Contents lists available at ScienceDirect

Information Sciences

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

An optimal mutation execution strategy for cost reduction of mutation-based fault localization

Yong Liu, Zheng Li, Ruilian Zhao*, Pei Gong

College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China

ARTICLE INFO

Article history:
Received 30 August 2016
Revised 5 July 2017
Accepted 4 September 2017
Available online 6 September 2017

Keywords:
Software debugging
Mutation-based fault localization
Mutation cost reduction
Dynamic mutation execution strategy

ABSTRACT

Identifying faulty program entities has been recognized as one of the most expensive, tedious and time-consuming processes in software debugging activity. Fault localization techniques are designed to assist developers in locating faults by giving a ranking of the probability that program entities incur failures. Mutation-based fault localization (MBFL) is a recently proposed fault localization approach via mutation analysis, which uses the location of mutants to identify the faulty statements. With improved effectiveness, the MBFL also brings huge execution cost. To reduce the execution cost of MBFL technique, this paper proposes a dynamic mutation execution strategy (DMES) to prioritize the execution on both mutants and test cases. As fewer mutants and test cases are executed with DMES, the whole process will become faster and the cost will be decreased. At the same time, it is proved that the fault localization accuracy of MBFL with DMES is the same as that of the original MBFL. Furthermore, this paper discusses the lowest mutation execution cost of MBFL in theory and gives a quick solution to compute the theoretical minimal mutation execution cost of MBFL in the case of keeping fault localization accuracy non-decreasing. The empirical studies show that MBFL with the dynamic strategy proposed in this paper can reduce mutant-test execution cost by 32.4–87% and is close to the theoretical optimal cost. Furthermore, the additional run time required by utilizing our strategy is minimum and can be ignored.

1. Introduction

Both industrial software developers and software engineering researchers are interested in measuring software debugging quality since debugging is an essential process to improve the software quality [6]. When detecting defects in a program, developers need to identify the exact locations of program faults and ultimately fix them. The identification of the defective statement locations is usually referred to as the fault localization process, which has been recognized as one of the most expensive, tedious and time-consuming processes in the whole debugging activity [19,26,44,45], for it usually is difficult to understand the complex internal logic of the program under test and reason the possible locations of faults. Therefore, many researchers have been paying much attention attention to fault localization in the past decades and have put great effort to automating the fault localization activity that can guide developers to locate faults with minimal human intervention.

Nowadays, various Fault Localization (FL) methods have been proposed to effectively detect the location of faults, such as state-based FL [46,49], slice-based FL [14], spectrum-based FL [1,31,37,39,43] and mutation-based FL [7,13,15,30,33–35]. State-

* Corresponding author.
E-mail address: rlzhao@mail.buct.edu.cn (R. Zhao).

http://dx.doi.org/10.1016/j.ins.2017.09.006
0020-0255/© 2017 Elsevier Inc. All rights reserved.

based FL method modifies values of variables to determine which one is the cause of erroneous program execution [46]. Slice-based FL method uses static analysis to extract the parts affecting the given variables and inspects the only associated statements [14]. Spectrum-based FL is one of the traditional approaches, which uses the coverage information and test results to estimate the probability that program entities incur fault, also known as Coverage-based Fault Localization (CBFL). Empirical evidence has shown that CBFL is effective and helpful in diminishing the debugging effort [2]. However, CBFL has also been criticized for its impractical localization accuracy and the unrealistic usage model, i.e., all statements in the same basic block share the same spectrum and, therefore, the same ranking. This often increases the number of statements needed to be checked before encountering the faults [30].

Mutation-based fault localization (MBFL) is a recently proposed FL approach using mutation analysis [33]. It is observed that the faults on the same statement are more likely to have similar behaviors than on the diverse statements when running the same test suite [15,35]. Therefore, the underlying idea of MBFL is to gain information from mutants, which have been shown to correlate with faults [3], and provide guidance to distinguish faulty statements by identifying suspicious mutants and their locations. Thus, for each statement covered by failed test cases, a set of mutants are created and rechecked by the previous test suite. The suspiciousness of each statement is calculated, and then the statements are ranked according to their suspiciousness values from high to low.

Two main ways have been presented to compute the suspiciousness value of statements [7]. The first one is called traditional MBFL, such as Metallaxi, which advocates the idea that killing mutants results in finding faults and localizes faults based on the associated mutants in accordance with the test suite. Similar to CBFL approaches, Metallaxi calculates the suspiciousness of mutants by straightforwardly applying the existing formula *Ochiai* in CBFL with elements representing the killed mutants. Then, the maximum is set to the associated statement as its suspiciousness value [35]. The other is named as MUSE, however, follows a different way in identifying suspicious statements. Instead of using the traditional way of judging whether mutants are killed or not, MUSE localizes faults based on mutants that turn failed test executions into passed ones or vice versa. Thus, the suspiciousness values of program statements are measured by the sum of two terms, one of which represents the proportion of test cases turned from failed to passed, and the other of which is the proportion of test cases turned from passed to failed [30].

Mutation analysis explores the programs with respect to their behavior rather than code coverage by injecting artificial faults into its code, and the mutants as relevant substitutes of real faults could be useful to improve fault localization activity [7,15,35]. Therefore, it is assumed that MBFL is more precise than the traditional CBFL. However, the mutation analysis also brings huge execution cost since it requires a vast number of defects to be injected and related mutants to be executed with the whole test suite. This seriously restricts the applications of MBFL technique into practice.

There have been a lot of studies on reducing mutation execution cost, of which two approaches have become prominent: mutation sampling and selective mutation [18]. They decrease the cost by randomly sampling mutants or limiting the types of mutation operators, but it has been reported in the literature that useful mutants may be removed simultaneously [34], bringing the threat of reducing fault localization ability. In addition, many improved mutation strategies are proposed to avoid the executions of test cases that are not valuable [20,29,36,48]. For example, identifying uncovered mutants can filter the mutant execution task and refrain from executing the test cases against the mutants whose mutated statement are not covered by the test cases, therefore reducing the number of total required executions in a mutation analysis. And there is an interesting observation that the execution order of test cases has an influence on the number of mutant executions [29]. Additionally, high-quality test data can be produced by combining mutant schemata and dynamic symbolic execution at a relatively low cost, and they are able to kill a remarkable number of the mutants, resulting in major execution saving [36]. However, these strategies designed for mutation testing are not suitable for MBFL. More specifically, in mutation testing, when a test case is found to kill a mutant, the executions of the remaining test cases on that mutant are unnecessary, thus could be skipped or ignored. However, the traditional MBFL assigns the suspiciousness value to a statement with the maximum suspiciousness of its associated mutants. Hence, only the executions of test cases on mutants with lower suspiciousness value are useless and can be reduced. The MUSE measures the suspiciousness value to program statement by the sum of two proportions, and then all test executions on mutants cannot be canceled.

Recent research demonstrated that MBFL techniques are relatively accurate and practical. More specifically, both traditional MBFL and MUSE significantly outperform the state-of-the-art CBFL techniques [30,35]. Moreover, Metallaxis are twice as much accurate as MUSE with only half of the cost. [7], but its execution cost is still very high. So, the study reported in this paper focuses on reducing the cost of traditional MBFL by dynamically prioritizing the mutants and test cases that can contribute to higher suspiciousness value, and a Dynamic Mutation Execution Strategy (DMES) is proposed, which contains execution optimizations on both mutants and test cases. As fewer mutants and test cases are executed with DMES, the whole process will become faster and the cost will be decreased without losing the accuracy of fault localization.

The main contributions of the present paper are summarized as follows:

- The paper proposes a dynamic mutation execution strategy that includes optimization on two aspects, i.e., mutants reduction and test cases reduction respectively, and presents how to use the presented strategy in MBFL to calculate the suspiciousness value of statement during executing mutants.
- The paper proves that MBFL with the proposed DMES would not decrease the accuracy of fault localization, discusses the lowest mutation execution cost of MBFL in theory, and presents a quick solution to compute the theoretical minimal mutation execution cost of MBFL.