chosen from a value set V_i ($v_i = |V_i|$). We call an $n-$tuple $(x_1, x_2, \ldots, x_n)$ as a test case t, where $x_i \in V_i$ for $1 \le i \le n$; and a set of test cases as a test suite $T = \{t_1, t_2, \ldots, t_m\}$. In our example, to represent different scenes of making and receiving voice calls, we select five parameters to form a simplified testing model. Table 1 shows it, where the first three parameters are