RESEARCH-ARTICLE

Mutant Reduction Evaluation: What is There and What is Missing?

PENG ZHANG, Nanjing University, Nanjing, Jiangsu, China

YANG WANG, Nanjing University, Nanjing, Jiangsu, China

XUTONG LIU, Nanjing University, Nanjing, Jiangsu, China

YANHUI LI, Nanjing University, Nanjing, Jiangsu, China

YIBIAO YANG, Nanjing University, Nanjing, Jiangsu, China

ZIYUAN WANG, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

View all

Published: 12 July 2022
Online AM: 15 March 2022
Accepted: 01 December 2021
Revised: 01 November 2021
Received: 01 July 2021

Mutant Reduction Evaluation: What is There and What is Missing?

PENG ZHANG, YANG WANG, XUTONG LIU, YANHUI LI, and YIBIAO YANG,
Nanjing University
ZIYUAN WANG, Nanjing University of Posts and Telecommunications
XIAOYU ZHOU, Southeast University
LIN CHEN and YUMING ZHOU, Nanjing University

Background. Mutation testing is a commonly used defect injection technique for evaluating the effectiveness of a test suite. However, it is usually computationally expensive. Therefore, many mutation reduction strategies, which aim to reduce the number of mutants, have been proposed.

Problem. It is important to measure the ability of a mutation reduction strategy to maintain test suite effectiveness evaluation. However, existing evaluation indicators are unable to measure the "order-preserving ability", i.e., to what extent the mutation score order among test suites is maintained before and after mutation reduction. As a result, misleading conclusions can be achieved when using existing indicators to evaluate the reduction effectiveness.

Objective. We aim to propose evaluation indicators to measure the "order-preserving ability" of a mutation reduction strategy, which is important but missing in our community.

Method. Given a test suite on a **Software Under Test (SUT)** with a set of original mutants, we leverage the test suite to generate a group of test suites that have a partial order relationship in defect detecting ability. When evaluating a reduction strategy, we first construct two partial order relationships among the generated test suites in terms of mutation score, one with the original mutants and another with the reduced mutants. Then, we measure the extent to which the partial order under the original mutants remains unchanged in the partial order under the reduced mutants. The more partial order is unchanged, the stronger the **Order Preservation (*OP*)** of the mutation reduction strategy is, and the more effective the reduction strategy is. Furthermore, we propose **Effort-aware Relative Order Preservation (*EROP*)** to measure how much gain a mutation reduction strategy can provide compared with a random reduction strategy.

Result. The experimental results show that OP and EROP are able to efficiently measure the "order-preserving ability" of a mutation reduction strategy. As a result, they have a better ability to distinguish various mutation reduction strategies compared with the existing evaluation indicators. In addition, we find

This work is partially supported by the National Natural Science Foundation of China (61772259, 62172205, 61872177, 62072194, 62172202).

Authors' addresses: P. Zhang, Y. Wang, X. Liu, Y. Li (corresponding author), Y. Yang, L. Chen (corresponding author), and Y. Zhou (corresponding author), Department of Computer Science and Techonology Nanjing University 163 Xianlin Avenue, Qixia District Nanjing, Jiangsu Province, China, 210023; emails: {dz1833034, njuwy, xryu}@smail.nju.edu.cn, {yanhuili, yangyibiao, lchen, zhouyuming}@nju.edu.cn; Z. Wang, School of Computer Science Nanjing University of Posts and Telecommunications 9 Wenyuan Road, Qixia District Nanjing, Jiangsu Province, China, 210023; email: wangziyuan@njupt.edu.cn; X. Zhou, School of Computer Science and Engineering Souteast University 2 Souteast University, Jiangning District Nanjing, Jiangsu Province, China, 211189; email: zhouxy@seu.edu.cn.

1049-331X/2022/07-ART69 $15.00
https://doi.org/10.1145/3522578

that **Subsuming Mutant Selection (SMS)** and **Clustering Mutant Selection (CMS)** are more effective than the other strategies under OP and EROP.

Conclusion. We suggest, for the researchers, that OP and EROP should be used to measure the effectiveness of a mutant reduction strategy, and for the practitioners, that SMS and CMS should be given priority in practice.

CCS Concepts: • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Mutant reduction, evaluation, test suites, order preservation

ACM Reference format:
Peng Zhang, Yang Wang, Xutong Liu, Yanhui Li, Yibiao Yang, Ziyuan Wang, Xiaoyu Zhou, Lin Chen, and Yuming Zhou. 2022. Mutant Reduction Evaluation: What is There and What is Missing? *ACM Trans. Softw. Eng. Methodol.* 31, 4, Article 69 (July 2022), 46 pages.
https://doi.org/10.1145/3522578

1 INTRODUCTION

In software development, test suites are widely used to test a software system to examine whether it has faults. In this context, evaluating the test effectiveness of a test suite is essential for many activities. On the one hand, it enables developers to know the fault detecting potential that a given test suite has. Knowing such an "absolute" effectiveness score of a test suite can inform whether there is a need to generate new test cases to enhance its effectiveness. On the other hand, it enables to compare the fault detecting potential among different test suites. Knowing the "relative" order of the test effectiveness score among test suites can provide a useful means for comparing the effectiveness of the corresponding test-generation techniques.

Currently, mutation testing is regarded as a useful technology to evaluate the effectiveness of a test suite.[1] Specifically, the evaluation process proceeds as follows. First, a set of mutants are generated by modifying a given **SUT (Software Under Test)** in small ways (i.e., by applying a number of mutation operators that mimic typical programming errors). A mutant will be called "killed" if the test suite can distinguish the outputs between the SUT and the mutant; otherwise, it will be called "alive" under the test suite. Then, the effectiveness of the test suite is measured by mutation score, an indicator denoting the ratio of killed mutants to all the non-equivalent mutants (a mutant is called "equivalent" if its output cannot be distinguished from the output of the SUT by test cases). The higher the mutation score, the more effective the test suite is considered. In the literature, mutation score has been a commonly used indicator to evaluate the effectiveness of a given test suite. By mutation score, developers can not only know the "absolute" effectiveness of a given test suite, but also know the "relative" order of the test effectiveness among different test suites.

However, the main disadvantage of mutation testing comes from the huge cost of executing all mutants [1, 2]. Selecting a mutant subset from all mutants is a way to reduce overhead. In the last decades, many reduction strategies have been proposed based on two main practices. The first practice is leveraging the characteristics of the mutants to select the mutants which are worth executing (e.g., selecting the mutants generated by specific mutation operators [3–7]). The rationale behind this practice is that some mutants are more relevant to faults. As a result, it is more practical to leverage their characteristics to search the subset. The second practice is selecting a mutant subset to maximize a pre-defined optimization function [8]. One of the most used optimization functions is to maximize the global mutation score [27]. To compute the global mutation score,

[1]Note that, in the literature, mutation testing can be used for different purposes: using mutants to decide when to stop testing, or developing test suites for killing all the mutants, or distinguishing between a stronger and a weaker test suite. In our study, we focus on the last point, i.e., mutants are used to distinguish between a stronger and a weaker test suite.

we start with an adequate test suite that can kill all mutants. Then, for the reduced mutant set given by a reduction strategy, we select a minimal set of test cases that can kill the reduced set. At last, we run this test set against the original mutant set to compute a mutation score as the global mutation score. The rationale behind setting such an objective function is that the criterion of mutant reduction is to maintain the effectiveness of the mutation set (i.e., the number of unique faults that the set contains), which should be measured by global mutation score. Another optimization function is to minimize the variation in mutation scores. The rationale behind setting such an objective function is that the criterion of mutant reduction is to keep the evaluation of a test set consistent, which is approximated by mutation score.

Given a mutant reduction strategy, an important question is how to objectively evaluate its reduction effectiveness, i.e., the ability to maintain test suite effectiveness evaluation. Normally, mutation score is used as a metric to measure the effectiveness of test suites. A mutant reduction strategy aims to provide a simple way to compute a new mutation score as a new metric to measure the effectiveness of test suites. In this context, we should measure the difference between the two metrics to evaluate the effectiveness of the mutant reduction strategy. Specifically, under the original mutants, developers can obtain the "absolute" effectiveness of each single test suite as well as the "relative" order of the effectiveness among different test suites. Ideally, a mutant reduction strategy should not lead to a change in the "relative" order of the effectiveness among different test suites. Note that the "absolute" values can be changed. For example, filtering equivalent mutants can increase the mutation score. However, the increase of mutation score does not mean the improvement of the effectiveness of test suites. In this sense, it is not necessary to maintain the "absolute" values. In fact, it is more important to preserve the relative order among test suites. Otherwise, a reduction strategy will twist the evaluation for test suites. In other words, if the mutation reduction technique used by practitioners cannot provide a good order preservation, they are likely to obtain wrong conclusions when comparing multiple test suites.

In our community, there are two main indicators for evaluating the effectiveness of a mutation reduction strategy (i.e., how well a subset represents the original mutant set): **GMS (Global Mutation Score)** [4, 6–11] and **VMS (Variation in Mutation Score)** [1, 20]. The larger the GMS is, the better the subset represents the original mutant set. Different from GMS, the computation of VMS does not require an adequate test suite to kill all non-equivalent mutants in the original mutant set: given a test suite, simply compare the mutation score on the mutant subset with the mutation score on the original mutant set. The smaller the difference (i.e., VMS) is, the better the subset represents the original mutant set. As can be seen, for a reduction strategy, both GMS and VMS ignore measuring their abilities to maintain the "relative" order of the effectiveness among test suites. As a result, it is not possible to obtain a comprehensive understanding on the effectiveness of a mutant reduction strategy. This may lead to a misjudging on the actual effectiveness of mutant reduction strategies and hence may affect their use in software development (the detailed motivating examples which show the problems of GMS and VMS are in Section 3.2).

In order to tackle the above problem, in this study, we propose evaluation indicators to measure to what extent a mutation reduction strategy can maintain the "relative" order of the test effectiveness among test suites. More specifically, for a test suite on a SUT with a set of original mutants, we leverage the test suite to generate a group of test suites that have a partial order relationship in defect detecting ability. When evaluating a reduction strategy, we first obtain two partial order relationships among the generated test suites in terms of mutation score, one with the original mutants and another with the reduced mutants. Then, we measure the extent to which the partial order under the original mutants remains unchanged in the partial order under the reduced mutants. The more partial order is unchanged, the stronger the **Order Preservation (OP)** of the mutation reduction strategy is, and the more effective the reduction strategy is.

Furthermore, we propose **Effort-aware Relative Order Preservation (EROP)** to measure how much gain a mutation reduction strategy can provide compared with a random reduction strategy. Based on a number of open-source programs, we investigate the usefulness of OP and EROP in understanding the effectiveness of various mutation reduction strategies. The experimental results show that OP and EROP can complement existing evaluation indicators for distinguishing various mutation reduction strategies. In addition, we find that **Subsuming Mutant Selection (SMS)** and **Clustering Mutant Selection (CMS)** are more effective than the other strategies under OP and EROP.

In this study, we make the following main contributions:

- We conduct an in-depth analysis on the pitfalls of existing indicators for evaluating the effectiveness of mutant reduction strategies. We show that, for a mutant reduction strategy, the existing evaluation indicators are unable to depict its ability to maintain the "relative" test effectiveness order among test suites that have a partial order relationship in defect detecting ability. This leads to an incomprehensive (or even counter-intuitive) understanding on the actual effectiveness of a mutant reduction strategy.
- We propose two indicators to evaluate the effectiveness of a mutant reduction strategy: OP and EROP. Different from the existing evaluation indicators, OP and EROP are able to measure the ability of a mutant reduction strategy to maintain the "relative" order of the test effectiveness among different test suites that have a partial order relationship in defect detecting ability. To the best of our knowledge, this is the first approach that takes the order-preserving ability into account. By revealing the pitfalls of the existing indicators, both experimentally and analytically, we seriously suggest that OP and EROP should be taken as the main evaluation indicators in the review of previous studies and the exploration of future work on mutant reduction.
- We conduct an experiment to investigate the effectiveness of five mainstream mutation reduction strategies with the existing and our proposed indicators.[2] The investigated strategies include **RMS (Random Mutant Selection), COS (Certain Operator Selection), SMS (Subsuming Mutant Selection), CMS (Clustering Mutant Selection)**, and the state-of-the-art mutation reduction strategy **Sentinel**. The experimental results show that OP and EROP are better than the existing evaluation indicators for distinguishing various mutation reduction strategies. In particular, SMS and CMS are found to be more effective than the other strategies under OP and EROP.

The rest of this paper is organized as follows. In Section 2, we introduce the relevant background, including mutation testing, mutant reduction, and existing evaluation indicators. In Section 3, we analyze the pitfalls of the existing evaluation indicators and propose our solutions. Section 4 presents the data sets used in this study and the experimental design. Section 5 reports the experimental results. Section 6 discusses the experimental results. Section 7 presents the implications of our study. Section 8 analyzes the threats to the validity of our study. Section 9 concludes the paper and outlines directions for future work.

2 BACKGROUND

In this section, we first introduce mutation testing. Then, we introduce mutant reduction. Finally, we overview the commonly used indicators for evaluating the effectiveness of a mutant reduction strategy.

[2]The replication kit is available via https://github.com/zhangpengNJU/MutantReductionEvaluation.

2.1 Mutation Testing

Mutation testing is a testing technology that mutates (changes) statements in SUT to obtain faulty versions (i.e., mutants) to simulate real defects. There are two concepts of mutation testing: strong mutation testing and weak mutation testing. In the former concept, killing means that the mutant and the original program generate different outputs. In the latter concept, instead, killing means that the program state of the mutant differs from the original one at some point during execution. When we refer to mutation testing, we are talking about strong mutation testing in this paper. The mutants are generated automatically by a series of pre-defined mutation operators (i.e., mutators). A test case is said to kill a mutant when the test result differs between the mutant and the SUT according to the strong mutation testing criterion. A mutant is said to be alive (i.e., survived) if and only if none of the test cases kills it. Mutation score is a commonly used indicator to measure the effectiveness of a test suite, which is computed by dividing the number of killed mutants by the total number of mutants:

$$MS\,(M,T) = \frac{|KM\,(M,T)|}{|M|} \tag{1}$$

where M is a set of mutants and T is a test suite (i.e., a set of test cases). *MS(M, T)* is the mutation score computed when executing T against M; *KM(M, T)* is the set of mutants in M killed by T; and $|M|$ is the number of mutants in M. For convenience, we say a test suite T kills a mutant set M if and only if *MS(M, T)* = 100%. A test suite with a high mutation score indicates that it can kill most of the mutants, which is considered to have a strong ability to detect real defects [3, 4, 12–15]. Equivalent mutants refer to those mutants which are semantically equivalent to the original program. Sometimes, researchers exclude the equivalent mutants, as they will never be killed by any test case. It is worth noting that the equivalent mutants do not impact our study in this paper. Equivalent mutants only affect the upper bound of mutation scores. As a result, it cannot change the order between two test suites w.r.t. mutation scores. For two test suites $T1$ and $T2$, if $T1$ achieves a higher score than $T2$ on the original mutant set, $T1$ still achieves a higher score on the mutant set without equivalent mutants. For this reason, we do not filter the equivalent mutants in this paper.

With the support of mutation score, on the one hand, developers can quantify the "absolute" effectiveness of a single test suite. This enables to determine whether a test suite has a weak fault detecting potential. On the other hand, for a group of test suites, developers can leverage their mutation scores to obtain a "relative" test effectiveness order. If these test suites are generated by different test-generation techniques, this "relative" order will provide a means to determine which test-generation techniques are superior [21]. As can be seen, for practitioners, both the "absolute" effectiveness score and the "relative" effectiveness order are important for understanding the test effectiveness of a test suite. For test suites, once a set of mutations is given, their "absolute" effectiveness scores and "relative" effectiveness order will hence be determined.

2.2 Mutant Reduction

Considering that the number of mutants is often hundreds to thousands, the overhead of executing all mutants is often unacceptable. Therefore, mutant reduction has become a research hotspot which aims to find a subset M_s of all mutants M for reducing the cost. Currently, there are two main practices for reduction strategies: one is leveraging the characteristics of the mutants to select the mutants which are worth executing, while the other one is selecting a mutant subset to maximize or minimize a pre-defined optimization function.

For the first practice, the core idea is to leverage the characteristics of mutants that are worth preserving to reduce mutants. The rationale is that some mutants are more relevant to faults or more representative than the others. From this perspective, several reduction strategies have been proposed:

- **COS (Certain Operator Selection)** [3–7, 23]: Select the mutants generated by a specific mutator subset of all mutators. There are numerous practices on COS whose core idea is to remove redundant mutation operators and select important operators that are more related to defects. Mathur first proposed COS in 1991 and call it selective mutation [23]. They found that the number of redundant mutants can be reduced by 30% to 40% by deleting the "arithmetic operator replacement" mutator and the "scalar variable replacement" mutator. Later, Offutt et al. identified five important mutation operators for COS, including the relational, logical, arithmetic, absolute, and unary insertion operators [4].
- **SMS (Subsuming Mutant Selection)** [10–13, 44, 45]: Select the subsuming mutants. By the definition in [13], "One mutant subsumes another if at least one test kills the first and every test that kills the first also kills the second". For all mutants, killing the subsuming mutants is to kill all the killable mutants. Therefore, the subsumed mutants should be regarded as redundancy. The practical difficulty of SMS lies in how to figure out the "true" subsumption relationships among all the mutants. In [13], Kurtz et al. proposed both dynamic and static methods to approximate the "true" subsumption. In [11], Gong et al. manually inserted mutant branches into the original program to generate a giant program which is used to analyze the approximation of the "true" subsumption.
- **CMS (Clustering Mutant Selection)** [24]: Select one mutant from each mutant cluster. First, for all the mutants, a number of features are collected. Second, the data for the mutant instances is used as the input of a clustering algorithm. Third, by the output of the clustering algorithm, the mutants can be grouped into different clusters. Fourth, the selection is generated by selecting one mutant from each mutant cluster. For example, Hussain [24] executed all the mutants to obtain the information on if a test can kill a mutant or not against all the tests and mutants. After that, K-means and Agglomerative clustering were applied to cluster the mutants. By the clustering algorithm, the mutants in the same cluster were guaranteed to be killed by a similar set of test cases. As a result, randomly selecting one mutant from each cluster can be seen as a selection of representative mutants.

For the second practice, Global Mutation Score is one of the most used objectives for the pre-defined optimization function, which is used by the state-of-the-art mutation reduction strategy Sentinel [8]. Considering the complex computation of Global Mutation Score, we will elaborate on it in the next subsection. Here, we give a brief introduction to Sentinel:

- Sentinel [8]: Generate the cost-optimal reduction strategies instead of selecting mutants directly. To search the possible strategy space, Sentinel defines two objective functions. One objective function is to minimize the execution time. The other objective function is to maximize the Global Mutation Score. With several pre-defined basic mutant selection (i.e., mutant reduction) strategies, Sentinel searches a combination of basic strategies instead of selecting mutants directly for the training project, which can achieve the best values of the objective functions.

Besides GMS, the following is the other optimization function for the second practice:

$$\min |MS(M_s, T) - MS(M, T)| \tag{2}$$

The meaning of this objective function is as follows: For a given test suite T, the mutation score on the reduced mutant set should be as close as possible to the mutation score on the original mutant set. By (2), the second practice usually attempts to search the mutant subset with an approximate mutation score to the original mutation score, against the given test suite. The rationale behind setting such an objective function is that the criterion of mutant reduction is to keep the evaluation

of a test set consistent, which is approximated by mutation score. Considering the wide use of this idea, there are two mutant reduction strategies discussed, which can be classified as follows:

- **RMS (Random Mutant Selection)** [9, 16–18]: Select a specified number or proportion of mutants from all mutants randomly. For such a simple strategy, the selecting process of RMS does not need the guidance of an objective function. However, it is true that RMS can maintain the mutation score in mathematical expectation. As a result, we can regard the objective function (2) as implied by RMS (please see the appendix for more details).
- **ROS (Random Operator Selection)** [19]: Randomly select a specified number or proportion of operators to generate mutants. Similar to RMS, the objective function (2) is implied by ROS, as ROS can also maintain the mutation score in mathematical expectation.

After a brief introduction to several major reduction strategies, we will describe how the existing work evaluated these strategies in the next section.

2.3 Mutant Reduction Evaluation Indicator

When evaluating a reduction strategy, researchers are most concerned with the following questions: How many mutants have been reduced? What percentage of unique faults are retained in the reduced mutant set, or does the evaluation on the given test suite change significantly before and after reduction? As a result, they mainly evaluate a reduction strategy from two dimensions: one is the number of reductions, while the other is the effect of reductions.

To hit the first dimension, the simplest indicator is the Reduction Ratio *RR* of mutants:

$$RR = \frac{|M| - |M_s|}{|M|} \tag{3}$$

Besides, instead of directly evaluating the number of reduced mutants, the execution time after reduction can be used as an alternative [8]. The meaning of these indicators is clear: the more reduction, the less cost. In the literature, a tiny minority of studies are evaluated only by using reduction ratio [22]. Considering that the meaning of the indicators for the first dimension do not have any issue, this paper will focus on the indicators for the second dimension.

To hit the second dimension, the **global mutation score (GMS)** is one of the indicators in existing studies [4, 6–10, 27]. They hold the idea that mutation reduction is to maintain the effectiveness of the mutation set (i.e., the number of unique faults that the set contains) while eliminating redundant mutants [27]. To evaluate the effectiveness of a mutant set, the global mutation score is used as the proxy. The global mutation score is calculated as follows:

- First of all, an adequate test suite that can kill all killable mutants should be prepared for evaluation. For the preparation, researchers should identify the equivalent mutants and remove them. After that, some test cases have to be created by hand to achieve 100% mutation score [4]. Another simple but dubious approach is to remove all surviving mutants when evaluating [27]. When an adequate test suite is obtained, the redundant test cases which only kill mutants that can be killed by other test cases should be removed.
- Second, to evaluate the reduced mutant set, a minimal set of test cases is selected among all test cases to kill the reduced mutant set. Assume that T is the test suite on the SUT with the original mutant set M. For a target mutant reduction strategy S, let M_s be the set of the mutants reduced from M. From the original test suite T, we keep deleting one test case which kills the least mutants while keeping the remaining test cases to kill M_s. After that, we can get a minimal test suite T_s. This is the reverse greedy algorithm given in [27]: "The computation of the minimum test suite is an instance of the problem of set cover. It is one of the well-known NP-Complete problems, and hence the minimum test suite can only

be approximated". As a result, there is no guarantee that the minimal one that meets the requirements will be selected. We use this algorithm to achieve an approximation.

- Third, run the above test suite T_s against the original mutant set to compute a mutation score as the global mutation score (GMS). Normally, *GMS* depicts the mutant killing ability of the test set T_s associated with a reduction strategy. The higher the mutant killing ability of the test set T_s is, the higher *GMS* a reduction strategy has.

Beside GMS, the variation in mutation score (i.e., $VMS = |MS(M, T) - MS(M_s, T)|$) is an alternative in existing research [1, 11, 20, 26]. For the existing studies, the smaller the variation, the better the effect is considered. For example, Zhang et al. held the idea that a perfect strategy should have the same mutation score on sampled mutants and all original mutants before sampling [20]. The meaning of this indicator seems to be justified: the smaller the variation, the smaller the change in the "absolute" evaluation on the given test suite. If a reduction strategy can provide a small variation on most experimental projects, in practice, we have a reason to believe that for any test suite, the "absolute" evaluation deviation caused by the reduction is small.

As can be seen, *GMS* and *VMS* do not explicitly measure the degree to which a mutation reduction strategy can maintain the relative order of the effectiveness among test suites. That is to say, they may answer the questions: What percentage of unique faults are retained in the reduced mutant set, and does the "absolute" evaluation on a given test suite change significantly before and after reduction? However, they cannot answer the question: Does the "relative" effectiveness order among test suites change significantly before and after reduction? As we mentioned before, "by mutation score, developers can not only know the 'absolute' effectiveness of a given test suite, but also know the 'relative' order of the effectiveness among different test suites." In the next section, we will figure out the pitfalls of ignoring the "relative" effectiveness measurement and explore the feasible solutions.

3 MUTANT REDUCTION EVALUATION: ESSENCE, PITFALLS, AND SOLUTIONS

After mutation reduction, a new mutation score metric will be obtained. In the context of test suite effectiveness evaluation, practitioners will use the new mutation score to replace the original mutation score metric to distinguish between a stronger and a weaker test suite. At this time, practitioners cannot help asking: how well will the new metric accurately hit the aim of measuring test suite effectiveness as the original metric? This problem is naturally raised by mutation reduction, regardless of which strategy is used to reduce the original mutants. In our community, the current practice is to employ GMS or VMS to evaluate the effectiveness of how a mutant subset represents the original mutant set. As a result, it is natural to believe that a reduction strategy with a higher GMS or VMS should lead to a more accurate new mutation score metric. Is it true in the context of test suite effectiveness evaluation? In this section, we first analyze the essence of mutation reduction evaluation from the viewpoint of measurement theory. Then, we point out the pitfalls of the existing reduction evaluation indicators. Finally, we propose our solutions to mutation reduction evaluation.

3.1 Essence of Mutation Reduction Evaluation

For the sake of simplicity, we make the following assumptions for the mutation reduction evaluation problem: (1) M is the original mutation set; (2) S is a mutant reduction strategy; and (3) M_s is the resulting reduced mutant set by S. In this context, there is a need to use an indicator ***REI* (Reduction Evaluation Indicator)** to evaluate the reduction effectiveness of S. With such an indicator *REI*, practitioners could answer the following important question: How well is

the reduced mutant set M_s in maintaining the ability of test suite effectiveness evaluation of the original mutant set M?

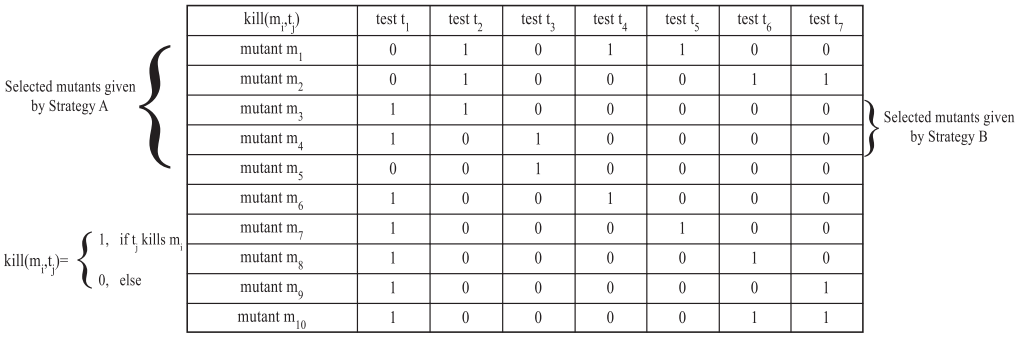
Under the given original mutant set M, for each test suite T, we can compute the corresponding mutation score $MS(M, T)$. From the viewpoint of measurement theory, mutation score MS is a mapping (i.e., metric) $MS(M, .)$: $E \rightarrow V$ which yields for each test suite $T \in E$ a measurement value $MS(M, T) \in V$. If $MS(M, .)$ is a valid metric, this means that there is a scale $<ERS, NRS, MS>$. Here, $ERS = (E, \preceq)$ is an empirical relationship system, where E is a set of test suites and $\preceq$ is a binary empirical relationship on E. $NRS = (V, \leq)$ is a numerical relationship system, where V is the set of real numbers and $\leq$ is a binary numerical relationship on V. In particular, $MS(M, .)$ satisfies the following representation condition: $T1 \preceq T2 \Leftrightarrow MS(M, T1) \leq MS(M, T2)$ for all $T1, T2 \in E$. In other words, the binary empirical relation $\preceq$ is preserved in V by $MS(M, .)$.

What does mutation reduction mean from the viewpoint of measurement theory? Assume that $MS(M, .)$ is a valid metric, and the binary relationship $\preceq$ denotes "less effective than or as effective as" with respect to test effectiveness. Furthermore, assume that the mutation reduction strategy S selects a subset M_s from the mutation set M. Consequently, we will obtain a new mutation score metric $MS(M_s, .)$. Recall that the purpose of mutation reduction is to reduce the computation cost of mutation score but maintain the ability of test suite effectiveness evaluation (abbreviated as "reduce computation cost but maintain evaluation ability"). Ideally, we hope that $MS(M_s, .)$ is also a valid metric, i.e., $T1 \preceq T2 \Leftrightarrow MS(M_s, T1) \leq MS(M_s, T2)$ holds for all $T1, T2 \in E$. In other words, the binary empirical relation $\preceq$ is also preserved in V by $MS(M_s, .)$. In this case, $MS(M, .)$ and $MS(M_s, .)$ produce the same numerical relationship on V. This means that, compared with $MS(M, .)$, $MS(M_s, .)$ causes a zero-loss in the ability of test suite effectiveness evaluation.

However, in practice, it is possible that, after mutation reduction, a loss will be caused in the ability of test suite effectiveness evaluation. In this case, how to evaluate the effectiveness of a mutation reduction strategy S? A natural idea is to compare the numerical relationships on V produced by $MS(M, .)$ and $MS(M_s, .)$. If there is a large difference, this means that the numerical relationship produced by $MS(M, .)$ is largely twisted by mutation reduction, i.e., there is a large change in the ability to evaluate test suite effectiveness. If there is a small difference, this means that $MS(M, .)$ and $MS(M_s, .)$ have a similar ability to evaluate test suite effectiveness. Given this situation, we can use the difference between the numerical relationships on V before and after reduction to measure the effectiveness of the mutation reduction strategy S: the smaller the difference is, the more effective the reduction strategy is. Since the numerical relationship in our context is a binary relationship, the essence of mutation reduction evaluation is to measure the "order-preserving ability", i.e., to what extent the mutation score order among test suites is maintained after mutation reduction.

3.2 Pitfalls of Existing Evaluation Indicators

According to Section 3.1, the essence of mutation reduction evaluation is to measure the extent of change in mutation score order among test suites before and after mutation reduction. During this measurement, the mutation score order among test suites before mutation reduction is regarded as the ground truth. However, the existing *REI*s such as *GMS* and *VMS* do not directly measure such a change in mutation score order among test suites. In contrast, for a mutation reduction strategy S, they quantify either the ability of the tests killing the reduced mutants to kill the original mutants (*GMS*) or its influence on the mutation score of the same test suite (*VMS*). Clearly, if S has a high *GMS* and/or *VMS*, this does not necessarily imply that it also has a high order-preserving ability. In other words, it is not possible to use *GMS* and *VMS* to determine the extent to which the partial order under the original mutants remains unchanged in the partial order under the reduced mutants. Consequently, misleading conclusions could be possibly achieved when using



kill(m_i,t_j)	test t_1	test t_2	test t_3	test t_4	test t_5	test t_6	test t_7
mutant m_1	0	1	0	1	1	0	0
mutant m_2	0	1	0	0	0	1	1
mutant m_3	1	1	0	0	0	0	0
mutant m_4	1	0	1	0	0	0	0
mutant m_5	0	0	1	0	0	0	0
mutant m_6	1	0	0	1	0	0	0
mutant m_7	1	0	0	0	1	0	0
mutant m_8	1	0	0	0	0	1	0
mutant m_9	1	0	0	0	0	0	1
mutant m_{10}	1	0	0	0	0	1	1

Fig. 1. An example to show the misleading of GMS.

the existing *REI*s to evaluate the reduction effectiveness. This is especially true when comparing the effectiveness of multiple reduction strategies.

3.2.1 Pitfalls of Global Mutation Score. We use the example in Figure 1 to reveal the pitfalls of *GMS*. At a high level, on the one hand, we will show that, by *GMS*, a wrong conclusion will be drawn compared with the conclusion drawn from the "order-preserving ability". On the other hand, by using the minimal test suite to compute *GMS*, we will show that *GMS* is indeed not the proxy of the effectiveness of a mutant set.

Figure 1 shows a kill relationship matrix for 10 distinguished mutants and seven test cases. The matrix reflects the kill relationship between test cases and mutants: kill(m, t) = 1 if and only if t kills m. In this example, the original mutation set $M = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}\}$ and the original test suite $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. Assume that there are two mutation reduction strategies A and B: according to A, the reduced mutant set $M_A = \{m_1, m_2, m_3, m_4, m_5\}$; according to B, the reduced mutant set $M_B = \{m_3, m_4\}$.

First, we analyze whether *GMS* can depict the "order-preserving ability". On the one hand, it is a fact that A must not have a lower "order-preserving ability" than B. This is because for any test suite pair distinguished by the original mutant set, if it can be distinguished by M_B, it must be distinguished by M_A. On the other hand, due to the fact that the minimal test set for A is $T_A = \{t_2, t_3\}$ and the minimal test set for B is $T_B = \{t_1\}$,[3] we have:

$$GMS\,(A) = \frac{|KM\,(M, T_A)|}{|M|} = \frac{|KM\,(M, \{t_2, t_3\})|}{|M|} = \frac{|\{m_1, m_2, m_3, m_4, m_5\}|}{|M|} = 50\% \tag{4}$$

$$GMS\,(B) = \frac{|KM\,(M, T_B)|}{|M|} = \frac{|KM\,(M, \{t_1\})|}{|M|} = \frac{|\{m_3, m_4, m_6, m_7, m_8, m_9, m_{10}\}|}{|M|} = 70\% \tag{5}$$

Therefore, according to *GMS*, the reduction strategy B is more effective than the reduction strategy A. As a result, *GMS* is unable to reveal the fact that A must not have a lower "order-preserving ability" than B.

Second, we show that *GMS* is indeed not the proxy of the effectiveness of a mutant set. In the literature, it is assumed that the higher *GMS* a strategy has, the better the resulting mutant subset represents the original mutant set. In the last decades, this assumption has been the basis for evaluating and comparing the effectiveness of mutation reduction strategies. However, such an assumption is indeed problematic. For example, from $GMS(A) < GMS(B)$, by definition, we can

[3]Note that, in practice, for a given mutation reduction strategy, the corresponding minimal test suite is obtained by an approximation algorithm. However, in this example, the minimal suites obtained are actually the minimum suites. As a result, the pitfall in this example is not caused by the approximation algorithm.

	kill(m_i,t_j)	test t_1	test t_2	test t_3	test t_4
Selected mutants given by Strategy C	mutant m_1	1	0	0	0
	mutant m_2	0	0	1	0
Selected mutants given by Strategy D	mutant m_3	1	1	1	0
	mutant m_4	1	1	1	1
	mutant m_5	0	0	0	1

$$\text{kill}(m_i,t_j)=\begin{cases}1, & \text{if } t_j \text{ kills } m_i\\ 0, & \text{else}\end{cases}$$

Fig. 2. An example to show the misleading of VMS.

conclude that: M_A contains fewer unique faults than M_B. However, due to $M_A \supset M_B$, we have the fact that M_A must not contain fewer unique faults than M_B. Therefore, it is problematic to use *GMS* to characterize the effectiveness of a reduced mutant set (i.e., the number of unique faults that the set contains). This indicates that, when using *GMS* as an evaluation indicator, a misleading conclusion may be drawn.

Why does the GMS(S) make mistakes? Normally, for a mutation reduction strategy *S*, *GMS*(*S*) depicts the mutant killing ability of the test set T_s. The larger $|KM(M, T_s)|$ is, the higher (i.e., better) *GMS*(*S*) is. However, as we show, *GMS*(*S*) is indeed not the proxy of effectiveness of a mutant set. Another main reason is that the aim of *GMS*(*S*) is not directly related to the "order-preserving ability". In other words, a higher *GMS*(*S*) does not necessarily mean a higher "order-preserving ability".

3.2.2 Pitfalls of Variation in Mutation Score. We use the example in Figure 2 to reveal the pitfalls of *VMS*. Figure 2 shows a kill relationship matrix for five mutants and four test cases. In this example, the original mutation set is $M = \{m_1, m_2, m_3, m_4, m_5\}$ and the original test suite is $T = \{t_1, t_2, t_3, t_4\}$. Assume that there are two mutation reduction strategies *C* and *D*: according to C, the reduced mutant set is $M_C = \{m_1, m_2\}$; according to *D*, the reduced mutant set is $M_D = \{m_3, m_4\}$. Now, the problem we need to answer is: Is *C* or *D* more effective in mutation reduction? As can be seen, *MS*(*M*, .), *MS*(M_C, .), and *MS*(M_D, .) are indeed three metrics corresponding to the original mutation set *M*, the reduced mutant set M_C, and the reduced mutant set M_D, respectively. According to Section 3.1, if a reduction strategy leads to a metric that has a high "order-preserving ability", it will be regarded as a good reduction strategy. Therefore, in order to compare the reduction effectiveness of C and *D*, we need to analyze the "order-preserving ability" of *MS*(M_C, .) and *MS*(M_D, .) compared with *MS*(*M*, .). Before this analysis, there is a need to know the mutation score order among test suites by *MS*(*M*, .). To this end, we leverage the original test suite T to generate all possible non-empty subsets, i.e., $2^4 - 1 = 15$ non-empty subsets. Figure 3(a) describes the mutation score order under *MS*(*M*, .) among the test suites that have a subset relationship.[4] For each arrow, the test suite on the head is a subset of the test suite on the tail. In particular, a green arrow indicates that the test suite on the tail has a higher mutation score, while a yellow arrow indicates that these two test suites have the same mutation score. Figures 3(b) and 3(c) respectively describe the mutation score order under *MS*(M_C, .) and *MS*(M_D, .) among the test suites that have a subset relationship. For a pair of test suites with a subset relationship, if the mutation score order is changed compared

[4]It is worth noting that we limit our analysis to the test suites having a subset relationship, as this can ensure that the mutation score order among test suites under *MS*(*M*, .) can be regarded as the ground truth.

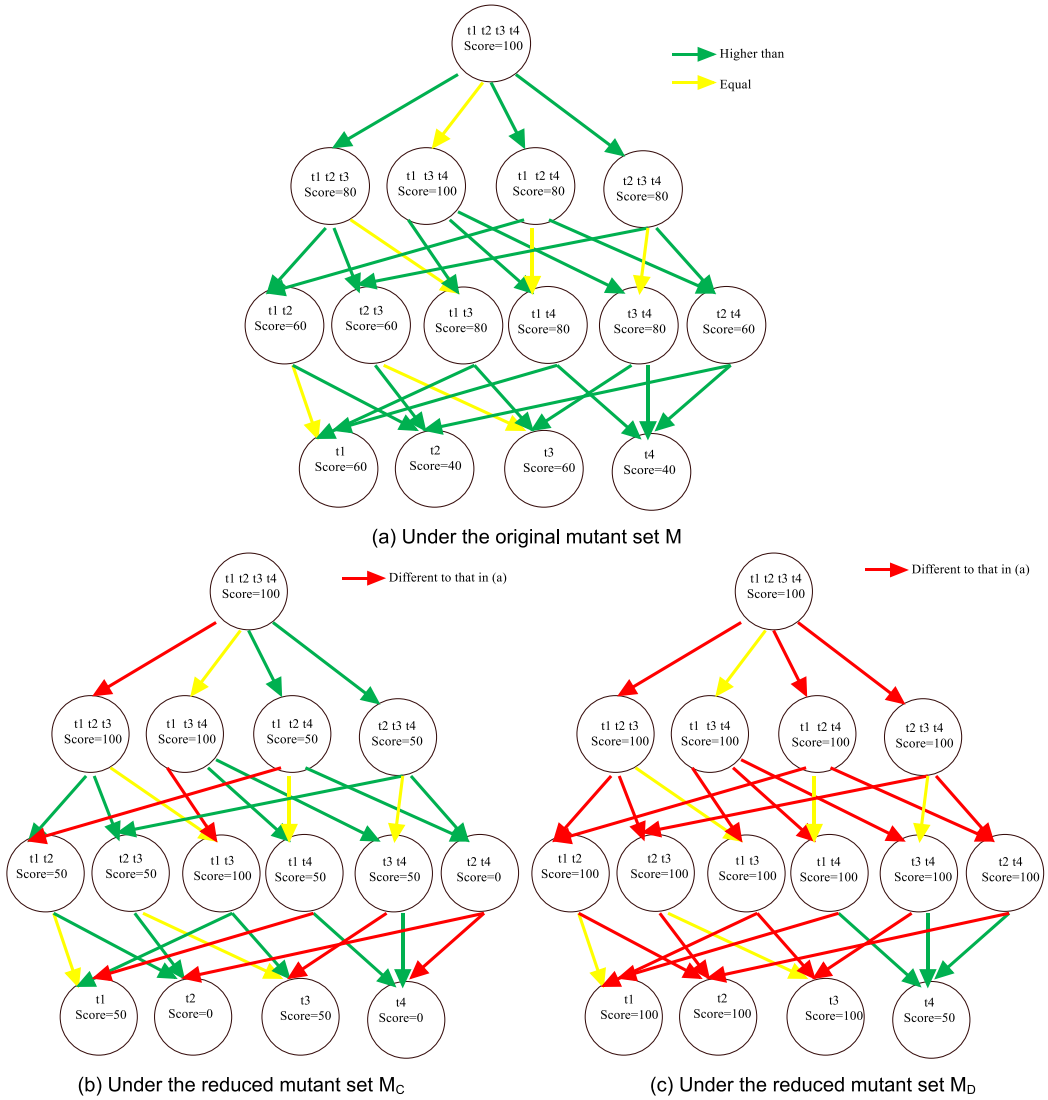


Fig. 3. The visualization of mutation score order among test suites (generated from the test suite shown in Figure 2) under different mutant sets.

with that under $MS(M, .)$, the corresponding arrow is shown in red. When comparing Figure 3(b) with Figure 3(a), we can find that 7 out of 28 arrows are red. When comparing Figure 3(c) with Figure 3(a), we can find that 19 out of 28 arrows are red. Therefore, from the viewpoint of "order-preserving ability", both C and D are not perfect, but C is a more effective reduction strategy than D.

In the following, we will use this example to investigate whether *VMS* could achieve the same conclusion. From Figure 2, we can see that: $VMS(C) = |\text{MS}(M, T) - MS(M_C, T)| = |1 - 1| = 0$ and $VMS(D) = |\text{MS}(M, T) - MS(M_D, T)| = |1 - 1| = 0$. Therefore, according to *VMS*, both C and D are perfect in maintaining the ability of test suite effectiveness evaluation, and C is as effective as D. Compared with the conclusion drawn from the "order-preserving ability", we can find that: on the

one hand, *VMS* is unable to reveal the fact that both $MS(M_C, .)$ and $MS(M_D, .)$ twist the mutation score order among test suites under $MS(M, .)$; on the other hand, *VMS* is unable to reveal the fact that $MS(M_C, .)$ has a higher "order-preserving ability".

Why does the variation in mutation score go wrong? Indeed, a mutation reduction strategy that can keep the mutation score unchanged is, of course, the best. However, it is worth noting that the mutation score must be maintained for ALL test suites. That is to say, when using mutation score to guide mutant reduction, instead of using (2), we should use:

$$\min \sum_{T} |MS(M_s, T) - MS(M, T)| \tag{6}$$

For a reduction strategy, if (6) holds, it is natural that the mutation score order under M is completely preserved under M_s, i.e., M_s has the same ability to evaluate test suite effectiveness as M. Obviously, such a requirement is strict, which means that it is hard to satisfy (6) in practice. Therefore, previous studies use *VMS* to depict the difference in mutation score before and after reduction. In particular, it is a common practice to compute VMS based on a number of test suites and believe that a smaller VMS indicates a better reduction strategy. However, there are two pitfalls in this practice. First, VMS may have a low ability to distinguish between different reduction strategies. As an example, if $MS(M, T) = 100\%$, this means that all the mutants in M can be killed by T. In this context, any mutant reduction strategy will lead to $MS(M_s, T) = 100\%$. Consequently, *VMS* is unable to distinguish between any two reduction strategies. This is a possible reason why prior studies report that many reduction strategies have the similar reduction effectiveness [29]. Second, more importantly, for a reduction strategy S, even if its *VMS* is small for all test suites, it does not necessarily mean that $MS(M_S, .)$ has a good "order-preserving ability". For example, for two test suites $T1$ and $T2$, assume that $MS(M, T1) = 90\%$, $MS(M, T2) = 89\%$, $MS(M_s, T1) = 89\%$, and $MS(M_s, T2) = 90\%$. As can be seen, although *VMS*s on $T1$ and $T2$ are both trivial, the mutation score order on $T1$ and $T2$ is inverse before and after mutation reduction. The fundamental reason for the above pitfalls, we believe, is that *VMS* aims to measure the "mutation-score-preserving ability" of a reduction strategy, which is not coincident exactly with the "order-preserving ability". In other words, a lower VMS does not necessarily mean a higher "order-preserving ability".

3.3 Proposed Evaluation Indicators: OP and EROP

As shown in Section 3.2, when evaluating the effectiveness of a reduction strategy, the existing mutation reduction evaluation indicators are unable to measure the "order-preserving ability" after reduction. As a result, they may lead to misleading conclusions. In order to tackle this problem, we next propose indicators to quantify the "order-preserving ability" for a mutation reduction strategy S. Assume that, by S, the original mutation set M on the SUT is reduced to a mutation set M_s. As a result, we have two mutation score metrics, $MS(M, .)$ and $MS(M_s, .)$. If the resulting reduced mutants are used to replace the original mutants for test suite effectiveness evaluation, it is very natural for practitioners to expect that this reduction should not twist the "relative" effectiveness order among test suites that have a partial order relationship in defect detecting ability. Otherwise, the resulting reduced mutants will lose their value in the evaluation of test suite effectiveness. To this end, at a high level, we first leverage the original test suite T to generate a group of test suites that have a partial order relationship defined by the subset relation.[5] Then, we generate two mutation score

[5]In the literature, for an SUT, it is often the case that only one test suite (i.e., the original test suite T) is provided. We use T to generate a group of test suites with a partial order relationship on test effectiveness. This ensures, on the one hand, our proposed evaluation indicators can be easily applied in practice, and on the other hand, the mutation score order produced by $MS(M, .)$ reflects the Score Monotonicity Property: adding test cases to a test suite does not decrease its mutation score [25], which can be used as the ground truth.

orders between these test suites that have a subset relationship: one from $MS(M, .)$, and another from $MS(M_s, .)$. Finally, we compare them to calculate to what extent the mutation score order produced by $MS(M, .)$ is changed with respect to the mutation score order produced by $MS(M_s, .)$. During this process, the mutation score order produced by $MS(M, .)$ with subset constraints is used as the ground truth. By this way, we know to what extent the ability to evaluate test suite effectiveness is changed after mutation reduction. Note that, the analysis of test suite effectiveness is limited to those test suites that have a "subset" relation, rather than arbitrary test suites.

During the above process, the computation complexity is mainly from the comparison of two mutation score orders produced by $MS(M, .)$ and $MS(M_s, .)$. In our context, a mutation score order is indeed a set of ($T1$, $T2$, operator) satisfying the following conditions: (1) $T2$ is a proper subset of $T1$; and (2) operator is ">" if $T1$ has a higher mutation score and is "=" if $T1$ has the same mutation score compared with $T2$. For each pair of test suites, we examine whether the operator on ($T1$, $T2$, operator) is changed before and after reduction. Assume that $T1$ is a subset of $T2$ and $T2$ is a subset of $T3$. Clearly, it is redundant to compare $T1$ with $T3$ if we have compared $T1$ with $T2$ and $T2$ with $T3$. In our study, in order to simplify the computation, we only take into account the following mutation score comparisons: $T1$ vs. $T2$ and $T2$ vs. $T3$. With the above constraint, how many pairs of test suites we need to examine at least for comparing the two mutation score orders produced by $MS(M, .)$ and $MS(M_s, .)$? Let P(n) be the minimum total number of pairs of test suites we need to examine, where n is the number of test cases in the original test suite T. Suppose that **SNES(T)** is the **Set of Non-Empty Subsets of T** (each element is a set of test cases, i.e., a test suite). As a result, we have $|\mathrm{SNES}(T)| = 2^n - 1$. For the sake of simplicity, if $T1$ is an element of SNES(T) and $T1$ consists of k test cases, then we call $T1$ a k-subset of T ($1 \le k \le n$). For each k-subset T1 with $k > 1$, there are k (k-1)-subsets, each one corresponding to one set by removing one single test case from $T1$. In this context, in order to compute P(n), for each k-subset with $k > 1$, we need to compare it against each of the corresponding k (k-1)-subsets, i.e., there are k comparisons involved in total. Since there are C_n^k k-subsets, therefore, we have:

$$P(n) = \sum_{k=2}^{n} k \times C_n^k \tag{7}$$

Through simplification, we can get:

$$P(n) = \sum_{k=2}^{n} k \times C_n^k = \sum_{k=2}^{n} n \times C_{n-1}^{k-1} = n \times \sum_{k=2}^{n} C_{n-1}^{k-1} = n \times (2^{n-1} - 1) \tag{8}$$

In the following, for the sake of simplicity, such a calculation approach is called a "continuous subsample" approach. Taking Figure 3(a) as an example, there are four 3-subsets, six 2-subsets, and four 1-subsets. As a result, 1*4 + 2*6 + 3*4 = 28 comparisons (i.e., arrows) are needed, which can be computed by $P(4) = 4 \times (2^3 - 1)$.

As shown in Equation (8), the computation of P(n) has an exponential time complexity, which hinders the quantification of the "order-preserving ability" for a mutation strategy in practice. For the sake of practicability, we need to reduce the number of comparisons. To this concern, for comparing two mutation score orders produced by $MS(M, .)$ and $MS(M_s, .)$, in this study, we hence propose to use the following way to generate a group of test suites that have a partial order relationship defined by the subset relation:

(1) $T_0 = T$, $i = 0$;
(2) If $|T_i| = 1$, then stop, else go to (3)
(3) Generate T_{i+1} by randomly selecting half of the test cases (round down) in T_i;

(4) i = i + 1;
(5) Go to (2).

Consequently, we obtain the following k test suites:

$$T_0 \supset T_1 \supset T_2 \supset \cdots \supset T_k \neq \emptyset$$

which satisfies:

$$|T_{i+1}| = int(|T_i| \times 0.5), i = 0, 1, \ldots, k-1$$

and

$$k = int(log_2|T_0|) = int(log_2 n)$$

For such a sequence of test suites, we only need to compare k times, far less than $n \times (2^{n-1} - 1)$, to determine the change in two mutation score orders produced by *MS*(*M*, .) and *MS*(M_s, .). After computing *MS*(*M*, T_i) and *MS*(M_s, T_i) for all i, we have:

$$MS(M, T_{i+1}) < MS(M, T_i) \textbf{ or } \mathbf{MS}(M, T_{i+1}) = MS(M, T_i)(i = 0, 1, \ldots, k-1)$$

and

$$MS(M_s, T_{i+1}) < MS(M_s, T_i) \textbf{ or } \mathbf{MS}(M_s, T_{i+1}) = MS(M_s, T_i)(i = 0, 1, \ldots, k-1)$$

Although there are two possible signs, there is only one sign in fact. We record the 2 × k signs according to the mutation score order. By the 2 × k relationships, we obtain the set of index of changed signs:

$$X = \{i | MS(M, T_{i+1}) < MS(M, T_i) \text{ and } MS(M_s, T_{i+1}) = MS(M_s, T_i)\} U$$

$$\{i | MS(M, T_{i+1}) = MS(M, T_i) \text{ and } MS(M_s, T_{i+1}) < MS(M_s, T_i)\}$$

However, indeed, if *MS*(*M*, T_{i+1}) and *MS*(*M*, T_i) are the same, *MS*(M_s, T_{i+1}) and *MS*(M_s, T_i) must be the same (i.e., the latter set above is an empty set). This is because if T_{i+1} and T_i cannot be distinguished by M, T_{i+1} and T_i must not be distinguished by a subset of M. For example, the yellow arrows in Figure 3(a) are still yellow in Figures 3(b) and 3(c). As a result, we have:

$$X = \{i | MS(M, T_{i+1}) < MS(M, T_i) \text{ and } MS(M_s, T_{i+1}) = MS(M_s, T_i)\}$$

As such, we propose to use the **Order Preservation (*OP*)** to measure the change in two mutation score orders produced by *MS*(*M*, .) and *MS*(M_s, .) as follows:

$$OP(S) = 1 - |X| / k \tag{9}$$

Note that Kendal Tau_b [21], a common indicator for rank correlation, is inappropriate for measuring the rank change in our paper. In our study, only two test suites with a subset relationship are comparable. For example, {t1} and {t2} are incomparable. Therefore, Tau_b cannot be directly applied in our context.

As can be seen, there is randomness when generating $T_1, \ldots, T_k$ from the given original test suite *T*. In order to reduce the influence of randomness, we recommend that the *k* test suites are generated 100 times, and the resulting average OP is used for evaluating the "order-preserving ability". In the following, when we talk about OP, we refer to the final OP computed by average. The higher OP value, the higher the "order-preserving ability" of a mutation reduction strategy (based on a given reduction ratio). In particular, for the sake of simplicity, such a calculation procedure will be called a "continuous half-sample" approach.

We next use the example in Figure 3 to demonstrate how to compute *OP*. At first, from the original test suite $T = \{t_1, t_2, t_3, t_4\}$, we generate the following sequence of test suites:

$$T0 = \{t_1, t_2, t_3, t_4\}, T1 = \{t_1, t_2\}, T2 = \{t_1\}.$$

Table 1. Mutation Scores Under Different Mutation Sets

MS (M_i, T_j)	M = {m_1, m_2, m_3, m_4, m_5}	MC = {m_1, m_2}	MD = {m_3, m_4}
$T0$	100%	100%	100%
$T1$	60%	50%	100%
$T2$	60%	50%	100%

Table 1 reports for each test suite in this sequent the mutation score before and after mutation reduction:

As can be seen, the mutation score order before reduction is:

$$\{(T0, T1, \text{“>”}),\ (T1, T2, \text{“=”})\}$$

The mutation score order after applying the mutation reduction strategy C is:

$$\{(T0, T1, \text{“>”}),\ (T1, T2, \text{“=”})\}$$

The mutation score order after applying the mutation reduction strategy D is:

$$\{(T0, T1, \text{“=”}),\ (T1, T2, \text{“=”})\}$$

As such, the sets of index of changed signs for strategy C and strategy D are, respectively, X(C) = {} and X(D) = {0}. Therefore, $OP(C) = 1 - 0/2 = 1$ and $OP(D) = 1 - 1/2 = 0.5$. By using OP, we can distinguish strategy C from strategy D. For this sequence, the conclusion is that the strategy C is better than the strategy D, which is consistent with our analysis in Section 3.2. If we use the variation in mutation score, we have VMS(C) and $VMS(D) = 0$, i.e., the variation in mutation score cannot distinguish strategy C from strategy D.

However, OP is a non-effort-aware evaluation indicator, as it does not take into account the number of reduced mutants. Indeed, a strategy that does not delete any mutant will have an OP of 1.0 (i.e., the maximum). To tackle this problem, in this study, we propose an effort-aware evaluation indicator **Effort-aware Relative Order Preservation (EROP)** to measure the “order-preserving ability” as follows:

$$EROP(S) = RR(S) * ROP(S) \tag{10}$$

Here, RR(S) is the mutation reduction ratio, while **ROP**(S) = OP(S) − OP($random$) is the **Relative Order Preservation** ability of a reduction strategy S compared with a random reduction strategy that selects the same number of mutants. The higher EROP value, the better effectiveness of a strategy. If the value is negative, the strategy is useless, as it is not superior to a random reduction strategy. Similar to OP, we also recommend that the k test suites are generated 100 times to compute the average EROP when evaluating the “order-preserving ability” of a reduction strategy S in practice.

In summary, in this study, we propose two indicators, OP and EROP, for evaluating the “order-preserving ability” of a mutation reduction strategy. The former is non-effort-aware, while the latter is effort-aware. For a SUT with the original mutant set M and the original test suite T, OP and EROP can be efficiently computed. In the following sections, we empirically investigate the actual usefulness of OP and EROP, especially when used to compare the effectiveness of different reduction strategies. Note that OP and EROP are not limited to the scenario we analyzed above. For any two mutation score metrics, the corresponding OP can be calculated after the ground truth metric is specified.

Table 2. The Information of Programs

	Program/Project	LOCs	Path	Tests	Mutants	MS
J1	Crypt	25	org.apache.commons.codec.digest	12	11	1.000
J2	StringUtils	48	org.apache.commons.codec.binary	172	39	0.974
J3	Md5Crypt	107	org.apache.commons.codec.digest	16	62	0.887
J4	DefaultParser	178	org.apache.commons.cli	64	109	0.929
J5	Option	192	org.apache.commons.cli	303	111	0.800
J6	ArithmeticUtils	207	org.apache.commons.math3.util	307	213	0.816
J7	ResizableDoubleArray	217	org.apache.commons.math3.util	60	149	0.743
J8	UnixCrypt	311	org.apache.commons.codec.digest	8	215	0.967
J9	HelpFormatter	416	org.apache.commons.cli	31	134	0.881
J10	Dfp	1702	org.apache.commons.math3.dfp	159	1161	0.954
P1	fileupload	2496	org.apache.commons.fileupload	65	7056	0.459
P2	pool	8138	org.apache.commons.pool2	281	10467	0.622
P3	validator	9026	org.apache.commons.validator	551	20204	0.340
P4	net	14150	org.apache.commons.net	2281	56898	0.245
P5	beanutils	16495	org.apache.commons.beanutils2	1215	24407	0.555
P6	jexl	18533	org.apache.commons..jexl3	460	102718	0.394
P7	collections	33670	org.apache.commons.collections4	2500	63593	0.282

4 EXPERIMENTAL SETUP

In this section, we describe in detail the experimental design. First, we report the programs and tools used in our study. Then, we introduce the investigated mainstream mutation reduction strategies and their implementations. Finally, we list the research questions under study.

4.1 Subject Items and Tools

Programs and projects. As shown in Table 2, ten Java programs and seven Java projects, which are from Apache projects, are used for the experiments [28]. We selected all projects in [28] which are successfully built and tested for study, as they were widely used in previous work [8, 11, 26, 27, 29–31]. The projects involved are *codec, commons, math, fileupload, pool, validator, net, beanutils, jexl*, and *collections*. These projects provide widely used APIs. For the programs, they range in size from dozens to thousands of lines. In particular, as indicated by mutation score, there is a relatively sufficient set of test cases against each program. With these subjects, we are able to investigate the usefulness of our proposed indicators.

Mutants. The tool we use to generate and run the mutants is PIT 1.4.3 [32], which is a widely used tool for Java mutation testing. To generate the mutants, seven default mutators are used for ten programs, while all 29 mutators are used for seven projects (see Section 4.2.3 for details). Here we use two mutator settings based on the following consideration: Sentinel is built based on the default mutators [8]. On the other hand, using seven default mutators for ten programs enables us to conduct a fair comparison between our experimental results and those reported in [8]. On the other hand, using all 29 mutators for seven projects enables us to mitigate the issues related to the reduced operator set used by PIT. From Table 2, on the one hand, we can see that the ten programs have a high mutation score, suggesting their test suites are effective. On the other hand, we can see that the seven projects have a relatively low mutation score. The main reason is that we use more mutators and do not filter the equivalent mutants.

It is worth noting that one of the aims of our experiment is to evaluate various mutation reduction strategies. Therefore, we need to execute all the mutants. After the mutants are executed,

not only the mutation score but also the detail of which tests are executed can be obtained by the PIT reports for each program. Table 2 reports the number of executed tests. In the execution process, whether each test case can kill a mutant is recorded and used to obtain the kill relationship matrix (e.g., Figure 1). This matrix is used only when evaluating the strategies, and all the studied strategies are not implemented on this information.

Coverage. The tool we use to collect the coverage information is Cobertura [33], which is a widely used coverage tool. All the tests recorded by PIT reports will be executed against SUT by Cobertura to obtain the coverage relationship matrix, which records whether each test case can cover a mutant or not. In the following, we say a test case covers a mutant if and only if the test case covers the corresponding statement to be mutated in the original program SUT.

Tests. For each target item, the manually written test suite provided by Apache projects according to the SUT is the raw test suite. However, not all test cases in the raw test suite are used. In our study, only the test cases which cover the target program and execute successfully by PIT are used.

4.2 Five Mutation Reduction Strategies and their Implementations

We use **Random Mutant Selection (RMS), Sentinel, Certain Operator Selection (COS), Subsuming Mutant Selection (SMS)**, and **Clustering Mutant Selection (CMS)** as the mainstream mutation reduction strategies in the experiment. Given a SUT, the corresponding mutants, and a test suite, since the uncovered mutants must be alive, we first use the coverage information to filter out uncovered mutants which do not need to be executed. Then, we implement a strategy by the following specific steps.

4.2.1 RMS. Random Mutant Selection (RMS) selects a specified number or proportion of mutants from all mutants randomly. To implement RMS, given the number of selected mutants n, we keep randomly selecting one of the remaining mutants with equal probability until there are n mutants selected.

4.2.2 Sentinel. Sentinel is a state-of-the-art multi-objective evolutionary hyper-heuristic approach on mutant reduction [8]. For the "multi-objective evolutionary", it has two objective functions. Normally, the first objective is to maximize the average "absolute" strategy effectiveness of s over n runs (as there is randomness in s):

$$\uparrow \text{SCORE (s)} = \frac{\frac{1}{n}\sum_{i=1}^{n} MS\,(M, T_s\,(i))}{MS\,(M,\ T)} = \frac{1}{n}\sum_{i=1}^{n} \frac{|KM\,(M, T_s\,(i))|}{|KM\,(M, T)|}$$

As the $|KM(M, T)|$ can be regarded as a constant, this objective can be regarded as maximizing the mutant killing ability of the test sets $T_s(\mathrm{i}), i = 1, \ldots, \mathrm{n}$. The second objective is to minimize the execution time. Instead of acting over the search space of mutants, Sentinel searches the heuristic space for good mutant reduction strategies that can achieve the best score of objective functions on a training set. With several pre-defined basic mutant selection strategies (i.e., Random Mutant Selection, Random Operator Selection, and Certain Operator Selection), Sentinel searches a combination of basic strategies (e.g., randomly select two mutators to generate mutants and then select 10 mutants randomly among them) which can achieve the best values of the objective functions. After that, we can use the combination of basic strategies for selecting mutants on a target project. In [8], Guizzo et al. focused on cross-version mutant reduction. However, it is laborious to find a suitable training project for a target project. On the other hand, cross-version information for Sentinel is unfair to other strategies. As a result, we use the example project *Triangle* provided in the Sentinel's source code as the training data. At a high level, Sentinel provides a combined reduction

Table 3. The 5-Operators Selection (in red) Among 22 Mutators

Mutator	Description
AAR	array reference for array reference replacement
ABS	absolute value insertion
ACR	array reference for constant replacement
AOR	arithmetic operator replacement
ASR	array reference for scalar variable replacement
CAR	constant for array reference replacement
CNR	comparable array name replacement
CRP	constant replacement
CSR	constant for scalar variable replacement
DER	DO statement end replacement
DSA	DATA statement alterations
GLR	GOTO label replacement
LCR	logical connector replacement
ROR	relational operator replacement
RSR	RETURN statement replacement
SAN	statement analysis
SAR	scalar variable for array reference replacement
SCR	scalar for constant replacement
SDL	statement deletion
SRC	source constant replacement
SVR	scalar variable replacement
UOI	unary operator insertion

strategy by searching the strategy space from the training project, which considers the execution cost and the GMS at the same time.

In our study, in order to implement Sentinel, given the number of selected mutants n, it can be used as a hyper parameter for training of Sentinel. That is to say, by setting this hyper parameter for the out-of-the-box Sentinel and after the training process, Sentinel can provide a combination of basic strategies that selects n mutants on the training program *Triangle*. Then, for a target program, we apply the combined strategy to the target program to select n mutants. It is worth noting that if the number of remaining mutants is going to be less than n after a basic strategy is executed, we will skip the basic strategy. For example, we set $n = 4$ and obtain the combined reduction strategy on Triangle: first, remove the mutator which will generate the most mutants; second, generate the mutants; third, select 4 mutants randomly. When we use this strategy on a target mutant set generated by only one mutator, none of the mutants will be generated after removing "the mutator which will generate the most mutants". As a result, we will skip this basic strategy and select 4 mutants randomly. We will discuss more details of Sentinel in Section 6.

4.2.3 COS. Certain Operator Selection (COS) selects the mutants generated by a specific subset of all mutators. The 5-Operators Selection proposed by Offutt et al. [4] is the approach chosen in this paper. Table 3 summarizes the five mutation operators (shown in red color) selected from 22 mutation operators with the description. The five operators include the relational, logical, arithmetic, absolute, and unary insertion operators. For PIT, we remove *Return Values Mutator* and *Void Method Calls Mutator* among the seven default mutators (i.e., *Conditionals Boundary Mutator, Increments Mutator, Invert Negatives Mutator, Math Mutator, Negate Conditionals Mutator, Return Values*

Table 4. The implementation of COS for PIT

Default Mutators In PIT	Type	Other "5-Operators Selection" Mutators In PIT	Type
ConditionalsBoundaryMutator	ROR	RemoveConditionalsMutator	LCR, ROR
IncrementsMutator	AOR	ExperimentSwitchMutator	ROR
InvertNegsMutator	UOI, ABS	NegationMutator	ABS
MathMutator	AOR, LCR	ArithmeticOperatorReplacement Mutator	AOR
NegateConditionalsMutator	ROR	Bitwise Operator Mutator	LCR
ReturnValsMutator		RelationalOperatorReplacement Mutator	ROR
VoidMethodCallMutator		UnaryOperatorInsertionMutator	UOI

Mutator, and Void Method Calls Mutator) as the implementation of COS on 10 programs (as shown in Table 4). On seven projects, we select *Conditionals Boundary Mutator, Increments Mutator, Invert Negatives Mutator, Math Mutator, Negate Conditionals Mutator, Remove Conditionals Mutator, Experiment Switch Mutator, Negation Mutator, Arithmetic Operator Replacement Mutator, Bitwise Operator Mutator, Relational Operator Replacement Mutator* and *Unary Operator Insertion Mutator* among all 29 mutators as the implementation of COS. It is worth noting that these seven-default operators were called the "defaults" mutators in the paper of Sentinel, and we hence reuse this abbreviation. However, on the current PIT website [34], they are called "old-defaults" mutators. The details about "ALL" mutators can be found on the website. We report the name of each used mutator with type, which is convenient for readers to reproduce and compare.

In our study, in order to implement COS, given the number of selected mutants n, we first select the mutants generated by certain mutators. Then, we keep randomly selecting one of the remaining mutants with equal probability until there are n mutants selected. It is worth noting that COS can also make the remained number of mutants less than n. In such a situation, we will exclude the COS strategy from subsequent analysis.

4.2.4 SMS. Subsuming Mutant Selection (SMS) selects the subsuming mutants. By the definition in [13], "One mutant subsumes another if at least one test kills the first and every test that kills the first also kills the second". For all mutants, killing the subsuming mutants is to kill all the killable mutants. Therefore, the subsumed mutants should be regarded as redundancy. In the following, we first introduce the relevant definitions and then elaborate on the implementation of SMS in our study.

Definition 1 (Subsuming). Let m_i and m_j be two mutants in M. We say that m_i subsumes m_j, if: (1) there exists some test cases kill m_i; and (2) for any test case in (1), it kills m_j.

Definition 2 (Non-subsumed set). The non-subsumed set is a subset of M. m_i is an element of non-subsumed set if and only if none of the other mutants among M subsumes m_i.

The goal of SMS is to find this non-subsumed set as the result of mutant selection. However, in order to determine the exact subsuming relationship, all mutants must be executed against all test cases, which is contrary to the aim of mutant reduction. As a result, how to approximate the kill relationship matrix between test cases and mutants (the matrix in Figure 1 is an example) is the main problem of SMS. In our study, we run the test suite on the original program SUT to obtain a coverage relationship matrix as the approximation of the kill relationship matrix. There are two reasons for us to take this as the approximation: on the one hand, covering is a necessary condition for killing, which reveals that the two matrixes are highly correlated; on the other hand, once the statement to be mutated is covered in SUT, the mutant will be covered (we say a test case covers a mutant if and only if the test case covers the corresponding statement to be mutated in the original

program). As a result, the one time running for the test suite against SUT can obtain the coverage relationship matrix between test cases and mutants. This is a low-cost approach to approximate the kill relationship matrix. As such, we define the coverage subsuming relationship between two mutants as follows:

Definition 3 (Coverage subsuming). Let m_i and m_j be two mutants in M. We say that m_i coverage-subsumes m_j, if: (1) there exists some test cases cover m_i; and (2) for any test case in (1), it covers m_j.

Definition 4 (Non-coverage-subsumed set). The non-coverage-subsumed set is a subset of M. m_i is an element of non-coverage-subsumed set if and only if none of the other mutants among M coverage-subsumes m_i.

Consequently, in our study, we define the Coverage-based Subsuming Mutant Selection to find this non-subsumed set as the result of mutant selection. In other words, we use the Coverage-based Subsuming Mutant Selection as the implementation of SMS. Specifically, Coverage-based Subsuming Mutant Selection consists of the following steps:

- Step 1: Execute the test suite on SUT (e.g., a program or a project) to collect the coverage information of the mutated statements corresponding to the mutants. By this step, a coverage relationship matrix is obtained;
- Step 2: Find non-coverage-subsumed set as the selected mutants.

It is worth noting that SMS can only select a specific number of mutants, which cannot be given in advance.

4.2.5 CMS. Clustering Mutant Selection (CMS) [24] selects one mutant from each mutant cluster which is proposed by Hussain [24]. In [24], first, all mutants were executed against all test cases to obtain a kill relationship matrix. Each row of the kill relationship matrix was an instance corresponding to a mutation. For example, in Figure 2, the instance of m1 is [1, 0, 0, 0]. Then, a clustering algorithm was applied on all mutant instances to classify the mutants into different clusters. Among each cluster, the mutants were guaranteed to be killed by similar test cases. Finally, the selection was obtained by randomly selecting one mutant from each cluster.

Since the kill relationship matrix is not available before executing all mutants, in our study, we use the coverage relationship matrix as the approximation. Specifically, in order to implement SMS, we use Coverage-based Clustering Mutant Selection consisting of the following steps in this paper:

- Step 1: Execute the test suite on SUT to collect a coverage information of the mutated statements corresponding to the mutants. By this step, the coverage relationship matrix is obtained;
- Step 2: Given the number of selected mutants n, a k-means algorithm is used to obtain n clusters by mutant instances from the coverage relationship matrix;
- Step 3: Randomly select one mutant from each cluster to obtain the selected mutants.

4.3 Research Questions

The RQs mainly focus on two aims: one is to investigate the distinguishing ability of mutation reduction evaluation indicators, while the other is to find the effective mutation reduction strategies. However, the two aims are not independent. First, we believe that there are differences between these reduction strategies. If an indicator makes all strategies look the same numerically on most programs or projects, it has a weak discrimination ability. If an indicator makes some strategies

ahead of the others numerically on most programs, it has a strong discrimination ability. On the whole, the two aims are achieved at the same time: by comparing the five strategies with different indicators, we will find the indicators with a strong discrimination ability as well as the effective strategies. We compare OP with global mutation score (GMS) and variation in mutation score (*VMS*). For EROP, on the one hand, we must examine the rationality of OP before verifying EROP. On the other hand, EROP is the first indicator that considers the number of reductions and the effectiveness of mutation reduction at the same time. As a result, we put EROP in the last RQ to investigate the effectiveness of reduction strategies.

RQ1 (Discrimination ability under a sufficient test suite): *Which is more reasonable as the indicator to evaluate reduction strategy, Order Preservation (OP), global mutation score (GMS), or the variation in Mutation Score (VMS), under a sufficient test suite?*

The purpose of RQ1 is to investigate which evaluation indicator can find the best reduction strategy under a sufficient test suite. In our study, each of the subject programs has a relatively sufficient test suite. For RQ1, we use 17 subjects (i.e., 10 programs and 7 projects) with their test suites (see Section 4.1 for detail) directly for the experiment without any change. Specifically, we apply the five strategies to each subject and calculate *VMS* and OP against the sufficient test suites. In particular, the above process is repeated 100 times to obtain the results for examining the influence of randomness on mutation reduction strategies. To compute GMS, a test suite with 100% mutation score should be used. Similar to previous studies [19, 27], we remove alive mutants before applying the reduction strategies. As shown in Section 2.3, the computation of GMS is based on a minimal test suite that can kill the reduced mutant set. However, there is randomness in the greedy algorithm for generating such a minimal test suite [27]. In our experiment, we run the greedy generation algorithm 10 times to compute 10 GMS values for examining the influence of randomness. The basic idea is that these strategies should not be difficult to distinguish by a sensitive indicator, and effective strategies will stand out on most projects. To see the difference, on the one hand, we list the average result on each subject. On the other hand, we draw the boxplot for each strategy on each subject.

Given a kill matrix, note that *SMS* always provides a certain number of selected mutants by the subsuming relationship. As a result, under RQ1, for each of the other strategies, we set the same number of selected mutants as the number of mutants selected by SMS. In particular, for CMS, we first implement SMS and then use the selection result as the initial clustering center of CMS. This is to make the initial cluster centers as different as possible.

RQ2 (Discrimination ability under an insufficient test suite): *Which is more reasonable as the indicator to evaluate reduction strategy, Order Preservation (OP) or the variation in Mutation Score (VMS), under an insufficient test suite?*

The purpose of RQ2 is to investigate whether the result of RQ1 will change or not under an insufficient test suite. The "change" here refers to two sub questions: (1) RQ 2.1: when researchers use an insufficient test suite to evaluate the reduction strategies, whether the conclusion is consistent with RQ1?, and (2) RQ 2.2: when researchers use an insufficient test suite to implement the strategies (i.e., SMS and CMS in this paper), whether the conclusion on OP is consistent with RQ1? Note that GMS is not compared in RQ2, since an adequate test suite which can kill all mutants should be prepared. In other words, GMS is an evaluation indicator only used under a sufficient test suite, and it is not appropriate to be compared in RQ2.

In order to simulate a relatively insufficient test suite, we delete test cases used in RQ1 with even index for each program or project. The "index" here is determined by the order in which it appears in the PIT report. After that we divide RQ2 into two sub questions. For RQ 2.1, on each subject, we perform the five reduction strategies with the insufficient test suite and calculate *VMS* and OP values against the insufficient test suite. The above process is repeated 100 times to obtain

the results for examining the influence of randomness on mutation reduction strategies. To see the difference, on the one hand, we list the average result for each subject. On the other hand, we draw the boxplot for each strategy with 100 *VMS* and 100 OPs against each subject. Note that, we take the same method as used in RQ1 to set the number of selected mutants and to implement CMS. For RQ 2.2, on each subject, we apply SMS and CMS with the insufficient test suite and calculate the OP value against the original test suite. The above process is repeated 100 times to examine the influence of randomness. To see the difference, on each subject, we compare the average result with that in RQ1. Note that, we take the same method as used in RQ1 to set the number of selected mutants and to implement CMS.

RQ3 (Influence of reduction ratio on reduction effectiveness): *How do the effectiveness of reduction strategies in terms of OP change under different mutation reduction ratios?*

The purpose of RQ3 is to investigate the influence of reduction ratio (i.e., RR in (3)) on reduction effectiveness. In the process of exploring this question, on the one hand, we can have a more comprehensive understanding of mutation reduction strategies, and on the other hand, we can know how much mutation reduction is the best. To the best of our knowledge, the quantitative analysis of reduction ratio is scarce in the literature. To answer RQ3, we set the reduction ratio from 90% to 10% with a step length of 10% for J1 while setting the reduction ratio from 95% to 0% with a step length of 5% for other 16 subjects. For each ratio, we first apply the feasible strategies to 10 programs and 7 projects for 100 times and calculate an average OP and an average EROP against the sufficient test suites. Then, with the OP values and EROP values under different ratios, we draw the OP-Reduction Ratio scatter diagram and EROP-Reduction Ratio scatter diagram of each strategy. It is worth noting that SMS can only be reduced to a fixed number. As a result, we also add a group of points under the specific reduction ratio (as shown in the last column in Table 5(a)) for showing SMS. Besides, COS has only a few data points due to the deletion of some mutators, which leads to fewer mutants for reduction. For CMS under RQ3, we use the default settings of *sklearn*[6] to generate the initial cluster centers. The k of k-means is set to the number of selected mutants in RQ1. For example, if there are four mutants selected in J1 under RQ1, CMS will first cluster mutants into four clusters for J1. Then, CMS randomly selects one mutant from each cluster in turn until x% mutants are selected.

5 EXPERIMENTAL RESULTS

In this section, we report in detail the experimental results on the ability of OP and EROP to distinguish mutation reduction effectiveness under sufficient and insufficient test suites, as well as the influence of mutation reduction ratio on their discrimination ability.

5.1 RQ1: Discrimination Ability Under a Sufficient Test Suite

Table 5 reports for five strategies the average OP, VMS, and GMS values over 100 times reduction results on each program and each project. The EROP is not shown in RQ1 and RQ2, which can be computed from the data in the table. In particular, for each of OP, VMS, and GMS, we applied the improved Scott-Knott **Effect Size Difference (ESD)** test (v 2.0) [39] to compute for each strategy a rank (as shown in brackets, a lower rank is better). The Scott-Knott ESD test clustered variables (the OPs or *VMS*s or GMSs of five strategies in our context) into different groups to obtain their ranks according to statistically significant differences in their mean variable importance scores ($\alpha = 0.05$). During this process, two groups were merged when a pair of statistically distinct groups had a negligible Cohen's d effect size [40]. By convention, $|d| < 0.2$ means "negligible", $0.2 \leq |d| < 0.5$

[6] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

Table 5. The Discrimination Ability Comparison between (a) OP, VMS and (b) GMS under a Sufficient Test Suite (the Strategy Rank by the Scott-Knott ESD test is Shown in Brackets and a Lower Rank is Better)

(a) OP vs. VMS

	OP					VMS					Reduction
Program/ Project	RMS	Sentinel	COS	SMS	CMS	RMS	Sentinel	COS	SMS	CMS	Ratio
J1	0.615(3)	0.270(4)	0.270(4)	0.872(1)	0.777(2)	0 (1)	0 (1)	0 (1)	0 (1)	0 (1)	7/11
J2	0.913(3)	0.920(3)	-	0.956(2)	0.970(1)	0.023(1)	0.020(1)	-	0.026(2)	0.026(2)	18/39
J3	0.403(4)	0.475(3)	0.313(5)	0.754(1)	0.697(2)	0.170(2)	0.158(2)	0.163(2)	0.161(2)	0.148(1)	58/62
J4	0.893(2)	0.851(3)	-	0.891(2)	0.936(1)	0.045(1)	0.092(2)	-	0.119(3)	0.046(1)	88/109
J5	0.698(3)	0.596(4)	0.586(5)	0.708(2)	0.724(1)	0.071(2)	0.296(5)	0.284(4)	0.075(2)	0.058(1)	96/115
J6	0.616(3)	0.513(5)	0.574(4)	0.700(2)	0.754(1)	0.104(1)	0.173(3)	0.110(1)	0.200(4)	0.144(2)	200/213
J7	0.620(3)	0.623(3)	0.604(4)	0.658(2)	0.699(1)	0.123(3)	0.130(3)	0.099(2)	0.013(1)	0.112(2)	141/149
J8	0.525(1)	0.528(1)	0.516(2)	0.531(1)	0.496(3)	0.063(1)	0.115(2)	0.081(1)	0.463(4)	0.161(3)	213/215
J9	0.649(3)	0.594(4)	0.654(3)	0.839(1)	0.749(2)	0.111(2)	0.175(3)	0.116(2)	0.093(1)	0.106(2)	126/134
J10	0.879(4)	0.892(3)	0.865(5)	0.947(2)	0.972(1)	0.023(1)	0.077(4)	0.026(1)	0.058(3)	0.036(2)	1103/1161
avg.	0.681	0.626	0.547	0.787	0.778	0.073	0.126	0.110	0.121	0.084	-
P1	0.681(3)	0.678(3)	0.690(3)	0.864(2)	0.885(1)	0.077(1)	0.079(1)	0.080(1)	0.229(3)	0.137(2)	7030/7056
P2	0.901(2)	0.899(2)	0.898(2)	0.850(3)	0.926(1)	0.035(1)	0.036(1)	0.035(1)	0.133(2)	0.177(3)	10347/10467
P3	0.948(2)	0.951(2)	0.955(1)	0.907(3)	0.959(1)	0.029(1)	0.029(1)	0.031(1)	0.151(3)	0.128(2)	20035/20204
P4	0.828(3)	0.827(3)	0.827(3)	0.908(2)	0.950(1)	0.032(1)	0.035(1)	0.035(1)	0.570(3)	0.494(2)	56785/56898
P5	0.966(2)	0.958(3)	0.967(2)	0.909(4)	0.972(1)	0.021(1)	0.023(1)	0.036(2)	0.238(4)	0.217(3)	24063/24407
P6	0.990(2)	0.984(2)	0.988(2)	0.937(3)	0.995(1)	0.029(1)	0.065(2)	0.030(1)	0.389(4)	0.341(3)	102552/102718
P7	0.900(2)	0.893(3)	0.871(4)	0.900(2)	0.973(1)	0.015(1)	0.015(1)	0.015(1)	0.524(3)	0.465(2)	62928/63593
avg.	0.888	0.884	0.885	0.896	0.951	0.034	0.040	0.037	0.319	0.280	-

(b) GMS

	GMS				
Program/ Project	RMS	Sentinel	COS	SMS	CMS
J1	0.703(5)	0.814(3)	0.850(2)	0.909(1)	0.784(4)
J2	0.822(3)	0.828(3)	-	0.986(1)	0.889(2)
J3	0.878(3)	0.898(2)	0.877(3)	0.982(1)	0.902(2)
J4	0.736(3)	0.622(4)	-	0.907(1)	0.832(2)
J5	0.705(3)	0.612(4)	0.702(3)	0.943(1)	0.799(2)
J6	0.688(3)	0.674(3)	0.688(3)	0.945(1)	0.744(2)
J7	0.664(3)	0.659(3)	0.662(3)	0.950(1)	0.722(2)
J8	0.978(1)	0.911(3)	0.977(1)	0.554(4)	0.941(2)
J9	0.731(3)	0.679(4)	0.716(3)	0.945(1)	0.817(2)
J10	0.865(3)	0.864(3)	0.848(4)	0.998(1)	0.940(2)
avg.	0.777	0.756	0.790	0.912	0.837
P1	0.825(3)	0.833(3)	0.826(3)	0.975(1)	0.885(2)
P2	0.779(3)	0.768(4)	0.763(4)	0.952(1)	0.887(2)
P3	0.761(3)	0.748(4)	0.739(4)	0.935(1)	0.811(2)
P4	0.750(3)	0.748(3)	0.727(4)	0.890(1)	0.769(2)
P5	0.790(3)	0.774(4)	0.776(4)	0.960(1)	0.878(2)
P6	0.635(3)	0.593(4)	0.631(3)	0.784(1)	0.690(2)
P7	0.769(3)	0.750(4)	0.724(5)	0.965(1)	0.872(2)
avg.	0.758	0.744	0.741	0.923	0.827

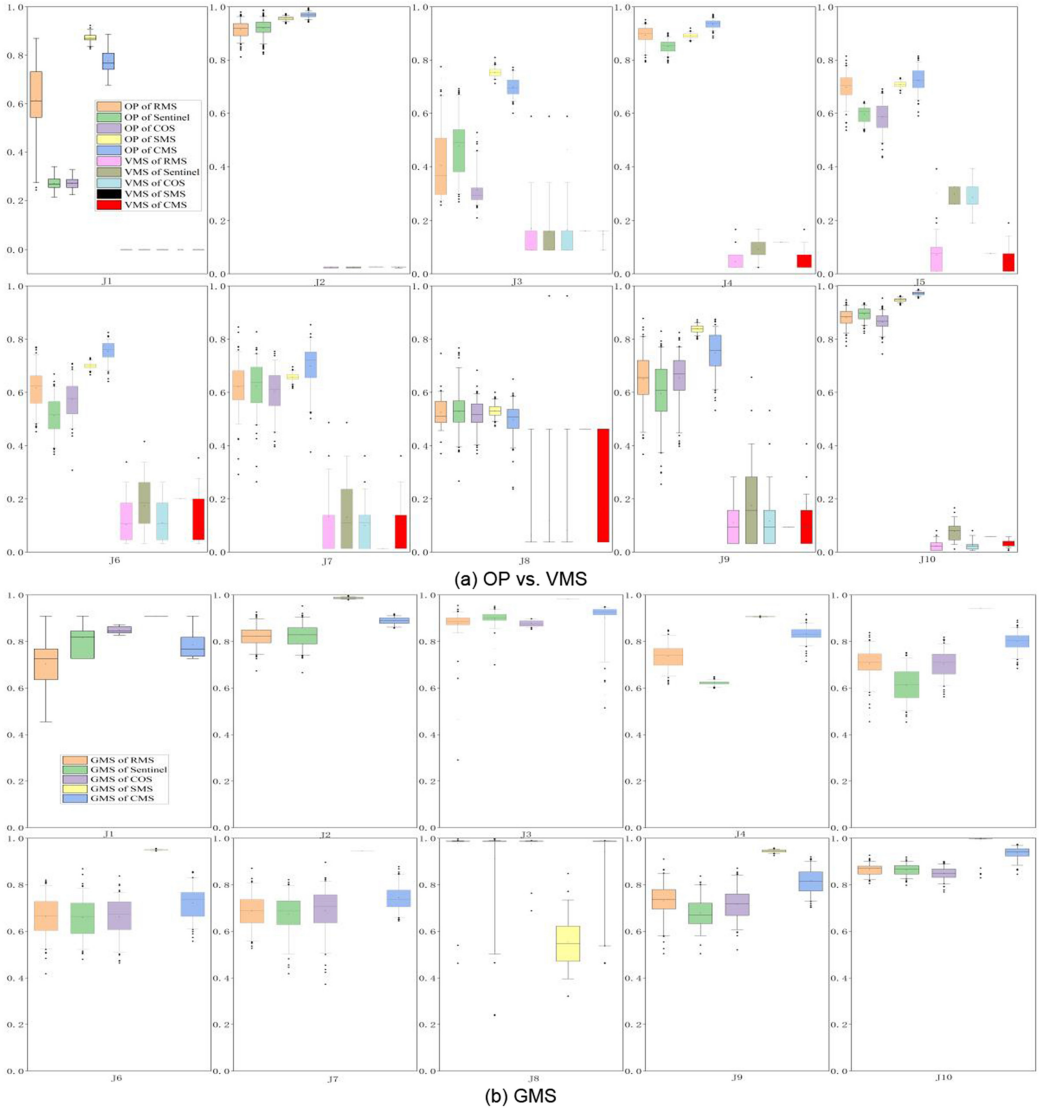


Fig. 4. The discrimination ability comparison between (a) OP, *VMS* and (b) GMS under a sufficient test suite for 10 programs. In (a), the legends from top to down are OP of RMS, Sentinel, COS, SMS, and CMS, *VMS* of RMS, Sentinel, COS, SMS, and CMS, respectively. Note that COS is skipped in J2 and J4 since there are fewer than 21 (39-18 and 109-88) mutants after filtering two mutators which means COS cannot be set to the same reduction ratio as SMS. In (b), the legends from top to down are GMS of RMS, Sentinel, COS, SMS, and CMS, respectively.

means "small", $0.5 \leq |d| < 0.8$ means "medium", and $|d| \geq 0.8$ means "large". Figures 4 and 5 show, for each of five strategies, the boxplot over 100 times mutation reductions.

From Table 5 and Figure 4, we have the following observations for ten programs:

- From the result on J1, we can find that, if the mutation score of a program is 100%, any selected mutant set will achieve a 100% mutation score. In other words, using *VMS* as a

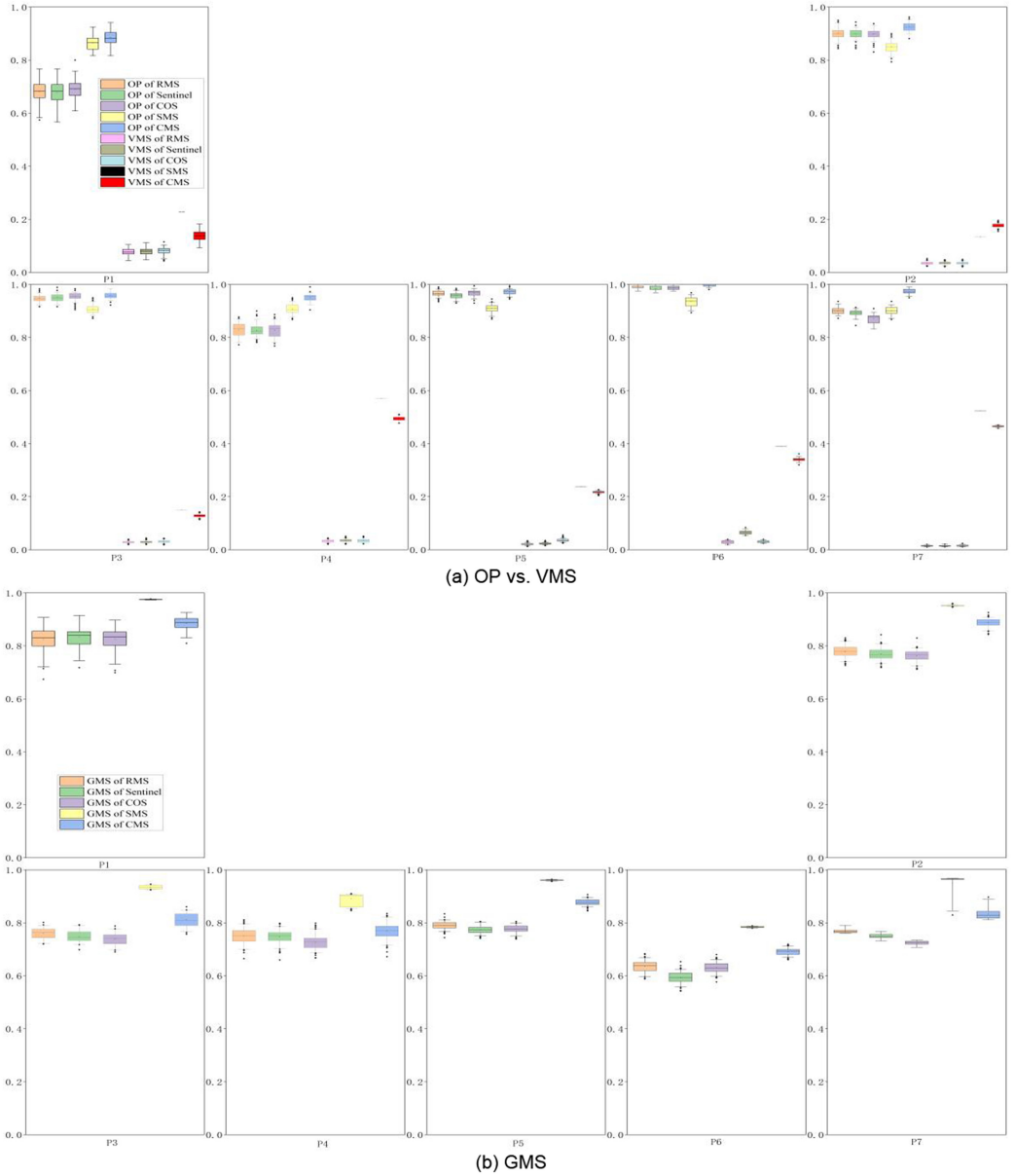


Fig. 5. The discrimination ability comparison between (a) OP, *VMS* and (b) GMS under a sufficient test suite for 7 projects. In (a), the legends from top to down are OP of RMS, Sentinel, COS, SMS, and CMS, *VMS* of RMS, Sentinel, COS, SMS, and CMS, respectively. In (b), the legends from top to down are GMS of RMS, Sentinel, COS, SMS, and CMS, respectively.

reduction effectiveness evaluation indicator is useless and misleading which makes all the evaluations the same. However, OP can clearly distinguish several strategies.

- By OP, SMS and CMS are more outstanding than RMS. In Table 5, on average, SMS and CMS have a 15% higher OP than RMS, while Sentinel and COS have lower OP than RMS. In Figure 4, the boxes of SMS and CMS are obviously higher than the other mutation reduction

strategies on most programs. On the whole, it is easy to distinguish several strategies. We speculate the reason is its objective function is not on OP. Thus, it may be difficult to achieve good performance under OP. See Section 6.4 for more possible reasons.

- By *VMS*, Sentinel, COS, and SMS are the "bad" strategies, while RMS is the "best" strategy. In Table 5, most of the average variations fluctuate around 0.1. However, the gap between programs is large. For J5, Sentinel and COS achieve two times worse *VMS* than their average *VMS*, while SMS and CMS achieve two times worse *VMS* for J8. In Figure 4, since SMS always selects the same mutants without randomness, the box for SMS is a line. For the other strategies, there is little difference on J2, J3 J4, J9, and J10. On the whole, it is not easy to distinguish several strategies since most are similar except Sentinel that looks a little worse. Regarding RMS, it can maintain the mutation score in mathematical expectation (see the proof in the appendix for detail), which can explain the best performance under *VMS*. This observation stands for the poor discrimination ability of *VMS*.
- By GMS, SMS and CMS are more outstanding than the other three strategies which is consistent to OP. However, we can find that the conclusions obtained by GMS and OP are different in many respects. For instance, on average, COS is worse than RMS and Sentinel by OP, while COS is better than RMS and Sentinel by GMS on 10 programs. This finding indicates that GMS cannot replace OP as an indicator of order preservation. At the same time, as we pointed out, GMS is not the proxy of the effectiveness of the reduced mutant set. As a result, the meaning of GMS needs further study. Even if the overall conclusion can be similar, the use of GMS may be riskier than OP, since we cannot assess the extent to which our conclusions based on GMS are correct. In other words, it is impossible to measure the frequency of pitfalls as mentioned in Section 3.2.2. Meanwhile, we cannot measure the error of the approximation algorithm.
- By the rank, on the one hand, the reduction strategies can be clustered into more groups under OP compared with GMS and VSM on most programs. This observation reveals the fact that OP has a stronger discrimination ability than GMS and VMS. On the other hand, the ranking result of OP is more consistent than GMS and VMS across programs: for OP, SMS and CMS can achieve the lower rank on most programs than the other strategies; for VMS, a strategy can achieve the highest (worst) rank on some programs while achieving the lowest (best) rank on some other programs. To sum the two observations, not only the discrimination ability of OP is stronger than GMS and VMS, but also the confidence of OP's conclusions is higher than GMS and VMS.

From Table 5 and Figure 5, we have the following observations for seven projects:

- By OP, CMS is the most outstanding mutation reduction strategy. In Table 5, on average, CMS achieves the OP value more than 0.95. In Figure 5, the boxes of CMS are obviously higher than the other mutation reduction strategies on most projects. On the whole, it is not easy to distinguish RMS, Sentinel, and COS. This may be due to the use of "ALL" mutation operators. A large number of mutants after mutator deletion led to that we mainly depend on random selection until the given reduction rate is achieved. Therefore, when COS is used to screen all mutation operators, it has a certain utility. However, its utility is not significantly greater than that of RMS. When using the default operator of PIT, the effect of COS screening mutation operator is lower than that of RMS. The performance of SMS is equable. SMS achieves OP around 0.9 on seven projects. Compared with the OP values on the 10 programs, all strategies achieve a better OP on the seven projects. This phenomenon was also observed in previous studies [21]. This means that comparing multiple test suites for a project is much less difficult than comparing multiple test suites for a program. On the

whole, the conclusion is consistent with that for programs: SMS and CMS are better than other strategies under OP on average.
- By *VMS*, SMS and CMS are the "bad" strategies, while RMS, Sentinel, and COS are the "good" strategies. If the developers need an absolute value of mutation score on a project against a test suite, RMS can provide a simple way for the estimate.
- By GMS, we can also find that the values on projects and programs are close. This is largely due to the elimination of the surviving mutants in advance. As a result, the mutation reduction is always performed on a set of mutants with 100% local mutation score. The consistent results show that, for a test suite that can kill all mutants selected by SMS, the mutation score on the original mutant set is usually high. However, this does not mean that the reduced mutant set contains the most unique faults, nor does it guarantee that the SMS can provide a high OP.

The conclusion obtained by OP is inconsistent with the conclusion obtained by *VMS*. Combining the above observations with the previous theoretical analysis, we can conclude that the use of OP is more objective and accurate, and the use of *VMS* may lead to misleading or confusing conclusions. For GMS, the conclusion is not credible before avoiding the pitfalls and estimating the error caused by the approximation algorithm.

To conclude, under sufficient test suites, OP is a better indicator than GMS and *VMS* in mutation reduction evaluation. CMS and SMS are more effective than RMS, Sentinel, and COS under OP on average.

5.2 RQ2: Discrimination Ability under an Insufficient Test Suite

First, we investigate the sub question RQ 2.1: When researchers use an insufficient test suite to evaluate the reduction strategies, whether the conclusion is consistent with RQ1. To this end, Table 6 reports for five strategies the average OP and *VMS* values over 100 times reduction results on each subject. Figures 6 and 7 show for each of the five strategies the boxplot over 100 times mutation reductions. From Table 6, Figures 6, and 7, we have the following observations:

- By OP, SMS and CMS are more outstanding than the other strategies. As shown in Table 6, on average, SMS has a 25% and 5% higher OP than RMS, CMS has a 20% and 10% higher OP than RMS for the programs and projects, respectively. Sentinel and COS have a lower or similar OP than RMS on these programs or projects, respectively. As shown in Figure 6, the boxes of SMS and CMS are obviously higher than the others on most programs. In Figure 7, the boxes of CMS are obviously higher. On the whole, it is easy to distinguish several strategies under OP. Overall, this conclusion is consistent with RQ1. Compared with the OP values in RQ1, the OP values in RQ2 are lower. It is worth noting that this is not due to the insufficient test suites. It is more likely due to the reduction ratio: selecting fewer mutants leads to the lower OP. We will analyze this in RQ3. As a result, we only care about whether the overall conclusion is consistent with RQ1, not about the quantitative change.
- By *VMS*, SMS is the "worst" strategy while RMS is still the "best" strategy. SMS achieves the worst *VMS* because of the highest *VMS* on J6, J8, and most projects. As shown in Figure 6, there is little difference for most of the five strategies on J1, J2, J3, J7, J8, and J9. In Figure 7, RMS, COS and Sentinel are similar to each other in most projects. On the whole, it is not easy to distinguish several strategies since RMS, COS, and Sentinel are similar in terms of VMS. Compared with the conclusion on *VMS* in RQ1, the difference is that the "worst" strategy on

Table 6. The Discrimination Ability Comparison between OP and VMS under an Insufficient Test Suite (the Strategy Rank by the Scott-Knott ESD Test is Shown in Brackets and a Lower Rank is Better)

	OP					VMS					Reduction
Program/ Project	RMS	Sentinel	COS	SMS	CMS	RMS	Sentinel	COS	SMS	CMS	Ratio
J1	0.725(3)	0.720(3)	-	1.000(1)	0.875(2)	0.097(2)	0.107(2)	-	0.018(1)	0.120(3)	6/11
J2	0.764(2)	0.753(2)	-	0.881(1)	0.882(1)	0.084(2)	0.077(1)	-	0.085(2)	0.102(3)	27/39
J3	0.433(3)	0.453(3)	0.393(4)	0.820(1)	0.599(2)	0.200(2)	0.245(3)	0.180(1)	0.210(2)	0.216(2)	59/62
J4	0.813(2)	0.791(3)	-	0.814(2)	0.895(1)	0.075(2)	0.098(3)	-	0.058(1)	0.079(2)	95/109
J5	0.688(3)	0.534(4)	0.536(4)	0.724(2)	0.780(1)	0.091(1)	0.302(3)	0.306(3)	0.118(2)	0.099(1)	100/115
J6	0.499(3)	0.410(5)	0.479(4)	0.737(1)	0.651(2)	0.126(1)	0.170(2)	0.127(1)	0.409(3)	0.141(1)	203/213
J7	0.535(3)	0.547(3)	0.539(3)	0.756(1)	0.698(2)	0.152(2)	0.129(1)	0.133(1)	0.181(3)	0.181(3)	142/149
J8	0.646(2)	0.641(2)	0.666(1)	0.631(3)	0.608(4)	0.087(1)	0.162(3)	0.116(2)	0.453(5)	0.214(4)	213/215
J9	0.643(3)	0.595(4)	0.636(3)	0.862(1)	0.760(2)	0.119(1)	0.130(2)	0.114(1)	0.239(3)	0.107(1)	126/134
J10	0.779(4)	0.793(3)	0.754(5)	0.931(2)	0.962(1)	0.042(1)	0.095(3)	0.047(1)	0.125(4)	0.074(2)	1128/1161
avg.	0.653	0.624	0.572	0.816	0.771	0.107	0.145	0.146	0.182	0.129	-
P1	0.594(3)	0.571(4)	0.592(3)	0.819(1)	0.837(2)	0.090(1)	0.091(1)	0.090(1)	0.178(3)	0.155(2)	7036/7056
P2	0.823(3)	0.835(2)	0.822(3)	0.838(2)	0.881(1)	0.043(1)	0.045(1)	0.045(1)	0.220(3)	0.156(2)	10393/10467
P3	0.920(2)	0.922(2)	0.925(2)	0.893(3)	0.959(1)	0.037(1)	0.038(1)	0.042(1)	0.212(2)	0.231(3)	20084/20204
P4	0.787(3)	0.772(3)	0.800(2)	0.938(1)	0.942(1)	0.028(1)	0.038(1)	0.038(1)	0.536(3)	0.479(2)	56823/56898
P5	0.944(2)	0.940(2)	0.952(2)	0.887(3)	0.972(1)	0.025(1)	0.025(1)	0.040(1)	0.292(3)	0.255(2)	24168/24407
P6	0.971(2)	0.952(3)	0.971(2)	0.943(3)	0.992(1)	0.037(1)	0.067(2)	0.036(1)	0.421(4)	0.339(3)	102607/102718
P7	0.844(3)	0.845(3)	0.841(3)	0.885(2)	0.960(1)	0.023(1)	0.024(1)	0.024(1)	0.366(2)	0.395(3)	63224/63593
avg.	0.840	0.833	0.843	0.886	0.935	0.040	0.047	0.045	0.318	0.287	-

these programs changes from Sentinel to SMS, and the other strategies still seem difficult to distinguish.

Second, we investigate the sub question RQ 2.2: When researchers use an insufficient test suite to implement the strategies (i.e., SMS and CMS in this paper), whether the conclusion on OP is consistent with RQ1. To this end, Table 7 reports for SMS and CMS the average OP values over 100 times reduction results on each subject and compares them with those in RQ1. Note that the computation of OP in Table 7 is based on the sufficient test suites. From Table 7, we have the following observations:

- When implemented using a relatively insufficient test suite, SMS selects fewer mutants on most programs and projects, which is indicated by the reduction ratio. This means that, based on a small number of test cases, it is difficult to distinguish mutants by coverage information. There are more mutants that cannot be distinguished because they are covered by the same set of test cases.
- When implemented using a relatively insufficient test suite, SMS and CMS usually achieve lower OP values than those in RQ1. However, the average OP values drop less than 3%. An obvious fact is that SMS relies on coverage information to simulate subsuming relationship. As a result, fewer test cases lead to a more inaccurate approximation.

Combining the results in RQ 2.1 and RQ 2.2, we find that the main conclusions in RQ2 are consistent with those in RQ1: (1) OP has a stronger discrimination ability compared with *VMS*; (2) SMS and CMS are more effective than RMS, COS, and Sentinel under OP; (3) it is still hard to

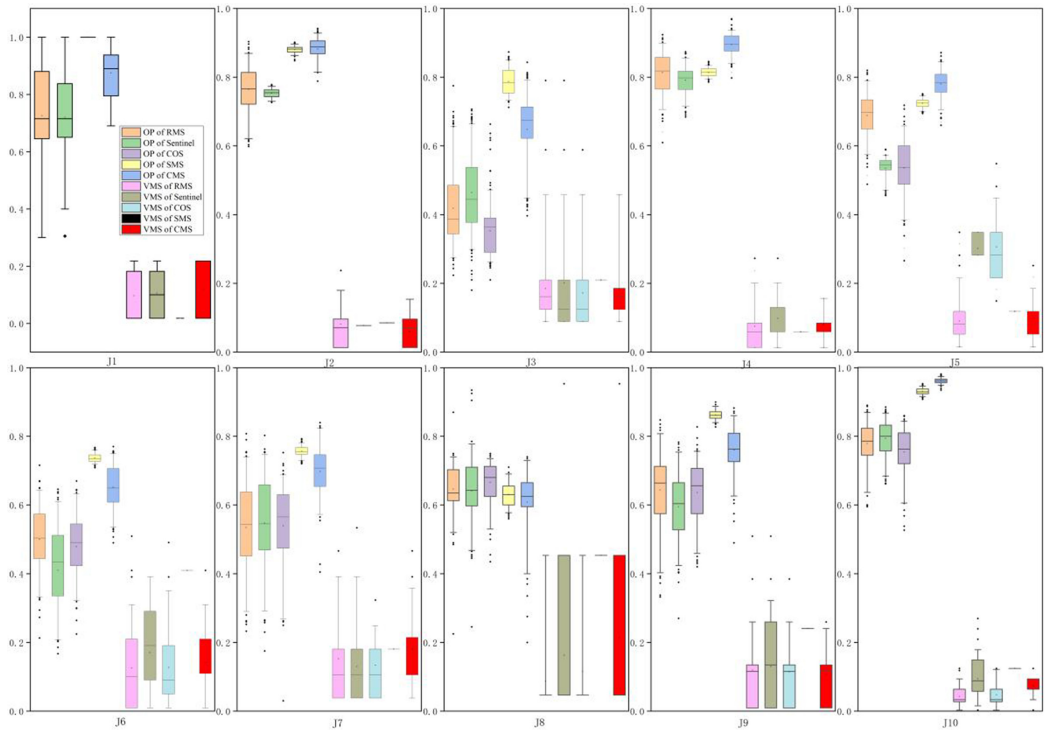


Fig. 6. The discrimination ability comparison between OP and *VMS* under an insufficient test suite. The legends from top to down are OP of RMS, Sentinel, COS, SMS, and CMS, *VMS* of RMS, Sentinel, COS, SMS, and CMS, respectively. It is worth noting that COS is skipped in J1, J2, and J4 since there are respectively fewer than 5, 12, and 14 mutants after filtering two mutators, which means COS cannot be set to the same reduction ratio as SMS.

distinguish five strategies by *VMS*; and (4) when implemented using a relatively insufficient test suite, SMS and CMS achieve the OP values with a trivial drop.

To conclude, under insufficient test suites, OP is still a better indicator than *VMS* in mutation reduction evaluation. In particular, the conclusions drawn from OP are stable when test suites change from sufficient to insufficient.

5.3 RQ3: Influence of Reduction Ratio on Reduction Effectiveness

Figures 8 and 9 show the OP-reduction ratio scatter diagrams on 10 programs and 7 projects, respectively. From Figures 8 and 9, we have the following observations:

- For all the mutation reduction strategies, on the whole, a higher reduction ratio leads to a greater loss in OP (i.e., the order preservation ability). Furthermore, for the four feasible strategies, setting the reduction ratio to 0.4 (i.e., select 60% mutants among all mutants) can achieve an OP value larger than 0.9 on most programs and projects. Considering that CMS needs coverage relationship matrix for implementation, when practitioners focus on maintaining the relative evaluation of test suites, selecting 60% mutants randomly is an easy choice than using CMS. Furthermore, we can see that existing reduction strategies may have an unsatisfactory OP (i.e., <0.9) under a high reduction ratio (i.e., >0.8). Therefore, in the

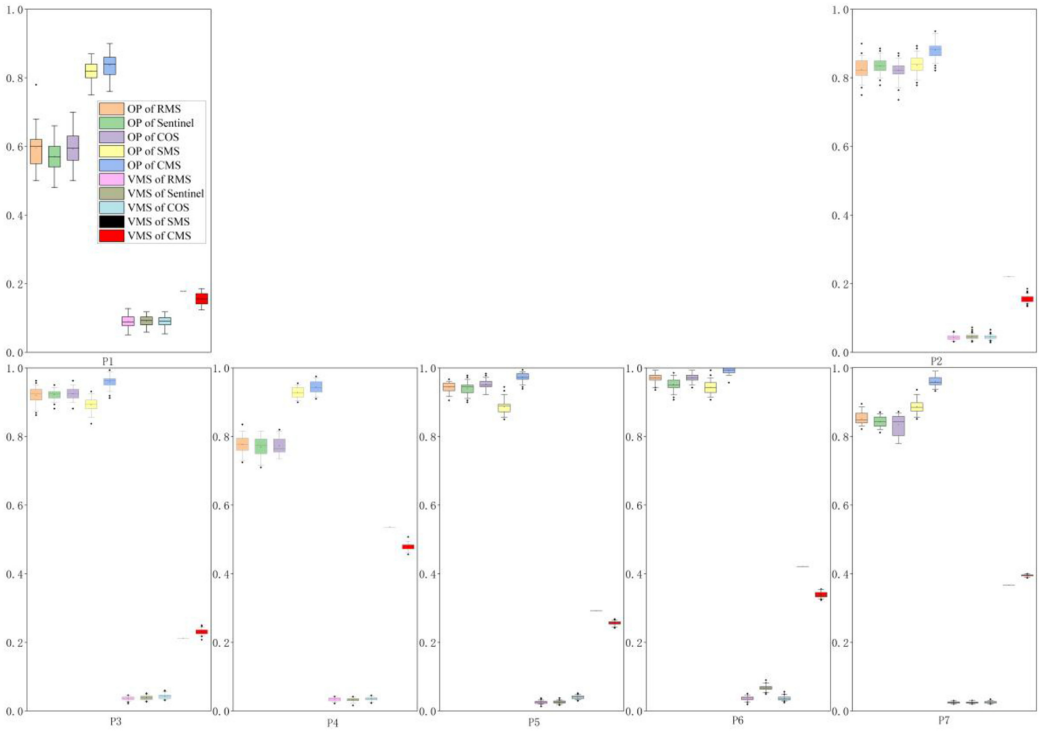


Fig. 7. The discrimination ability comparison between OP and *VMS* under an insufficient test suite for 7 projects. The legends from top to down are OP of RMS, Sentinel, COS, SMS, and CMS, *VMS* of RMS, Sentinel, COS, SMS, and CMS, respectively.

future, a natural research problem is how to keep a high reduction ratio and a high OP at the same time for a program?

- For the programs, CMS shows the best reduction effectiveness, which can maintain a relatively high OP and a relatively high reduction ratio at the same time. COS looks worse than RMS, while Sentinel looks close to RMS on most programs. We speculate that PIT has retained most of the valuable mutation operators in the process of screening mutation operators, and hence it is of little meaning to select operators on this basis. For Sentinel, the essence of this approach is to bundle a lot of random strategies. As a result, Sentinel has the similar effectiveness compared with RMS.
- For the projects, the gap between strategies has become smaller. We can still observe that CMS is above other strategies on the whole. However, an important fact is that any feasible strategy with more than 10% selection will achieve an OP value more than 0.9. This means that it is more practical to use simple strategies on projects.

Figures 10 and 11 show the EROP-reduction ratio scatter diagrams on 10 programs and 7 projects, respectively. From Figures 10 and 11, we have the following observations:

- For programs, on the whole, setting a reduction ratio between 60% and 80% is both effective and efficient for CMS. A reduction ratio less than 60% has little effect for all strategies, while a reduction ratio more than 80% has unstable effects among different programs. For COS and Sentinel, it is hard to recommend a reduction ratio for all programs.

Table 7. The Comparison of SMS and CMS Implemented by Different Test Suites for 10 Programs and 7 Projects

Program/ Project	implemented by the insufficient test suites			implemented by the sufficient test suites		
	SMS	CMS	Reduction Ratio	SMS	CMS	Reduction Ratio
J1	0.962	0.885	6/11	0.872	0.777	7/11
J2	0.879	0.880	27/39	0.956	0.970	18/39
J3	0.661	0.572	59/62	0.754	0.697	58/62
J4	0.806	0.871	95/109	0.891	0.936	88/109
J5	0.744	0.775	100/115	0.708	0.724	96/115
J6	0.693	0.687	203/213	0.700	0.754	200/213
J7	0.687	0.650	142/149	0.658	0.699	141/149
J8	0.492	0.509	213/215	0.531	0.496	213/215
J9	0.894	0.804	126/134	0.839	0.749	126/134
J10	0.868	0.935	1128/1161	0.947	0.972	1103/1161
avg.	0.769	0.757	-	0.787	0.778	-
P1	0.824	0.833	7036/7056	0.864	0.885	7030/7056
P2	0.849	0.878	10393/10467	0.850	0.926	10347/10467
P3	0.889	0.961	20084/20204	0.907	0.959	20035/20204
P4	0.919	0.939	56823/56898	0.908	0.950	56785/56898
P5	0.878	0.973	24168/24407	0.909	0.972	24063/24407
P6	0.943	0.993	102607/102718	0.937	0.995	102552/102718
P7	0.888	0.966	63224/63593	0.900	0.973	62928/63593
avg.	0.884	0.935	-	0.896	0.951	-

- For projects, on the whole, setting a reduction ratio around 90% is both effective and efficient for CMS. Other strategies have no obvious advantages compared with RMS.
- CMS shows the best performance, which is above other strategies on the whole. Sentinel has a performance close to RMS, while COS is the most ineffective strategy. Considering the unstable performance of COS and Sentinel on different projects or programs, we suggest using RMS if it is hard to collect the coverage information for CMS.

Combining the above results, we can say that CMS is a more effective strategy.

To conclude, for each reduction strategy, the reduction effectiveness as measured by OP decreases with the reduction ratio. For the best performing reduction strategy CMS, setting a reduction ratio with in [60%, 80%] and [90%, 95%] is both effective and efficient for the programs and projects, respectively. By EROP, it is difficult for existing strategies to significantly exceed RMS when the reduction ratio is less than 50%. When the reduction ratio is larger than 50%, only SMS and CMS can achieve a better performance than RMS on most programs.

6 DISCUSSION

In this section, we first discuss why not use the average variation in mutation scores on multiple test suites as the indicator to evaluate mutation reduction effectiveness. Then, we examine whether the subset constraint can be removed when generating a number of test suites to compute OP. After that, we diagnose whether the way to generate insufficient test suites affects the conclusions of RQ2. Later, we analyze the reason why Sentinel has a low OP value. Finally, we compare the efficiency of OP with VMS and GMS.

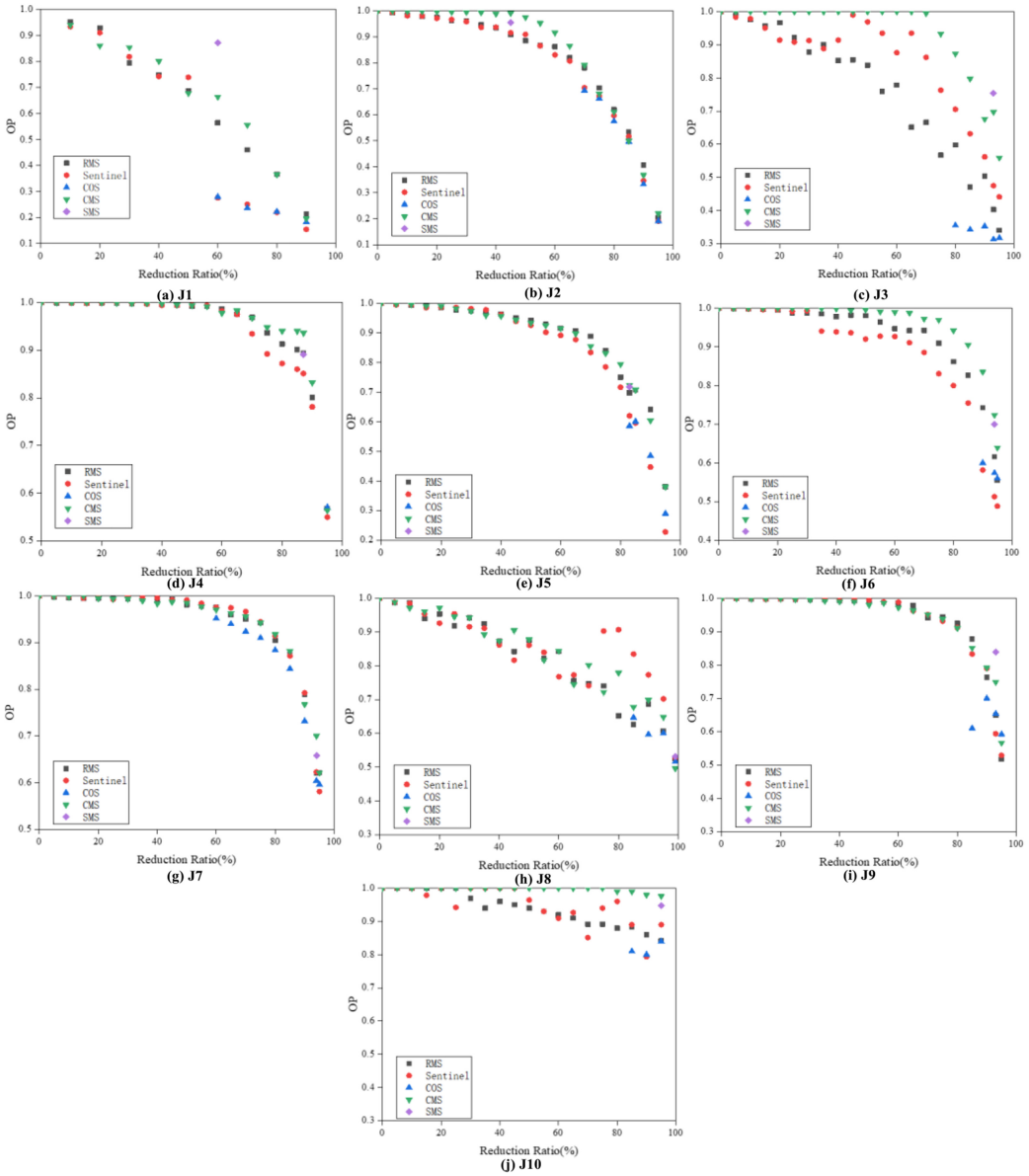


Fig. 8. The OP-reduction ratio scatter diagram on 10 programs.

6.1 Why not use the Average Variation in Mutation Scores on Multiple Test Suites as the Indicator?

In the current literature, it is a standard practice to use *VMS* from a single test suite to evaluate the effectiveness of a reduction strategy, i.e., the ability to maintain mutation score. However, as analyzed in Section 3, in order to obtain a comprehensive evaluation, we should assess the ability of a mutation reduction strategy to maintain mutation score on ALL test suites. In practice, it is not feasible to generate all possible test suites for mutation reduction evaluation. As a compromise, we can use multiple test suites to conduct the evaluation. In this context, a natural problem is: What if the average *VMS* on multiple test suites is used as the evaluation indicator? To figure out

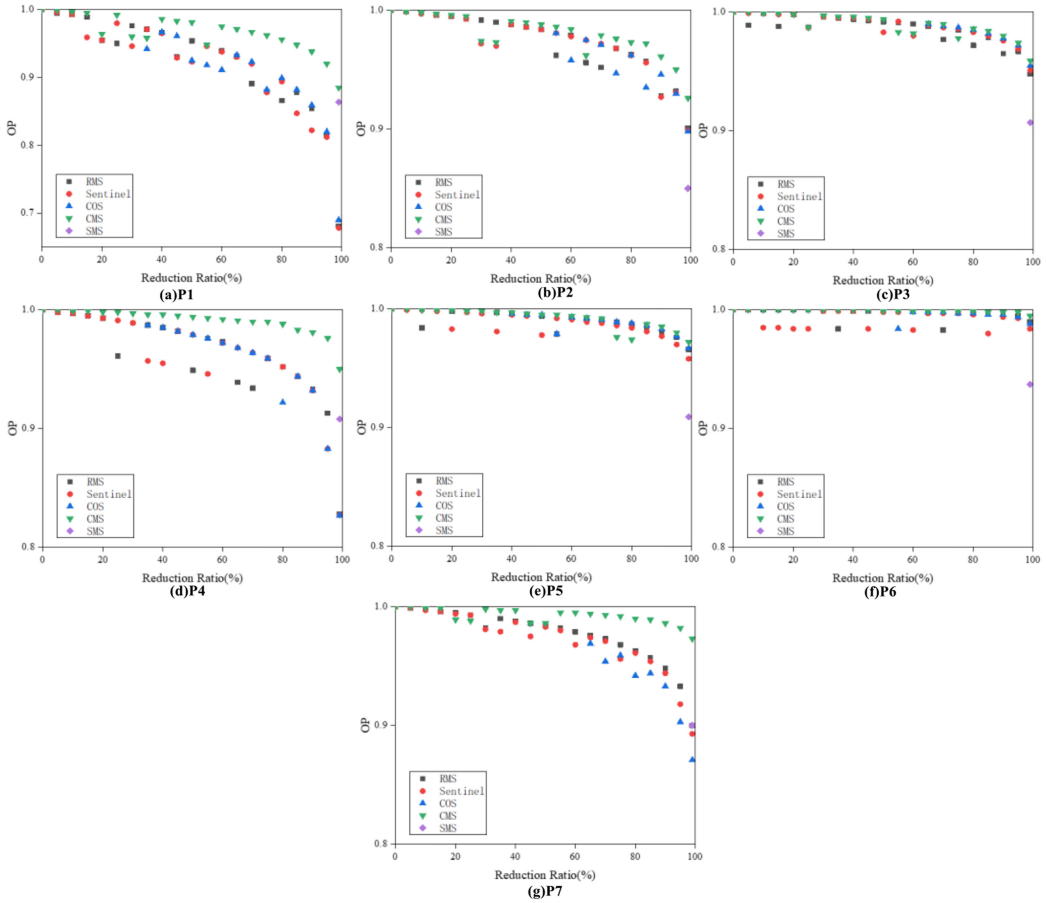


Fig. 9. The OP-reduction ratio scatter diagram on 7 projects.

this question, we compute for each program the average *VMS* (as shown in Table 8) on the same test suites that are used to compute OP. Note that the settings here are consistent with RQ1. From Table 8, we have the following observations:

- Compared with Table 5, the average *VMS* on multiple test suites is larger than *VMS* on the original test suite for most strategies on most programs or projects. Besides, the rank groups are more than that in Table 5, which means the discrimination ability of average *VMS* is stronger than *VMS*.
- RMS achieves the "best" performance according to average *VMS*. There is little difference among COS, Sentinel, and RMS on average. The large variations of SMS on J3, J4, J8, J10, and most projects lead to the poor average result. The most of other cells for the five strategies are less than 0.25.
- Compared with OP, it is not easy to distinguish these strategies by average *VMS*. Although average *VMS* produces more ranks than *VMS*, Sentinel, COS, SMS, and CMS can achieve the "1" or "2" rank on some programs while achieving the "4" or "5" rank on the other programs. As a result, it is hard to use average *VMS* to distinguish these reduction strategies based on the different ranks among 10 programs. For the projects, although the distinguish ability of VMS is comparable to that of OP, the conclusion is contrary to OP's conclusion.

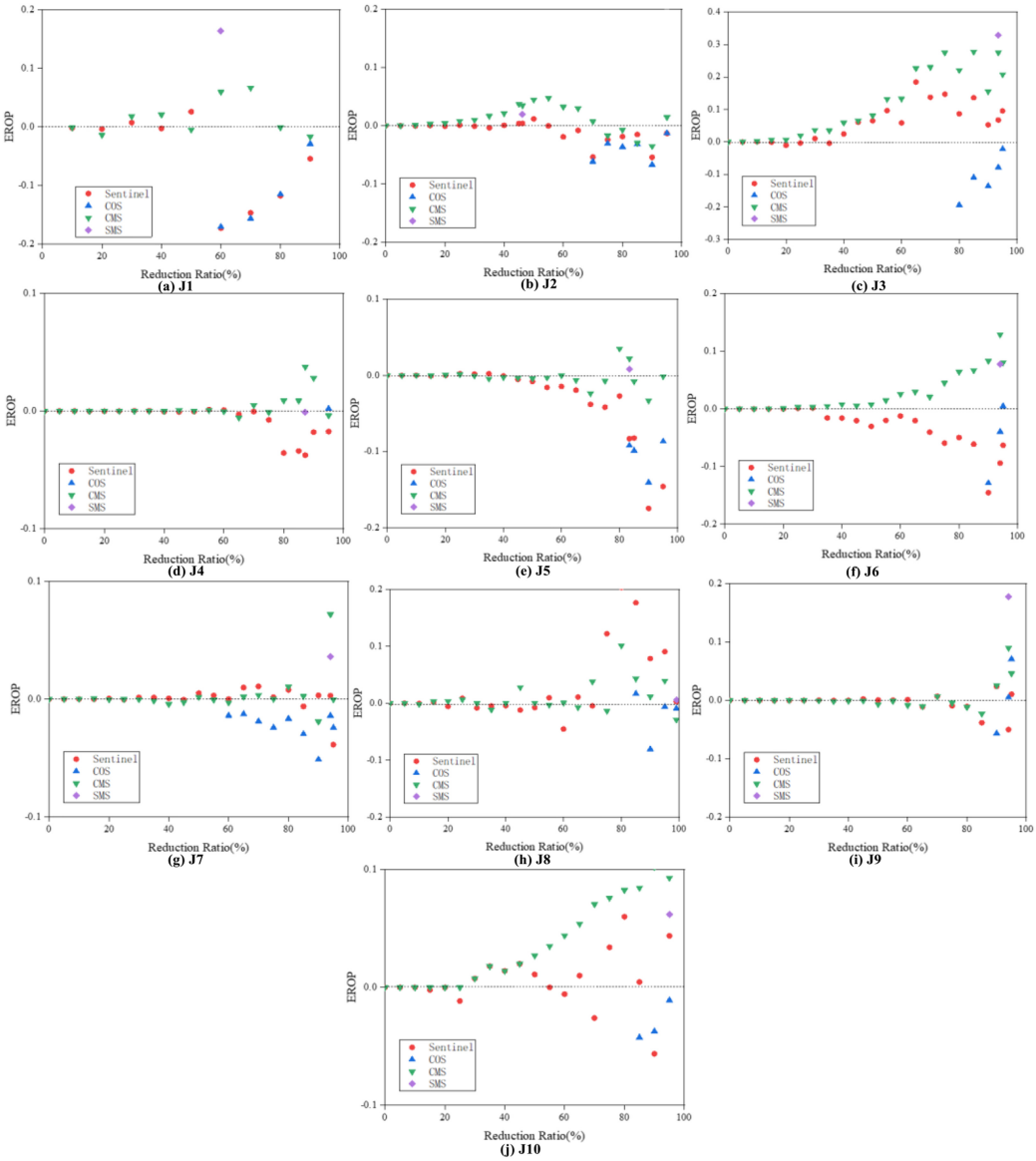


Fig. 10. The EROP-reduction ratio scatter diagram on 10 programs.

Based on the above observations, we can see that the average *VMS* still has an unsatisfactory ability to distinguish mutation reduction strategies. Furthermore, normally, the average *VMS* still measures the ability to maintain mutation score rather than the "order-preserving ability". As a result, we do not recommend using the average *VMS* as the indicator for the evaluation of the effectiveness of a reduction strategy.

6.2 Can the Subset Constraint be Removed when Generating Test Suites to Compute OP?

In Section 3.3, given a SUT with the original test suite T, in order to compute OP, we employ a "continuous half-sample" approach to generate a sequence of test suites from T such that they have

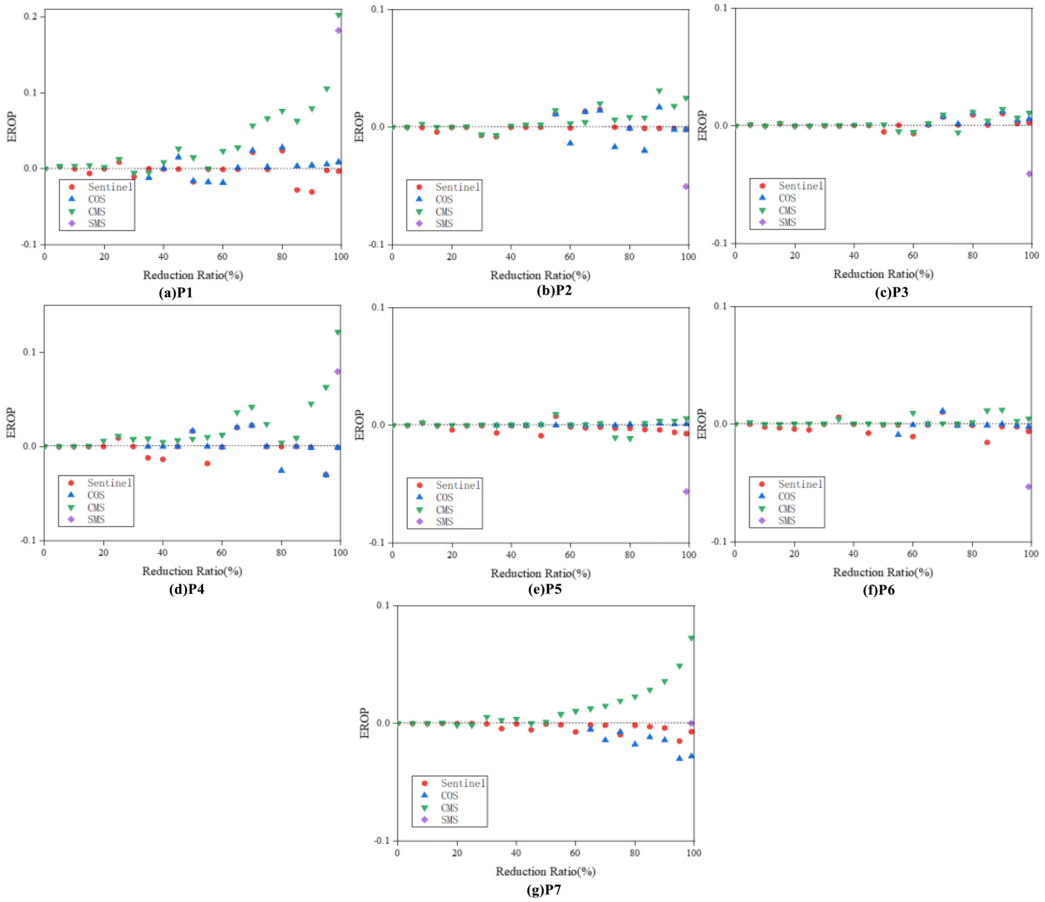


Fig. 11. The EROP-reduction ratio scatter diagram on 7 projects.

a subset constraint. Specifically, the subset constraint enables to obtain a sequence of test suites by generating "T_{i+1} by randomly selecting half of the test cases (round down) in T_i". Normally, this subset constraint places a restriction on which pairs of test suites the mutation score order can be used to compute OP. In other words, since T_{i+1} is a subset of T_i, we know that T_{i+1} must have a weaker or at most the same ability in detecting defects compared with T_i. In this context, the subset constraint ensures that the mutation score order on (T_i, T_{i+1}) before mutation reduction can be regarded as the ground truth in defect detecting ability. As a result, OP can reflect to what extent a reduction strategy maintains the ability to measure the effectiveness of test suites in detecting defects.

However, for most of the existing studies, the mutation score is regarded as the ground truth among all the test suites. In other words, given two test suites, regardless of whether two suites have a subset relation, the test suite with a higher mutation score is considered more effective in detecting defects than the other test suite. Under this assumption, a natural problem is immediately raised: Will the conclusions of OP be changed? To investigate this problem, in this subsection, we remove the subset constraint to compare two suites based on such an assumption: the mutation score order here is indeed a set of ($T1$, $T2$, operator) satisfying operator is "$>$" if $T1$ has a higher mutation score and is "$=$" if $T1$ has the same mutation score compared with $T2$. By this definition,

Table 8. The Discrimination Ability Comparison between OP and Average VMS under a Sufficient Test Suite (the Strategy Rank by the Scott-Knott ESD Test in Brackets and a Lower Rank is Better)

	OP					Average VMS					Reduction
Program/ Project	RMS	Sentinel	COS	SMS	CMS	RMS	Sentinel	COS	SMS	CMS	Ratio
J1	0.615(3)	0.270(4)	0.270(4)	0.872(1)	0.777(2)	0.093(1)	0.140(3)	0.140(3)	0.125(2)	0.099(1)	7/11
J2	0.913(3)	0.920(3)	-	0.956(2)	0.970(1)	0.050(2)	0.048(2)	-	0.066(3)	0.040(1)	18/39
J3	0.403(4)	0.475(3)	0.313(5)	0.754(1)	0.697(2)	0.154(1)	0.197(3)	0.143(1)	0.374(4)	0.169(2)	58/62
J4	0.893(2)	0.851(3)	-	0.891(2)	0.936(1)	0.069(1)	0.088(2)	-	0.216(4)	0.097(3)	88/109
J5	0.698(3)	0.596(4)	0.586(5)	0.708(2)	0.724(1)	0.087(1)	0.185(4)	0.211(5)	0.160(3)	0.101(2)	96/115
J6	0.616(3)	0.513(5)	0.574(4)	0.700(2)	0.754(1)	0.082(2)	0.127(3)	0.087(2)	0.068(1)	0.119(3)	200/213
J7	0.620(3)	0.623(3)	0.604(4)	0.658(2)	0.699(1)	0.128(3)	0.115(2)	0.123(3)	0.104(1)	0.106(1)	141/149
J8	0.525(1)	0.528(1)	0.516(2)	0.531(1)	0.496(3)	0.107(1)	0.172(2)	0.146(3)	0.527(5)	0.247(4)	213/215
J9	0.649(3)	0.594(4)	0.654(3)	0.839(1)	0.749(2)	0.123(2)	0.176(3)	0.123(2)	0.168(3)	0.106(1)	126/134
J10	0.879(4)	0.892(3)	0.865(5)	0.947(2)	0.972(1)	0.045(1)	0.090(2)	0.050(1)	0.348(4)	0.283(3)	1103/1161
avg.	0.681	0.626	0.547	0.787	0.778	0.094	0.134	0.128	0.216	0.137	-
P1	0.681(3)	0.678(3)	0.690(3)	0.864(2)	0.885(1)	0.061(1)	0.063(1)	0.066(2)	0.198(4)	0.100(3)	7030/7056
P2	0.901(2)	0.899(2)	0.898(2)	0.850(3)	0.926(1)	0.057(1)	0.057(1)	0.057(1)	0.169(3)	0.116(2)	10347/10467
P3	0.948(2)	0.951(2)	0.955(1)	0.907(3)	0.959(1)	0.043(1)	0.040(1)	0.041(1)	0.189(3)	0.167(2)	20035/20204
P4	0.828(3)	0.827(3)	0.827(3)	0.908(2)	0.950(1)	0.025(1)	0.027(1)	0.027(1)	0.237(3)	0.206(2)	56785/56898
P5	0.966(2)	0.958(3)	0.967(2)	0.909(4)	0.972(1)	0.035(1)	0.035(1)	0.052(2)	0.266(4)	0.245(3)	24063/24407
P6	0.990(2)	0.984(2)	0.988(2)	0.937(3)	0.995(1)	0.048(1)	0.057(2)	0.039(1)	0.184(3)	0.211(4)	102552/102718
P7	0.900(2)	0.893(3)	0.871(4)	0.900(2)	0.973(1)	0.033(1)	0.034(1)	0.033(1)	0.301(3)	0.239(2)	62928/63593
avg.	0.888	0.884	0.885	0.896	0.951	0.043	0.045	0.045	0.221	0.183	-

we can judge the effectiveness relationship between any two test suites. To compare with the original OP, we first randomly select 100*k (k = int($\log_2|T_0|$), see Section 3.3 to find the detail of k) pairs of test suites among the subset space of the original test suite T_0 to compute a New OP (NOP). If the operator on ($T1$, $T2$, operator) is not changed before and after reduction, we refer to ($T1$, $T2$) to an order-preserving pair. Then, we divide the total number of order-preserving pairs by 100*k to obtain NOP.

Table 9 reports the OP and NOP values under five mutation reduction strategies. Note that the settings here are consistent with RQ1. From Table 9, we have the following observations:

- By NOP, the conclusion on five reduction strategies is consistent with RQ1: SMS and CMS are better than RMS, Sentinel, and COS.
- Overall, NOP is larger than OP. We do not have enough evidence to prove whether NOP overestimates the "order-preserving ability" or OP underestimates it. The cause for this phenomenon is not clear, which is a direction of future work.

However, we do not recommend using this approach to calculate the "order-preserving ability" of a reduction strategy, since the assumption may be questionable. Papadakis et al. concluded that "MS might not be a reliable indicator of the test effectiveness" [38]. Compared with NOP, the effectiveness relationship of the test suite sequence used by OP is a ground truth based on *score monotonicity property* (see the footnote in Section 3.3 for detail). As a result, we believe that the confidence of OP is stronger than NOP. To conclude, the conclusion of NOP is consistent with the previous conclusion. It is possible to use NOP in future studies, but we do not recommend it since NOP requires stronger assumptions than OP.

Table 9. The Discrimination Ability Comparison between OP and NOP under a Sufficient Test Suite (the Strategy Rank by the Scott-Knott ESD test is Shown in Brackets and a Lower Rank is Better)

	OP					NOP					Reduction
Program/ Project	RMS	Sentinel	COS	SMS	CMS	RMS	Sentinel	COS	SMS	CMS	Ratio
J1	0.615(3)	0.270(4)	0.270(4)	0.872(1)	0.777(2)	0.676(3)	0.435(4)	0.363(5)	0.860(1)	0.790(2)	7/11
J2	0.913(3)	0.920(3)	-	0.956(2)	0.970(1)	0.891(4)	0.905(3)	-	0.917(2)	0.941(1)	18/39
J3	0.403(4)	0.475(3)	0.313(5)	0.754(1)	0.697(2)	0.368(3)	0.286(4)	0.235(5)	0.880(1)	0.805(2)	58/62
J4	0.893(2)	0.851(3)	-	0.891(2)	0.936(1)	0.801(4)	0.843(3)	-	0.878(2)	0.914(1)	88/109
J5	0.698(3)	0.596(4)	0.586(5)	0.708(2)	0.724(1)	0.775(3)	0.755(4)	0.716(5)	0.880(2)	0.904(1)	96/115
J6	0.616(3)	0.513(5)	0.574(4)	0.700(2)	0.754(1)	0.778(3)	0.737(5)	0.759(4)	0.888(2)	0.899(1)	200/213
J7	0.620(3)	0.623(3)	0.604(4)	0.658(2)	0.699(1)	0.706(4)	0.731(3)	0.654(5)	0.820(2)	0.845(1)	141/149
J8	0.525(1)	0.528(1)	0.516(2)	0.531(1)	0.496(3)	0.583(3)	0.595(2)	0.560(4)	0.620(1)	0.593(2)	213/215
J9	0.649(3)	0.594(4)	0.654(3)	0.839(1)	0.749(2)	0.645(5)	0.663(4)	0.695(3)	0.843(1)	0.772(2)	126/134
J10	0.879(4)	0.892(3)	0.865(5)	0.947(2)	0.972(1)	0.883(3)	0.881(3)	0.836(4)	0.929(2)	0.948(1)	1103/1161
avg.	0.681	0.626	0.547	0.787	0.778	0.711	0.683	0.602	0.852	0.841	-
P1	0.681(3)	0.678(3)	0.690(3)	0.864(2)	0.885(1)	0.697(3)	0.705(3)	0.690(3)	0.888(1)	0.860(2)	7030/7056
P2	0.901(2)	0.899(2)	0.898(2)	0.850(3)	0.926(1)	0.920(2)	0.921(2)	0.917(2)	0.948(1)	0.942(1)	10347/10467
P3	0.948(2)	0.951(2)	0.955(1)	0.907(3)	0.959(1)	0.958(2)	0.957(2)	0.961(2)	0.955(2)	0.967(1)	20035/20204
P4	0.828(3)	0.827(3)	0.827(3)	0.908(2)	0.950(1)	0.844(3)	0.845(3)	0.844(3)	0.939(2)	0.958(1)	56785/56898
P5	0.966(2)	0.958(3)	0.967(2)	0.909(4)	0.972(1)	0.973(2)	0.960(2)	0.973(2)	0.945(3)	0.981(1)	24063/24407
P6	0.990(2)	0.984(2)	0.988(2)	0.937(3)	0.995(1)	0.981(2)	0.980(2)	0.982(2)	0.944(3)	0.992(1)	102552/102718
P7	0.900(2)	0.893(3)	0.871(4)	0.900(2)	0.973(1)	0.917(3)	0.906(4)	0.899(4)	0.936(2)	0.979(1)	62928/63593
avg.	0.888	0.884	0.885	0.896	0.951	0.898	0.903	0.895	0.936	0.954	-

6.3 Does the Approach to Generating Insufficient Test Suites Affect the Conclusions of RQ2?

In RQ2, we investigate whether the conclusion in RQ1 would change under insufficient test suites. In the literature, many studies simulate insufficient test sets by changing the size of test sets [24, 35–37]. In RQ2, we take a similar idea to generate insufficient test suites. Specifically, we "delete test cases in RQ1 with even index for each program" to simulate an insufficient test suite, which enables readers to reproduce our experiment without randomness. However, for a SUT with the original test suite T, such a generation approach leads to only one specific insufficient test suite. It is unknown whether and to what extent it will affect the conclusions of RQ2. To tackle this problem, in this section, we randomly generate 100 subsets from T to simulate multiple insufficient test suites, each of which contains half of the test cases in T. Table 10 shows the average OP and *VMS* of 100 subsets. From Table 10, we can find that the conclusions are consistent with RQ2: (1) OP is a better indicator than *VMS* in mutation reduction evaluation; and (2) SMS and CMS are better than RMS, Sentinel, and COS. As a result, we can conclude that the approach to generating insufficient test suites has little effect on the conclusions of RQ2.

6.4 What is the Possible Reason that Sentinel has a low OP Value?

From the experimental results, we can find that Sentinel has a low OP value. Since Sentinel is a state-of-the-art work on mutant reduction, we want to figure out the possible reasons why it has an unsatisfactory OP value. In our opinion, the reasons for this are mainly two-fold. First, Sentinel is originally proposed for cross-version prediction (rather than for cross-project prediction), i.e., it is trained on the old version of a project and is tested on the new version of the same project [8]. In

Table 10. The Discrimination Ability Comparison between OP and VMS under Multiple Insufficient Test Suites (the Strategy Rank by the Scott-Knott ESD Test Is Shown in Brackets and a Lower Rank is Better)

	OP					VMS				
Program/ Project	RMS	Sentinel	COS	SMS	CMS	RMS	Sentinel	COS	SMS	CMS
J1	0.642(3)	0.410(4)	-	0.895(1)	0.768(2)	0.145(3)	0.155(3)	-	0.091(1)	0.115(2)
J2	0.776(3)	0.687(4)	-	0.903(1)	0.810(2)	0.076(1)	0.121(2)	-	0.133(3)	0.068(1)
J3	0.404(3)	0.407(3)	0.392(3)	0.825(1)	0.587(2)	0.199(4)	0.176(3)	0.178(3)	0.161(2)	0.148(1)
J4	0.825(3)	0.678(4)	-	0.864(2)	0.898(1)	0.076(2)	0.062(1)	-	0.058(1)	0.074(2)
J5	0.615(3)	0.507(4)	0.476(5)	0.665(1)	0.636(2)	0.102(1)	0.193(4)	0.298(5)	0.160(3)	0.133(2)
J6	0.544(3)	0.483(5)	0.509(4)	0.727(1)	0.715(2)	0.119(2)	0.184(3)	0.080(1)	0.300(4)	0.189(3)
J7	0.575(3)	0.655(2)	0.556(4)	0.709(1)	0.663(2)	0.180(3)	0.213(4)	0.136(2)	0.174(3)	0.084(1)
J8	0.536(3)	0.515(4)	0.539(3)	0.695(1)	0.670(2)	0.192(1)	0.204(1)	0.191(1)	0.375(3)	0.306(2)
J9	0.586(3)	0.572(3)	0.554(4)	0.768(1)	0.750(2)	0.135(2)	0.106(1)	0.142(2)	0.284(3)	0.127(2)
J10	0.735(4)	0.755(3)	0.730(4)	0.893(1)	0.798(2)	0.057(1)	0.078(2)	0.053(1)	0.182(3)	0.081(3)
avg.	0.624	0.567	0.537	0.794	0.730	0.128	0.149	0.154	0.192	0.133
P1	0.578(3)	0.566(4)	0.577(3)	0.845(1)	0.812(2)	0.075(1)	0.075(1)	0.074(1)	0.366(3)	0.258(2)
P2	0.845(2)	0.846(2)	0.845(2)	0.813(3)	0.900(1)	0.063(1)	0.066(1)	0.064(1)	0.271(3)	0.132(2)
P3	0.894(2)	0.901(2)	0.901(2)	0.876(3)	0.952(1)	0.046(1)	0.045(1)	0.045(1)	0.257(3)	0.209(2)
P4	0.765(3)	0.765(3)	0.766(3)	0.890(2)	0.919(1)	0.024(1)	0.025(1)	0.025(1)	0.358(3)	0.283(2)
P5	0.943(2)	0.926(3)	0.943(2)	0.893(4)	0.967(1)	0.061(1)	0.055(2)	0.061(1)	0.177(3)	0.168(3)
P6	0.955(2)	0.950(2)	0.956(2)	0.921(3)	0.992(1)	0.033(1)	0.043(2)	0.033(1)	0.349(4)	0.228(3)
P7	0.859(3)	0.857(3)	0.841(4)	0.891(2)	0.956(1)	0.043(1)	0.043(1)	0.044(1)	0.368(3)	0.297(2)
avg.	0.834	0.830	0.833	0.876	0.928	0.049	0.050	0.049	0.307	0.225

their experiment, in most cases, there is only 0%∼3% code churn when comparing the test version with the training version. In other words, the test version is very similar to the training version. As a result, it is understandable that the reduction strategy learned by Sentinel will have a good performance. However, in our context, we train Sentinel on one program and test it on another program. Normally, this is a cross-project (program) prediction scenario, where the training and test programs are very different. This is one possible reason why Sentinel does not perform well in our study. We emphasize that the conclusion of this paper is obtained by using Sentinel in a cross-project scenario. We do not deny that Sentinel has the possibility of achieving good OP under cross-version scenario.

Second, the optimization objectives and the used basic strategies in Sentinel determine that it is hard to achieve a high OP value. As introduced in Section 4.2.2, Sentinel has two optimization objectives: the GMS and the execution time, which are not directly related to the "order-preserving ability". Furthermore, Sentinel uses OP-ineffective basic strategies such as RMS and COS to generate a reduction strategy. Consequently, it is not surprising that Sentinel will lead to a low OP value.

6.5 What is the Computational Efficiency of OP Compared with GMS and VMS?

Given a kill matrix, *T, M*, and M_s, let us compare the (worst) time complexity of *VMS*, *OP*, and *GMS*.

Complexity of *VMS*. Note that the time complexity of *MS*(*M, T)* is $O(|M|*|T|)$. The reason is that, in the worst case, we need to traverse each element in the matrix. Besides, it is enough to compute MS by traversing once, within which we only need to record and count the killed mutants.

As a result, it is easy to get the upper bound of the complexity of *VMS*: $O(|M|^*|T|) + O(|M_s|^*|T|) = O(|M|^*|T|)$.

Complexity of *OP*. To compute OP, the following random test suite sequence is needed:

$$T = T_0 \supset T_1 \supset T_2 \supset \cdots \supset T_k \neq \emptyset \text{ which satisfies } |T_{i+1}| = int\,(|T_i| \times 0.5), i = 0, 1, \ldots, k-1$$

The complexity of producing such a random sequence is $O(|T|)$. Note that, for each T_i, we should compute *MS* against *M* and M_s. Therefore, the upper bound of the time complexity of *OP* is:

$$O\,(|T|) + \sum_{i=0}^{k}[O\,(|M| * |T_i|) + \mathrm{O}\,(|M_s\,|*|\,T_i|)] = O\,(|T|) + 2\,(O\,(|M| * |T|) + O\,(|M_s| * |T|)) = O\,(|M| * |T|)$$

Indeed, *OP* can be computed faster, since the $MS(M, T_i)$ for any T_i can be obtained by traversing the matrix once. Due to the randomness in generating the test suite sequence, there is a need to repeat the above computation several times to obtain the average *OP*. As a result, the computation of *OP* is slower than the computation of VMS. However, they have the same order of time complexity (big O notation).

Complexity of *GMS*. To compute GMS, there is a need to obtain the minimal test suite T_s. In order to generate T_s, the current practice is to keep deleting a test case that kills the least mutants while keeping the remaining test cases to kill M_s, from the original test suite *T*. If there are several test cases that kill the least mutants, we randomly delete one test case among them. To delete the first test case, in the worst case, we need to traverse each element in the matrix. Therefore, the complexity for this step is $O(|M|^*|T|)$. To delete the second test case, there is a need to traverse the matrix again, with the complexity of $\mathrm{O}(|\mathrm{M}|^*(|\mathrm{T}|-1))$. Under the worst case, the complexity to find the minimal test suite is

$$\sum_{i=0}^{k}(O\,(|M| * (|T| - i)) = \ O(|M| * |T|^2)$$

Even if *GMS* can be obtained at this step, we can see that the time complexity is already higher than *VMS*. In particular, due to the randomness in generating the minimal test suite, there is a need to repeat the above computation several times to obtain the average *GMS*. Under the same number of repetitions, the computation of *GMS* is slower than the computation of *OP*.

In summary, the above analysis reveals that: *GMS* has the highest time complexity, while the computation of *OP* is slower than that of *VMS* but they have the same order of time complexity.

7 IMPLICATIONS

Our study has important implications for both researchers and practitioners. By analyzing the pitfalls of existing indicators, we propose two useful indicators to evaluate the effectiveness of a mutant reduction strategy. Furthermore, we apply these two indicators to compare the effectiveness of five reduction strategies. The detailed implications are listed as follows:

- *Our work warns that existing reduction evaluation indicators should be used with caution.* We show that the existing indicators such as *GMS* and *VMS* have a low ability to distinguish between mutation reduction strategies or even may lead to counter-intuitive conclusions. In particular, for the most commonly used indicator *GMS*, through an example, we point out that the conclusion of GMS goes against the fact and the goal it wants to measure (i.e., number of unique faults). For VMS, it aims to measure the "mutation-score-preserving ability" of a reduction strategy, which is not coincident exactly with the "order-preserving ability". However, the "absolute" MS values have to be compared to make sense. As a result, it is more

important to preserve the relative order than the absolute value. Furthermore, we theoretically prove that a random reduction strategy RMS is the perfect strategy in expectation in terms of VMS (see the appendix). This means that, under VMS, any effective reduction strategy at most performs similarly to RMS. These pitfalls warn that existing indicators are unable to objectively depict the effectiveness of mutation reduction. Therefore, in future studies, the existing indicators such as *GMS* and *VMS* should be used and explained with caution, especially when using them to identify a good reduction strategy.

- *Our work provides simple but useful indicators for mutation reduction evaluation.* From the viewpoint of measurement theory, we show that the essence of mutation reduction evaluation is to evaluate the "order-preserving-ability" of a reduction strategy, i.e., to what extent the mutation score order among test suites is maintained before and after mutation reduction. Furthermore, we proposed two simple but useful indicators, OP and EROP, to measure the "order-preserving-ability" of a reduction strategy. In particular, we provide a light-weight "continuous half-sample" computation approach. Consequently, for a mutation reduction strategy, given a SUT with the original test suite and mutation set, OP and EROP can be efficiently and easily computed. As a result, we suggest using OP and EROP to evaluate mutation reduction effectiveness in practice.
- *Our work discloses the practical value of important mutation reduction strategies.* We apply OP and EROP to compare the effectiveness of five important mutation reduction strategies, including RMS, Sentinel, COS, SMS, and CMS. We find that, when the reduction ratio is less than 50%, in most cases, RMS has a strong "order-preserving-ability" (OP $>$ 0.9) and the other strategies do not exhibit an advantage (EROP $\approx$ 0). Furthermore, CMS exhibits a similar or higher "order-preserving-ability" than RMS, regardless of which mutation reduction ratio is considered. Therefore, from the viewpoint of practical use, RMS is a good option if the reduction ratio is less than 50% and the mutation reduction cost is considered. However, if the mutation reduction cost is not a concern, SMS and CMS are preferred. In particular, from the viewpoint of academic research, there is room to develop more effective mutation reduction strategies. The reason is that SMS and CMS have a EROP well below 0.1 in most cases.

8 THREATS TO VALIDITY

In this section, we discuss the important threats to the construct validity, internal validity, and external validity of our study. Construct validity denotes the extent to which the variables used in our study accurately measure what we purport to measure. Internal validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variable. External validity is the degree to which the results of the research can be generalized to the population under study and other research settings.

8.1 Construct Validity

In our study, we use OP and EROP to measure the effectiveness of a mutation reduction strategy. Ideally, for a SUT with the original test suite T, the OP and EROP values should be estimated using a "continuous subsample" approach from T. However, due to the high computation complexity, it is hard to apply such an approach in practice. To tackle this problem, we use a "continuous half-sample" approach to replace a "continuous subsample" approach to estimate OP and EROP. Therefore, the most important threat is that a "half-sample" approach may lead to biased OP and EROP values. In order to mitigate this threat, we run k (in default, $k = 100$) repeated "continuous half-sample" to generate OP and EROP. Indeed, from the viewpoint of statistical theory, subsample

and half-sample can produce similar estimates [41, 42]. Therefore, this threat should not have a large influence on the construct validity of OP and EROP.

8.2 Internal Validity

The first threat is the selection bias of the subject mutation reduction strategies. In RQ1, we compare five mutation reduction strategies in the experiment when investigating the discrimination ability of OP and EROP. The reasons are two-fold. On the one hand, these five reduction strategies are either very popular or the state-of-the-art reduction strategies. On the other hand, the other reduction strategies, such as a higher-order mutant strategy, may need to introduce new mutants, which violates our experimental principle of "selecting a mutant subset from all mutants" in this study. In the future, we will use more mutation reduction strategies to investigate the discrimination ability of OP and *VMS* to reduce this threat.

The second threat is that the way to generate insufficient test suites may lead to a biased conclusion. In RQ2, for each program, we "delete test cases in RQ1 with even index for each program" to simulate an insufficient test suite. As a result, the generalization of the conclusion under RQ2 may be questionable. To mitigate this threat, we re-run the experiment (see Section 6.3) by randomly generating 100 subsets from the original test set associated with each program. The experimental results show that the conclusion in RQ2 is stable. In this sense, this threat has been minimized as possible as we can.

The third threat is that the specific setting of the step length in reduction ratio may lead to a biased conclusion. In RQ3, we "set the reduction ratio from 95% to 5% with a step length of 5%" to investigate the influence. It is unknown the influence of a different step setting on the conclusion in RQ3. In order to reduce this threat, we used a step length of 1% instead of 5% to re-run the experiment. Consequently, we found that the conclusion in RQ3 remained unchanged. Therefore, this threat has been considered in our study.

The fourth threat is from the ground truth used for evaluating the order-preserving ability of a mutation reduction strategy. Our study uses the mutation score MS based on all the original mutants to derive the order relationship as the ground truth. However, the original mutation score may be inflated by the subsumed mutants. Consequently, it may be argued that we should use subsuming mutation score to derive the order relationship as the ground truth. However, the real subsuming relationships between mutants are difficult to obtain. The current practice is to use a mutant-test kill matrix to approximate the subsuming relationships. This means that the resulting subsuming mutants may be inaccurate. For example, adding new test cases to T may change the resulting subsuming mutants (e.g., the original subsuming mutants may not be subsuming mutants any more). In this sense, it is inappropriate to use subsuming mutation score as the target metric that a mutation reduction strategy should respect to. This is especially true, when we also consider the fact MS is still the most commonly used mutation score in the literature. Therefore, we decide to use MS as the target metric that a mutation reduction strategy needs to conform to. Indeed, OP and EROP are general-purpose measures for quantifying the order-preserving ability when using a new mutation score metric to replace an old mutation score metric, regardless of which new and old metrics are. In other words, in the future work, if a more accurate mutation score metric is developed, OP and EROP are still applicable.

8.3 External Validity

In this study, we use ten subject programs and seven projects with a variety of test suites and mutant sets to investigate the effectiveness of OP and EROP. The overall results show that OP and EROP perform better than the existing indicators such as VSM and *GMS* in mutation reduction

evaluation, especially in distinguishing between mutation reduction strategies. This is due to the fact that OP and EROP aim to measure the "order-preserving" ability, rather than the "mutation-score-preserving" ability. Given this situation, it is reasonable to believe that similar findings would be observed if different programs, test suites, and mutation sets are used. However, we need to explain why we choose these mutation operators, tools, and projects.

Operators. First, the seven default operators are selected to be consistent with the initial settings in [8]; Second, the setting of "all operators" is used to get the conclusions when using as many mutation operators as possible. At this time, the most obvious problem is the duplicated mutants [47]. However, the duplicated mutants will not affect our experimental results based on OP. The proof is in the online supplementary materials.

Tools. PIT is a popular mutation testing tool. In recent empirical study [43], the researchers show that PIT is better than Major and muJava in effectiveness. However, in order to minimize the impact of mutation testing tools on our conclusions, we attach the OP results based on Major in the last part of the appendix.

Projects. We experiment with all successfully built projects in Apache Commons, which are also used in previous work [8, 11, 26, 27, 29–31]. However, our conclusions will still be limited to specific projects. The generalization of the conclusions is always threatened. In future work, more benchmarks should be taken into consideration.

9 CONCLUSION AND FUTURE WORK

The motivation of this study is to understand the pitfalls of existing mutation reduction evaluation indicators and explore how to objectively measure the ability of a mutation reduction strategy to maintain test suite effectiveness evaluation. In the last decades, many mutation reduction strategies have been proposed, and their reduction effectiveness are evaluated by *GMS* and *VMS*. From our analysis and experimental results, however, we can see that *GMS* and *VMS* may lead to misleading conclusions and have a weak ability to distinguish various mutation reduction strategies. The fundamental reason is that they are not originally designed to directly measure the extent to which the partial order under the original mutants remains unchanged in the partial order under the reduced mutants. Indeed, from the viewpoint of measurement theory, for mutation reduction evaluation, the essence is to evaluate the "order-preserving ability" of a mutation reduction strategy, which is missing in our community. To this end, we propose two indicators, OP and EROP, for "order-preserving ability" evaluation. OP evaluates the degree to which the reduction strategy maintains a relative evaluation of a test suite, while EROP indicates the effort-aware OP relative to a random reduction strategy. In particular, OP and EROP can be easily and efficiently computed. Our experimental results show that OP and EROP have a strong ability to distinguish among various mutation reduction strategies. Furthermore, we find that SMS and CMS are more effective than the other mutant selection strategies under OP and EROP.

In the future, on the one hand, we plan to study how to employ OP and EROP to develop more effective mutation reduction strategies. As shown in our experimental results, the average OP of existing approaches is lower than 0.9, indicating a large room for improvement. This is especially true for Sentinel. As the first step toward this direction, we will hence explore how to improve Sentinel: first, we will modify the basic strategy of Sentinel by adding SMS into the basic strategy set or using CMS instead of grouping mutants; second, we will change the objective function to be OP oriented. On the other hand, we plan to apply OP and EROP to the evaluation of high-order mutant reduction strategies. In our current study, we focus on these mutation reduction strategies without adding new mutants. However, OP and EROP can also be directly used to evaluate high-order mutant selection strategies. Such an evaluation will inform developers whether and which high-order mutant reduction strategies are effective in practice.

APPENDICES

A RMS CAN MAINTAIN THE MUTATION SCORE IN MATHEMATICAL EXPECTATION

Assume that T is the test suite on the SUT with the original mutant set M. For RMS, let M' be the set of the mutants reduced from M. The random reduction strategy RMS always maintains the following property:

$$E[MS(M', T)] = MS(M, T)$$

where E is a mathematical expectation. In the following, we give a brief proof. Assume that $|M| = n$, $|M'| = m$, and $|KM(M, T)| = k$. Then, we have:

$$E\left[MS\left(M', T\right)\right] = E\left[\frac{|KM\left(M', T\right)|}{|M'|}\right] = \frac{E\left[|KM\left(M', T\right)|\right]}{|M'|} = \frac{1}{m}E\left[|KM\left(M', T\right)|\right]$$

$$= \frac{1}{m}\frac{1}{C_n^m}\sum_{i=0}^{k} iC_k^i C_{n-k}^{m-i}$$

$$= \frac{1}{m}\frac{1}{C_n^m}\sum_{i=1}^{k} iC_k^i C_{n-k}^{m-i} = \frac{1}{m}\frac{1}{C_n^m}\sum_{i=1}^{k} i\frac{k!}{(k-i)!i!}C_{n-k}^{m-i} = \frac{1}{m}\frac{1}{C_n^m}\sum_{i=1}^{k} k\frac{(k-1)!}{(i-1)!\left((k-1)-(i-1)\right)!}C_{n-k}^{m-i}$$

$$= \frac{1}{m}\frac{1}{C_n^m}k\sum_{i=1}^{k} C_{k-1}^{i-1}C_{n-k}^{m-i} = \frac{1}{m}\frac{C_{n-1}^{m-1}}{C_n^m}k = \frac{1}{m}\frac{\mathrm{m}}{n}k = \frac{k}{n} = MS\left(M, T\right)$$

B IS THE CONCLUSION OF OP GREATLY AFFECTED BY THE MUTATION TESTING TOOL?

In the manuscript, we used *PIT* as the tool of mutation testing. Then the question is that can the conclusion be extended to other tools? To investigate this question, we use *Major* to execute mutation testing. However, *Major* is executed by ant and most of the used projects are not built by ant. To this concern, we use *defects4j* [46], which is a widely used dataset for software testing. *defects4j* provides the option for mutation analysis based on *Major*. It saves us the work of building projects one by one. Meanwhile, most of the programs/projects used in our manuscript are used in *defects4j*. However, we use the different versions compared to *defects4j*.

We designed the experiment as follows: For a program we used in the manuscript, if it exists in the last fixed (i.e., bug-free) version for the corresponding project in the *defects4j* dataset, we use *Major* to analyze the mutants. For example, the program "Option" is in project "Cli". Then we use the command: "defects4j checkout -p Cli -v 40f" to obtain the last fixed version of Cli. After

Table B. The OP Values Computed Against Major

	OP				
Project	RMS	Sentinel	COS	SMS	CMS
Crypt	0.863	0.891	0.869	0.970	0.863
StringUtils	0.797	0.798	0.790	0.902	0.867
Md5Crypt	0.754	0.740	0.753	0.782	0.868
DefaultParser	0.892	0.864	-	0.906	0.929
Option	0.549	0.376	0.450	0.557	0.637
ResizableDoubleArray	0.629	0.599	0.609	0.755	0.677
UnixCrypt	0.709	0.702	0.721	0.758	0.717
HeplFormatter	0.695	0.655	0.691	0.872	0.77
avg.	0.736	0.703	0.698	0.813	0.791

that, if "Option.java" is in it, we run the test cases one by one to obtain the kill matrix. Other experimental settings are the same as RQ1 in the manuscript. Note that code of "Option" used here may be different to the "Option" in the manuscript. Then the result is showed in Table B. From Table B, we can find that the overall conclusion is consistent to the manuscript: SMS and CMS are better than other 3 strategies. To conclude, when using OP to compare several strategies, the overall conclusion is not affected by the mutation testing tools.

ACKNOWLEDGMENTS

We are very grateful to anonymous reviewers and the editor for their very insightful comments and very helpful suggestions, which dynamically improved the quality of our manuscript.

REFERENCES

[1] Y. Jia and M. Harman. 2011. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678.
[2] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Traona, and M. Harmanb. 2019. Mutation testing advances: An analysis and survey. *Advance in Computers* 112, (2019), 275–378.
[3] A. S. Namin, J. H. Andrews, and D. J. Murdoch. 2008. Sufficient mutation operators for measuring testing effectiveness. *ICSE* 2008, 10–18.
[4] J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. 1996. An experimental determination of sufficient mutation operators. *ACM Transactions on Software Engineering and Methodology* 5, 2 (1996), 99–118.
[5] P. Delgado-Prez, S. Segura, and I. Medina-Bulo. 2017. Assessment of C++ object-oriented mutation operators: A selective mutation approach. *Software Testing, Verification and Reliability* 27, 4–5 (2017), n/a-n/a.
[6] M. E. Delamaro, J. Offutt, and P. Ammann. 2014. Designing deletion mutation operators. *ICST* 2014, 11–20.
[7] M. E. Delamaro, L. Deng, V. H. S. Durelli, N. Li, and J. Offutt. 2014. Experimental evaluation of SDL and one-op mutation for C. *ICST* 2014, 203–212.
[8] G. Guizzo, F. Sarro, J. Krinke, and S. Vergilio. 2020. Sentinel: A hyper-heuristic for the generation of mutant reduction strategies. *IEEE Transactions on Software Engineering* 2020. https://doi.org/10.1109/TSE.2020.3002496
[9] M. Papadakis and N. Malevris. 2010. An empirical evaluation of the first and second order mutation testing strategies. *ICST* 2010, 90–99.
[10] B. Kurtz, P. Ammann, J. Offutt, M. E. Delamaro, M. Kurtz, and N. Gokce. 2016. Analyzing the validity of selective mutation with dominator mutants. *FSE* 2016, 571–582.
[11] D. Gong, G. Zhang, X. Yao, and F. Meng. 2017. Mutant reduction based on dominance relation for weak mutation testing. *Information and Software Technology* 81, (2017), 82–96.
[12] Y. Jia and M. Harman. 2009. Higher order mutation testing. *Information and Software Technology* 51, (2009), 1379–1393.
[13] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng. 2014. Mutant subsumption graphs. *ICSTW* 2014, 176–185.
[14] R. Just, B. Kurtz, and P. Ammann. 2017. Inferring mutant utility from program context. *ISSTA* 2017, 284–294.
[15] R. J. Lipton and F. G. Sayward. 1978. The status of research on program mutation. In *Proceedings of the Workshop on Software Testing and Test Documentation* 1978, 355–373.
[16] A. P. Mathur and W. E. Wong. 1994. An empirical comparison of data flow and mutation-based test adequacy criteria. *Software Testing, Verification and Reliability* 4, (1994), 9–31.
[17] W. E. Wong and A. P. Mathur. 1995. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software* 31, 3 (1995), 185–196.
[18] B. Kurtz, P. Ammann, and J. Offutt. 2015. Static analysis of mutant subsumption. *ICST* 2015, 1–10.
[19] L. Zhang, S. Hou, J. Hu, T. Xie, and H. Mei. 2010. Is operator-based mutant selection superior to random mutant selection? *ICSE* 2010, 435–444.
[20] L. Zhang, M. Gligoric, D. Marinov, and S. Khurshid. 2013. Operator-based and random mutant selection: Better together. *ASE* 2013, 92–102.
[21] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov. 2015. Guidelines for coverage-based comparisons of non-adequate test suites. *ACM Transactions on Software Engineering and Methodology* 24, 4 (2015), 1–33.
[22] M. Yu and Y. Ma. 2019. Possibility of cost reduction by mutant clustering according to the clustering scope. *Software Testing, Verification and Reliability* 29, 1–2 (2019), n/a-n/a.
[23] A. P. Mathur. 1991. Performance, effectiveness, and reliability issues in software testing. *COMPSAC* 1991, 604–605.
[24] S. Hussain. 2008. Mutation Clustering. *M.S. thesis London, UK King's College.* 2008.

[25] P. Zhang, Y. Li, W. Ma, Y. Yang, L. Chen, H. Lu, Y. Zhou, and B. Xu. 2020. CBUA: A probabilistic, predictive, and practical approach for evaluating test suite effectiveness. *IEEE Transactions on Software Engineering* 2020. https://doi.org/10.1109/TSE.2020.3010361
[26] A. Pizzoleto, F. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro. 2019. A systematic literature review of techniques and metrics to reduce the cost of mutation testing. *The Journal of Systems and Software* 2019, 157.
[27] R. Gopinath, I. Ahmed, M. A. Alipour, C. Jensen, and A. Groce. 2017. Mutation reduction strategies considered harmful. *IEEE Transactions on Reliability* 66, 3 (2017), 854–874.
[28] Apache Commons. http://commons.apache.org.
[29] R. Gopinath, M. A. Alipour, I. Ahmed, C. Jensen, and A. Groce. 2016. On the limits of mutation reduction strategies. *ICSE* 2016, 511–522.
[30] T. Laurent, M. Papadakis, M. Kintis, C. Henard, Y. L. Traon, and A. Ventresque. 2017. Assessing and improving the mutation testing practice of PIT. *ICST* 2017, 430–435.
[31] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. 2014. Are mutants a valid substitute for real faults in software testing? *FSE* 2014, 654–665.
[32] PIT. http://pitest.org/.
[33] Cobertura. https://github.com/cobertura.
[34] PIT mutators. https://pitest.org/quickstart/mutators.
[35] P. S. Kochhar, F. Thung, and D. Lo. 2015. Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. *SANER* 2015, 560–564.
[36] Y. T. Chen, R. Gopinath, A. Tadakamalla, M. D. Ernst, R. Holmes, G. Fraser, P. Ammann, and R. Just. 2020. Revisiting the relationship between fault detection, test adequacy criteria, and test set size. *ASE* 2020, n/a-n/a.
[37] L. Inozemtseva and R. Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. *ICSE* 2014, 435–445.
[38] M. Papadakis, C. Henard, M. Harman, Y. Jia, and Y. L. Traon. 2016. Threats to the validity of mutation-based test assessment. *ISSTA* 2016, 354–365.
[39] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. 2017. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2017), 1–18.
[40] J. Cohen. 1988. Statistical power analysis for the behavioral sciences. *Routledge. ISBN* 978-1-134-74270-7.
[41] G. Babu. 1992. Subsample and half-sample methods. *Annals of the Institute of Statistical Mathematics* 44, 4 (1992), 703–720.
[42] J. Shao and X. Shi. 1989. Half-sample variance estimation. *Communications in Statistics: Theory and Methods* 18, 11 (1989), 4197–4210.
[43] M. Kintis, M. Papadakis, A. Papadopoulos, E. Valvis, N. Malevris, and Y. L. Traon. 2018. How effective are mutation testing tools? An empirical analysis of Java mutation testing tools with manual analysis and real faults. *Empirical Software Engineering* 23, 4 (2018), 2426–2463.
[44] M. Kintis, M. Papadakis, and N. Malevris. 2010. Evaluating mutation testing alternatives: A collateral experiment. *APSEC* 2010, 300–309.
[45] M. Papadakis, T. T. Chekam, and Y. L. Traon. 2018. Mutant quality indicators. *ICSTW* 2018, 32–39.
[46] R. Just, D. Jalali, and M. D. Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. *ISSTA* 2014, 437–440.
[47] M. Papadakis, Y. Jia, M. Harman, and Y. L. Traon. 2015. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. *ICSE* 2015, 936–946.

Received July 2021; revised November 2021; accepted December 2021