Contents lists available at ScienceDirect

Information and Software Technology

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

Mutant reduction based on dominance relation for weak mutation testing

Dunwei Gong[a,b], Gongjie Zhang[c,e,*], Xiangjuan Yao[d], Fanlin Meng[f]

[a] School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu, 221116, China
[b] School of Electrical Engineering and Information Engineering, LanZhou University of Technology, Lanzhou, Gansu, 730050, China
[c] School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu, 221116, China
[d] School of Science, China University of Mining and Technology, Xuzhou, Jiangsu, 221116, China
[e] School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, Jiangsu, 221116, China
[f] Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, LE11 3TU, UK

A R T I C L E I N F O

Article history:
Received 17 August 2015
Revised 29 April 2016
Accepted 2 May 2016
Available online xxx

Keywords:
Software testing
Weak mutation testing
Mutant
Reduction
Dominance relation

A B S T R A C T

Context: As a fault-based testing technique, mutation testing is effective at evaluating the quality of existing test suites. However, a large number of mutants result in the high computational cost in mutation testing. As a result, mutant reduction is of great importance to improve the efficiency of mutation testing.

Objective: We aim to reduce mutants for weak mutation testing based on the dominance relation between mutant branches.

Method: In our method, a new program is formed by inserting mutant branches into the original program. By analyzing the dominance relation between mutant branches in the new program, the non-dominated one is obtained, and the mutant corresponding to the non-dominated mutant branch is the mutant after reduction.

Results: The proposed method is applied to test ten benchmark programs and six classes from open-source projects. The experimental results show that our method reduces over 80% mutants on average, which greatly improves the efficiency of mutation testing.

Conclusion: We conclude that dominance relation between mutant branches is very important and useful in reducing mutants for mutation testing.

1. Introduction

Software testing, which is used to seek existing defects or faults in software before it is released to the market, is an important way to improve software quality. Mutation testing is commonly used to evaluate the quality of existing test suites to guide testers how they might be able to improve them [1]. Compared with other structural coverage criteria, test suites that are mutation adequate can reveal more faults [2]. It is noticeable that mutation testing has attracted widespread attention from researchers and developers in both academia and industry.

Mutation testing is a fault-based technique [3,4], and the related concepts are given as follows. By making a simple syntactic change to the original program, a mutant is generated. A rule used to perform the syntactic changes is called a mutation operator. If a test datum can distinguish the outputs between a mutant and its original program, the mutant is said to be *killed*. A mutant is equivalent, if it cannot be killed by any test datum. Generally, the adequacy of mutation testing named *mutation score* is defined as the ratio of the number of killed mutants to the total number of non-equivalent mutants.

In order to optimize the execution of the traditional mutation testing, Howden first proposed weak mutation testing [5]. Instead of checking a mutant after executing the whole program, weak mutation testing checks a mutant immediately after executing the mutated statement.

In mutation testing, mutants are employed to reflect possible real faults in software under test [6–8]. Many lines of code (LOCs), complicated statements, and a variety of data types [9] in software greatly increase the number of mutants. It results in the high computational cost in mutation testing, therefore the mutant reduction is of great importance and necessity. Although there have been several techniques for mutant reduction [10–16], their efficiency needs to be further improved.

Just et al. focused on the COR and ROR mutation operators to identify redundant mutants [13]. Kaminski et al. sought a subset

* Corresponding author.
E-mail addresses: dwgong@vip.163.com (D. Gong), zhanggongjie@126.com (G. Zhang).

http://dx.doi.org/10.1016/j.infsof.2016.05.001
0950-5849/© 2016 Elsevier B.V. All rights reserved.

of relational operators that subsumes the others to reduce mutants [14]. Focusing only on a subset of mutation operators opens new research directions [13,14]. Papadakis and Malevris transformed the problem of killing mutants into the problem of covering mutant branches in the new program, and generated test data by conventional approaches [17]. Although it is relatively efficient, a large number of mutants without reduction, will inevitably add high complexity to the new program.

In the previous work on dominance analysis, Marre and Bertolino employed the subsumption relation between entities in a ddgraph (a simplified control flow graph) to seek the minimal set of entities named the spanning set, so as to reduce the number of entities needed to cover [18]. In addition, Ghiduk and Girgis identified the non-dominated nodes in a control flow graph (CFG) by analyzing the dominance relation between nodes [19]. Both the above methods are performed among the original entities (nodes) for structural coverage testing. Different from the above work, we analyze the dominance relation between mutant branches, which are instrumented branches transformed from mutants based on the method presented by Papadakis and Malevris for weak mutation testing, with the aim to reduce the number of mutants, and to improve the efficiency of testing.

Considering all the traditional (method level) mutation operators, we first construct mutant branches based on the statements before and after mutation, and form the new program by fusing all mutant branches into the original program using the method proposed by Papadakis and Malevris [17]. Then, we identify redundant mutants according to the dominated mutant branches after manual analysis with the aid of the dominance relation graph. Mutants associated with the non-dominated ones will remain. The test data that cover the non-dominated mutant branches can also cover all the mutant branches, i.e., kill all the mutants before reduction in weak mutation testing.

The basic idea of defining the dominance relation between mutant branches and applying the dominance relation to reduce mutants was initially reported, with examples on several small programs, at the 2nd Chinese Search Based Software Engineering (CSBSE'2013) workshop [20]. Given the fact that the two-page abstract is preliminary, we have extended the idea in the following four new directions:

(1) defining four concepts, mutant branch, dominance relation, non-dominated branch, and dominance relation graph;
(2) presenting two theorems on how to form the non-dominated mutant branch set and identify the non-dominated mutants;
(3) providing an example throughout the whole paper to intuitively demonstrate the above work;
(4) evaluating the proposed method by applying it to ten benchmark programs and six classes from open-source projects with various sizes and complexities.

The main contributions of this paper are as follows:

• A method of reducing mutants is proposed for weak mutation testing, which is conducted by analyzing the dominance relation between mutant branches in the new program.

• Four definitions of identifying the dominance relation between mutant branches are provided, and the dominance relation graph is given to describe all the dominance relations in the new program.

• Two theorems of determining the non-dominated mutant branches are given, so as to reduce redundant mutants.

• The proposed method is applied to ten benchmark programs and six classes from open-source projects, and the experimental results suggest that our method reduces over 80% mutants.

2. Related work

Reducing mutants is of effectiveness to save computational cost for mutation testing. Weak mutation testing is a technique in view of saving execution time. Additionally, there are correlations among statements in a program, and correlation analysis is helpful to mutation testing. This section will review the related work from the above aspects.

2.1. Mutant reduction

For software under test, a large number of mutants cause high cost in mutation testing, which can be solved by mutant reduction. Mutant sampling proposed by Acree and Budd randomly selects a specific percentage of mutants to execute testing [21,22]. Mathur and Wong investigated the influence of the sampling rate on the mutation adequacy [23]. They conducted a series of experiments by changing the rate from 0.1 to 0.4 in the step of 0.05, and the experimental results suggest that the mutation score decreases as the sampling rate reduces. Unlike random sampling, Hussain extracted a set of representative mutants to test after clustering, and his method maintains a high mutation score [24].

To reduce mutants, Mathur suggested that two mutation operators (i.e., Array reference for Scalar variable Replacement (ASR) and Scalar Variable Replacement (SVR)), which will generate around 30% to 40% of the total mutants, should be omitted, and proposed selective mutation testing [25]. By Extending Mathur's idea, Offutt et al. performed 2-selective experiments (omitting ASR and SVR) and achieved 24% mutant reduction with 99.99% mutation score. Further in 4-selective and 6-selective experiments, they got much higher reduction rates but lower mutation scores [26]. By analyzing the distribution of 22 mutation operators in 28 programs, Offutt et al. obtained a high mutation score with only 5 operators [27]. Different from the prior selective mutation that reduces mutants with acceptable loss in the test adequacy, Mresa and Bottaci compared mutation operators in terms of both score and cost to seek the most efficient mutation operators [28]. By manual analysis, Yao et al. revealed a highly uneven distribution of equivalent mutants and stubborn mutants (those that remain undetected by a test suite with high quality, yet are non-equivalent) [29]. Their work is beneficial to designing mutation tools, and reduces the human effort involved in mutation testing.

The above techniques, mutant sampling and selective mutation, are from First Order Mutants (FOMs) perspective. In view of the fact that High Order Mutants (HOMs, more than one syntactic change in a program) not only represent complex faults in practical software, but also reduce the number of mutants [30], studies on HOMs have arisen in recent years. Jia and Harman utilized meta-heuristic search algorithms to generate semantic HOMs which are hard to kill, and reduced mutants greatly [4]. Langdon et al. combined FOMs with close semantic relations to form HOMs [31]. Second-order mutants (SOMs, two syntactic changes in a program) proposed by Polo et al. get 50% cost saving [32]. Kintis et al. focused on control relations among nodes in the CFG of a program, and presented three strategies for combining SOMs [33]. Papadakis and Malevris conducted an empirical study for the first and the second order mutation testing strategies, and found that the first order mutation testing strategies are generally more effective than the second order ones, and the latter drastically reduce equivalent mutants, thus forming a valid cost effective alternative to mutation testing [34]. However, the growing order of HOMs results in significant cost increase in generating HOMs, which further illustrates the necessity of reducing mutants.