Threats to the Validity of Mutation-Based Test Assessment

Mike Papadakis
University of Luxembourg
michail.papadakis@uni.lu

Christopher Henard
University of Luxembourg
christopher.henard@uni.lu

Mark Harman
University College London
mark.harman@ucl.ac.uk

Yue Jia
University College London
yue.jia@ucl.ac.uk

Yves Le Traon
University of Luxembourg
yves.letraon@uni.lu

ABSTRACT

Much research on software testing and test techniques relies on experimental studies based on mutation testing. In this paper we reveal that such studies are vulnerable to a potential threat to validity, leading to possible Type I errors; incorrectly rejecting the Null Hypothesis. Our findings indicate that Type I errors occur, for arbitrary experiments that fail to take countermeasures, approximately 62% of the time. Clearly, a Type I error would potentially compromise any scientific conclusion. We show that the problem derives from such studies' combined use of both subsuming and subsumed mutants. We collected articles published in the last two years at three leading software engineering conferences. Of those that use mutation-based test assessment, we found that 68% are vulnerable to this threat to validity.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

Mutation testing, test assessment, subsuming mutants

1. INTRODUCTION

Much work on the empirical and experimental evaluation of software test technique effectiveness and efficiency relies on insertion of seeded faults. These faults, typically machine-generated, are known as mutants [10], allow experimenters to compare test techniques by comparing the number of mutants that they detect. This is called *mutation-based test assessment*. Mutation-based test assessment is a popular approach, used in approximately 19% of the software testing studies, reported in the last two years in the three leading conferences, ISSTA, (ESEC)FSE and ICSE.

Typically, researchers select a set of mutation operators, with which they insert mutants into a set of programs (used as the subjects of the experimental study). A mutant that behaves differently from the original program with respect to a test input t, is said to be killed by t. In this way, mutation-based test assessment allows researchers to explore the relative effectiveness of techniques.

In parallel with the use of mutant-based test assessment, researchers have developed the theory and practice of mutation testing itself. Much research has focused on the large number of mutants applicable, and consequent cost of thorough mutation testing. Many techniques have been introduced to reduce the number of mutants apparently needed, including random sampling [44, 52], principled selection [33, 36, 52] and higher order techniques [22, 44].

These techniques share the property that they concentrate on reducing the cost of the method by *excluding* mutants. They do this by demonstrating, experimentally, that removing mutants causes only an "acceptably low" reduction in test effectiveness. However, no previous work has measured the harmful effect of *including* certain mutants that really ought to be excluded to avoid skewing results.

Many software testing researchers draw on recent [24] and established [6] research that demonstrates a strong relationship between mutants and real faults. These results are comforting, because they indicate that the results obtained from mutation testing may also carry over to findings that would pertain to real faults in real systems. This removes one significant potential threat to the validity.

The current state of mutation-based test assessment is thus to rely on the assumption that mutants are good at simulating real faults and that all mutants included in these studies are, essentially, equally important and valuable in their contribution to the overall assessment. Unfortunately, fault realism is not the only threat to the validity of mutation-based test assessment studies; the inclusion of some kinds of mutant actually *harms* the test assessment.

That is, as we shall show in this paper, there are serious threats to validity that lie at the very heart of mutant-based test assessment: the presence of subsumed mutants (also known as redundant mutants), can artificially inflate the apparent ability of a test technique to detect faults. Although previous authors have discussed the use of mutation scores based on subsuming mutants only [3, 22, 28], no previous studies have investigated the harmful effect of including subsumed mutants in mutation-based test assessment (which is widely used practice; approximately one quarter of all test assessment studies use this approach[1]).

[1]We approximate this number by conducting a survey on the papers published in last two years in the three leading conferences, ISSTA, (ESEC)FSE and ICSE.

As our results show, the "subsumed mutant threat" can lead to the commission of Type I errors (incorrectly rejecting the null hypothesis, when it is in fact true). Thus, an experiment that compares two techniques, will seek to reject a Null Hypothesis that the performance metrics of the two techniques are drawn from the same overall distribution, thereby providing statistical evidence that one technique is superior. Our results show that subsumed mutants must be removed before such a test is performed, in order to avoid threats to the validity of such claims.

We present evidence from multiple experiments conducted with sets of mutants applied to 5 widely-used benchmark Unix utilities (Grep, Sed, Flex, Make and Gzip), to back these claims. We show that the inclusion of subsumed mutants in arbitrary sets of mutation-based test assessment experiments leads to 62% of these experiments committing a Type I error.

We found that fewer than 5% of all mutants are subsuming (and not subsumed themselves). The remaining 95% of mutants are subsumed by some other mutants. Clearly the presence of so many subsumed mutants can cause distorting effects, and this remains the case, even as the statistical effect size of test technique differences increases. Indeed, we show that there will remain errors when two arbitrary test techniques differ in their assessment by as much as 10%, even though they both yield identically sized test suites.

Finally, our results also indicate that the proportion of all mutants killed is not generally strongly correlated with the proportion of subsuming mutants killed. Therefore, using all mutants can skew results leading to serious threats to validity, and findings that are not well correlated with results from the more reliable subsuming mutant set.

Taken together, our findings provide strong evidence that authors of studies concerned with mutation-based test assessment should remove subsumed mutants before commencing mutation-based experimentation. We report on a simple sample of recent publications, which indicates that our findings would require a fundamental change in the experimental practice adopted by approximately two thirds of the mutation-based test assessment literature.

2. MUTATION ANALYSIS

2.1 Test Criteria and Mutation Testing

Testing criteria are used to drive several forms of testing process. Thus, they specify what are the requirements, herein after termed test requirements, that should be fulfilled by the test cases [4, 54]. Hence, test criteria are used for guiding the test generation (generating test cases that fulfill the uncovered requirements) [14, 43], selecting representative tests among a large set of test cases [48], assessing the level of test thoroughness (helping decide when to stop) and they are also used as a comparison basis for other test criteria [6, 22] and test generation or test selection methods.

Mutation testing seeks to reveal a deliberately introduced set of artificial defects. A program with an introduced defect is called a *mutant*, or mutant program, and it is obtained by applying basic syntactic rules. For instance, the following fragment of code "`if (a > b)`" can be mutated into "`if (a` $\geq$ `b)`" by applying the syntactic change, which consists of replacing the relational operator ">" by "$\geq$". The resulting program (embedding this altered fragment of code) is a mutant, because it is exactly the original program excepted for the syntactic change that has been made. The set of rules defining the possible changes are named *mutation operators*, and the application of all the rules allows the tester to obtain a set of mutants differing from the original program in one specific part of the code.

Mutants are produced to evaluate the ability of the test suite to expose them. The idea behind mutation is to assess whether the test suite will exercise the parts of the code that were changed, and whether the program execution is sufficiently "sensitive" to propagate the altered code to the observable output of the program. Thus, we say that a mutant is *killed* if the output of this program mutant is different from the output of the original program. If the output is the same, the mutant is called *live*. By executing the test suite towards all the mutants, we can evaluate the *mutation score* (MS), which is the ratio of the killed mutants to the number of killable mutants[2]. As a result, mutation score quantifies the level of test thoroughness.

2.2 Equivalent, Duplicated, Trivial and Subsumed Mutants

Killing all the mutants is not possible due to the fact that mutant programs can form functionally equivalent versions of the original program. These mutants are called *equivalent* mutants and must be removed from consideration when using mutation as a test assessment metric. Equivalent mutants form one of the main problems of mutation testing [4, 42, 47] because identifying them is an undecidable problem [8]. However, in the case that mutation is used for comparison, then, it might be possible to collect a large set of test data and sweep out all live mutants. Since the focus is on the differences between the techniques this practice is regarded to be adequate [6].

Recently, Papadakis *et al.* [42] provided evidence that such a practice might be problematic, due to the existence of numerous equivalences between killable mutants. Hence, it was advised that future research should remove from consideration all *duplicated* mutants, i.e., mutants equivalent to one another but not to the original program, in order to avoid counting them multiple times.

Several researchers have proposed reducing the cost of mutation based on the notion of subsuming mutants [22, 28]. Kaminski *et al.* [27] and Just *et al.* [25] identified some cases of subsumed mutants that were produced by the relational and logical mutation operators. Hence, they proposed using a restrictive version of the relational and logical operators.

In this paper, we investigate the harmful effects of subsuming mutants on other studies that used mutation as a basis of comparison (to decide which testing methods perform better). Given a test suite, a mutant *subsumes* another one, if it is killed by a subset of the test cases the kill the other. That is, whenever the *subsuming* mutant is killed, the *subsumed* mutant is also killed. We discuss the process of identifying subsuming mutants in the following section.

2.3 Subsuming Mutants

Program mutation is widely used as a means of assessing testing methods [5, 23]. To this end, the mutation score is used as a basis of comparison to decide which testing methods perform better, i.e., the winning method being the one yielding the highest score. However, such comparisons

[2]Some mutants cannot be killed because there is no test case that can be written to kill them.

do not take into account the fact that not every mutant is of equal value [22]. Indeed, some mutants are killed by most of the test cases, some are duplicated, some are hard to kill and some are equivalent to the original program.

As a result, it is important to remove mutants which are either equivalent or duplicated when using mutation for comparing different testing methods [42]. However, many mutants tend to be killed "collaterally" in most of the cases [3, 22, 28], because they are subsumed. Thus, they do not contribute to the testing process, despite being considered for the calculation of the mutation score. As a result, the mutation score is inflated and becomes skewed.

To avoid the inflation problem, only the mutants contributing to the mutation score should be used. We call these mutants *subsuming mutants*. Subsumption is defined, according to Ammann *et al.* [3], as follows: "one mutant subsumes another if at least one test kills the first and every test that kills the first also kills the second". It is noted that mutant subsumption is reflexive (every mutant trivially subsumes itself). We define *subsuming mutation score* (MS^*) as the ratio of the subsuming mutants that are killed by the tests under assessment to the number of all subsuming ones. Thus, for example if we have a mutant set of 3 mutants (named as m_1, m_2 and m_3) and a test set of 3 tests (named as t_1, t_2 and t_3), where m_1 is killed by t_1, m_2 by t_1 and t_2, and m_3 by t_3, we have two subsuming mutants, m_1 and m_2. Then the test set $<t_1, t_2>$ kills 66.7% of all the mutants and 50% of the subsuming ones.

The process of identifying the subsuming mutants is an NP-complete problem [3], but it can be approximated with a greedy algorithm, such as the one of Algorithm 1. From a set of mutants S, the algorithm starts by removing the live and duplicate mutants (lines 2 and 3). Then, the most subsuming mutant is identified (lines 8 to 15). This mutant, when it is killed, leads to the highest number of other mutants being "collaterally" killed. This most subsuming mutant is then added to the subsuming mutants set D (line 16) and the subsumed mutants are removed from S (line 17). This process is repeated until S is empty.

Algorithm 1: Subsuming Mutants Identification

Input: A set S of mutants, a set T of test cases, a matrix M of size $|T| \times |S|$ such as $M_{ij} = 1$ if test$_i$ kills mutant$_j$
Output: The subsuming mutant set D from S

```
1  D = ∅
                                        /* Remove live mutants */
2  S = S \ {m ∈ S | ∀i ∈ 1..|T|, M_ij ≠ 1}
                                    /* Remove duplicate mutants */
3  S = S \ {m ∈ S | ∃m' ∈ S | ∀i ∈ 1..|T|, M_ij(m) = M_ij(m')}
4  while (|S| > 0) do
5  |  maxSubsumed = 0
6  |  subsumedMut = null
7  |  maxMutSubsuming = null
                            /* Select the most subsuming mutant */
8  |  foreach (m ∈ S) do
9  |  |  sub_m = {m' ∈ S|∀i ∈ 1..|T|, (M_ij(m) = 1) ⇒ (M_ij(m') =
   |  |  1)}
10 |  |  if (|sub_m| > maxSubsumed) then
11 |  |  |  maxSubsumed = |sub_m|
12 |  |  |  maxMutSubsuming = m
13 |  |  |  subsumedMut = sub_m
14 |  |  end
15 |  end
                          /* Add the most subsuming mutant to D */
16 |  D = D ∪ {maxMutSubsuming}
       /* Remove the subsumed mutants from the remaining */
17 |  S = S \ subsumedMut
18 end
19 return D
```

3. RESEARCH QUESTIONS

We use the term *mutant inflation effect* for the skew in the mutation score that is caused by subsumed mutants.

Previous research has focused on reducing the cost of mutation by removing some redundant mutants [25, 27, 42]. Therefore, our first research question regards the prevalence of subsuming mutants, after removing those identified by the current state of the art, i.e., the use of restrictive operators that produce less subsumed mutants [25, 27] and using the Trivial Compiler Equivalence (TCE) technique for removing duplicated mutants [42]. Thus, we seek to investigate:

RQ1. What ratio of mutants are subsuming, i.e., contributing to the testing process?

Answering this question will help testers understand the extent of "noise" in the mutation score measurement. Of course demonstrating that subsumed mutants exist and they are numerous, does not necessarily indicate that the mutation score measurement is adversely altered by them or that technique comparisons based on them are biased by them. Also, there is no evidence that subsuming mutants do not suffer from the same problem. Thus, we investigate the following question:

RQ2. Do randomly selected sets of mutants suffer from the mutant inflation effect?

We answer this question by visualizing the relation of killing randomly sampled mutants, of varying sizes, with the mutation score (MS), computed based on all mutants, and the subsuming mutation score (MS^*).

We demonstrate that sampled mutants suffer from the inflation effect and thus, the measurement becomes skewed, as the number of mutants increases. This provides evidence suggesting that subsuming mutants can deflate the mutation score (MS) measurement.

A large body of literature uses mutation analysis as a comparison basis between techniques. In these cases, mutants are used to measure test thoroughness and determine the most effective method. Given the fact that large proportions of mutants are subsumed, an emerging question is to what extent these mutants can influence the abovementioned cases. Therefore, we ask the following question:

RQ3. Does the mutation score (MS), computed based on all mutants, correlate with the subsuming mutation score (MS^*)?

Knowing the level of this correlation can provide evidence in supporting (or not) the validity of the mutation-based assessment studies. In particular, in case there is a strong correlation, we can infer that the influence of the subsumed mutants on software testing experiments is minor. Otherwise, the effects of the subsumed mutants may be distorting.

The correlations reflect the influence of the subsumed mutants on software testing experiments. However, from these data we cannot infer the chance that researchers make wrong decisions in their test experiments, which involve mutation-based assessment. Therefore, we seek to investigate the extent to which subsumed mutants can lead to an erroneous judgment, with respect to test effectiveness, when comparing two arbitrary testing methods. Hence we ask:

RQ4. What is the chance of committing a Type I error when comparing the quality of test sets, produced by arbitrary test techniques using mutation score (MS)?

We answer this question by simulating a scenario that compares two hypothetical arbitrary testing methods. Thus, we randomly and repeatedly select two sets of test suites which have the same size and statistically significant differences in their mutation scores. We then compute the number of times that there is no statistical significance with respect to the subsuming mutants, i.e., with respect to the MS^* measure. This process simulates the Type I errors in arbitrary sets of mutation-based test assessment. The ratio of them represents the chance of committing such an error.

A positive answer to this question is critically important since it raises concerns regarding the validity of software testing studies that use mutation-based test assessment experiments.

Type I errors computed in RQ4 are based on sets of data that are statistically different. However, two sets might have statistically significant differences but at the same time their actual differences can be small. This is measured by the statistical effect size. Therefore, we seek to measure the sensitivity of the mutation score in committing an error when the two examined sets differ statistically with varying effect size differences, i.e., the mutation scores of the two sets differ by either 2%, 4%, 6%, 8%, or 10% on average. This leads us to our last research question:

RQ5. When comparing testing methods, how sensitive is the mutation score measurement in committing Type I errors when the examined sets of data have varying effect sizes, i.e., when their mutation scores differ by some percentage?

The answer to this question will estimate the chance to have an error when a method is better than another one by some percentage with respect to MS. It is natural to expect that as the effect size goes up, the number of errors will reduce. However, we are interested to know whether the error disappears completely.

4. METHODOLOGY

4.1 Definition of the experiment and analysis procedure

To answer the stated research questions we used five open-source programs with mature test suites and performed our analysis on them. Details regarding the programs and tools of the experiment can be found in Sections 4.3 and 4.4 respectively. We generated our analysis set of mutants, per program, using all the mutation operators supported by our tool (see Section 4.4). Then, we discarded all the trivially duplicated ones, using the TCE method of Papadakis *et al.* [42]. We carried out this process in order to avoid reporting large numbers of subsumed mutants that can be easily identified and removed. Along the same lines, our tool uses the restrictive mutation operators proposed by Kaminski *et al.* [27] and Just *et al.* [25]. We also discarded all mutants that were not killed by the available test suites. We call the resulting sets of mutants the *analysis mutant sets*. This is a usual process in mutation-based test assessment (see Andrews *et al.* work [6]).

To answer RQ1, we computed the number of subsuming mutants. This was done based on Algorithm 1. We report their ratios, per program, over the analysis mutant sets. These ratios represent an estimation of the number of mutants that have an effect in the test selection process. Similarly, since killing these mutants results in killing all the mutants, they can also be seen as the set of mutants that a tester needs to consider in order to design test cases according to the mutation testing criterion.

For RQ2 we randomly picked subsets of mutants of size of 0.5%, 1%, 2%, 4%, 8%, 16%, 32% and 64% of the analysis mutant set. We took 30 sets per size considered and per program. For every mutant set, we randomly selected tests that kill its mutants. This test selection was performed by randomly picking tests, executing them with the mutants and keeping only those that kill additional mutants (not killed by the previously selected tests). This process can be seen as the test design process followed by a tester where iteratively live mutants are selected and tests that kill them are designed. Overall, by using mutant sets of various sizes, selecting test cases that kill them, and measuring their ability to kill other mutants, we can explore the inflation effect. If, the more mutants we consider, the more skew we observe then the inflation effect is important. Ideally, we would like to have a relation that can clearly distinguish among all the different mutant sets. We also measured the ratio of the subsuming mutants that are killed by the tests, specifically selected to kill the mutant sets. Recall that subsuming mutants were selected by using the Algorithm 1.

Since we cannot know the representative experiments that might be used to perform some arbitrary mutation-based assessment, we constructed multiple randomly constructed experiments to simulate the effects of arbitrary choice of experiments. Thus, to address RQ3, we randomly selected 1,000 sets of tests for 30 different test suite sizes of, i.e., with sizes of 3, 6, 9, ..., 90, and measured the correlation between mutation score (MS) and subsuming mutation score (MS^*). To calculate the correlation, we used three correlation coefficients: Kendall, Pearson and Spearman. Additional details regarding the correlation analysis can be found in Section 4.2. Different test suite sizes were used to investigate the influence of test quality to the reported correlations. Following the suggestions made in literature, i.e., according to previous works [21, 32], we use different sizes of test suites to represent the various levels of test quality.

To simulate a scenario that compares two hypothetical testing methods and answer RQ4, we randomly and repeatedly select two sets of ten test suites that have the same size and perform a statistical comparison in their mutation scores. We considered test suite size of 3, 6, 9, ..., 90. Then we computed the mutation score for each set and performed a Mann-Whitney statistical test, with a significance level of 0.05, to see whether they have statistically significant differences. In case they have not, we discarded them and started over. In case they are significantly different, we performed a second time the statistical test to see whether they have statistically significant differences according to the subsuming mutants, using the MS^* metric, and we recorded the number of disagreements, i.e., the number that both mutation score and subsuming score judge that one set has higher score than the other. We repeated this process 30 times. Therefore, the number of disagreements estimates the chance of Type I errors in the testing experiments.

Finally, to answer RQ5 we constructed random sets of test suites, of size 3, 6, 9, ..., 90, and categorize them according to mutation score. Then we randomly chose 30 sets of 10 pairs, of the same test size, that have 2%, 4%, 6%, 8% and 10% differences in their mutation score, i.e., effect size differences. We verified that in all the 30 sets differ statistically. Then we count the number of times that mutation score measure agrees with that of the subsuming mutants, i.e., agreement means that according to both mutation score (MS) and subsuming score (MS^*) the one set has higher score than the other. The difference from the previous case, RQ4, is that we measure the sensitivity to errors when we control both test size and average mutation score difference between the two considered sets of data.

4.2 Statistical Analysis

We perform a correlation analysis to evaluate whether the mutation score, when considering all mutants, correlates with the subsuming mutation score (see Section 5.3). To this end, we use three correlations measures: *Kendall rank coefficient (τ)* (Tau-a), *Pearson product-moment correlation coefficient (r)* and *Spearman's rank coefficient (ρ)*. In all cases, we considered the 0.05 significance level.

The Kendall rank coefficient, often referred to as Kendall τ, measures the similarity in the ordering of the mutation scores. In this paper, we measure the mutation score MS and the subsuming mutation score MS^* when using 3, 6, ..., 90 test cases. We obtain pairs of the type (MS_3, MS_3^*), (MS_6, MS_6^*), ..., (MS_{90}, MS_{90}^*). Any two pairs (MS_i, MS_i^*), (MS_j, MS_j^*) are said to be concordant if either both $MS_i > MS_j$ and $MS_i^* > MS_j*$, or both $MS_i < MS_j$ and $MS_i^* < MS_j*$, and they are said to be discordant in the case $MS_i > MS_j$ and $MS_i^* < MS_j*$ or both $MS_i < MS_j$ and $MS_i^* > MS_j*$. The coefficient, τ, is calculated as the difference between the number of concordant and discordant pairs, divided by the total number of pairs.

The Pearson product-moment correlation coefficient (r) is a coefficient obtained by calculating the covariance between the MS and MS^* values, divided by the product of the standard deviation of the MSs and MS^*s. Similarly, the Spearman's ρ coefficient is the Pearson product moment between the ranked MSs and MS^*s. These three coefficients take values from -1 to 1. A coefficient of 1, or -1, indicates a perfect correlation while a zero coefficient denotes the total absence of correlation.

To evaluate whether the achieved mutation scores MSs and subsuming mutation scores MS^* are significantly different, we use a Mann-Whitney U Test performed at the 5% significance level. This statistical test yields a probability called p-value which represents the probability that the MSs and MS^* are equal. Thus, a p-value lower than 5% indicates that the two metrics are statistically different. We use an unpaired and two-tailed U test, because there's no relationships between each of the runs and because we make no assumptions regarding which of the MS or MS^* outperforms the other.

4.3 Programs Used

We use the following programs that we obtained from the Software-artifact Infrastructure Repository [12]: Grep, Sed, Flex, Make, Gzip. We selected these programs because they are real world C programs that have been widely used in software engineering research and they are also accompanied by mature test suites which are important for conducting our experiments. Grep and Sed are regular expressions-based processing tools for respectively searching and parsing/transforming text. Flex is a tool which generates lexical analyzers, Make is a tool for automatically building libraries and executable programs and Gzip is a popular file compressor. Table 1 records details regarding our subjects. For each one of them, it records its version, lines of code, number of the used test cases and mutants.

Table 1: The subject programs used in the experiments. For each of them, its versions, lines of codes, number of test cases together with the number of non-duplicated mutants are presented.

Program	Version	LoC	Test Cases	Mutants
Grep	2.4	12,826	440	11,395
Sed	4.0.8	18,687	324	4,330
Flex	2.5.2	12,249	500	11,693
Make	3.78.1	20,461	111	22,023
Gzip	1.2.2	5,048	156	5,338

4.4 Mutation Operators

We built a prototype mutation testing tool based on a program matching and transformation tool called Coccinelle [40]. It operates on the source code by performing program transformations using a semantic patch language in which the sought transformations are specified. We wrote the employed mutation operators as program transformations rules. Following the recommendations from the literature, i.e., [4, 6, 23, 36] we used the following operators: Arithmetic (AOR), Logical Connector Replacement (LCR), Relational (ROR), Unary Operator Mutation (UOM), Arithmetic assignment mutation (OAAA), Bitwise operator mutation (OBBN), Logical context negation (OCNG), Statement Deletion (SSDL) and Integer Constant replacement (CRCR).

To reduce the subsumed mutants, we followed the guidelines of Tai *et al.* [50] and Kaminski *et al.* [27] and used a restrictive versions of the ROR and LCR operators. We also restricted the unary operators to replacement only.

5. RESULTS

5.1 Subsuming Mutants, Answering RQ1

The first research question regards the prevalence of subsuming mutants, i.e., mutants contributing to the testing process, in our subjects. Our results show that only a tiny proportion, ranging from 0.4% to 4.8% (GREP 1.7%, SED 3.2%, FLEX 4.8%, MAKE 0.4%, GZIP 2.5%), of mutants is actually needed for performing mutation analysis. It is noted that the above results correspond to the killed mutants (we removed the duplicated and live mutants from our analysis mutant sets) and that based on our test suites, killing these mutants guarantees, under the same test suite pool, the killing of all mutants of our analysis mutant sets.

Evidently the presence of so many subsumed mutants can potentially have a distorting effect in the accuracy of the mutation score. As a result, they skew the measurement and provide a misleading information regarding the level of test thoroughness. This is further investigated in our next research questions that are discussed in the following sections.

5.2 Inflation Effect, Answering RQ2

In our second research question we investigate the influence of the mutant inflation effect when using mutant sets of different sizes and measuring the effectiveness of the tests that kill them. We use all mutants and the subsuming mutants as effectiveness measurements. Figures 1(a) to 1(e) depict these results, for our subjects, when selecting mutants of size equal to the 0.5%, 1%, 2%, 4% and 8% respectively. Figure 1(f) depicts an example of all of our data, i.e., for mutant sets of sizes from 0.5% to 100%, to provide the whole picture of the two measurements. The upper line, gray boxes, represent the MS values while the lower line, white boxes, represent the MS^* values.

Our results demonstrate that killing a random 0.5% of all the mutants results in an effectiveness measure of at least 50%, according to MS, while in some cases, this measure is close to or above to 90%. When considering subsuming mutants, i.e., the MS^*, as an effectiveness measure, we observe that the effectiveness in the best case is 30%, while in most of the cases it is approximately 10%. the situation is similar when considering larger sets of mutants. Based on our results, with 8% of the mutants, we can have mutation scores of more than 90% (according to all mutants). However, in practice the great majority of these mutants will be subsumed and, as shown by our data, with respect to subsuming mutants, 8% of the mutants potentially can have effectiveness of at most 50%. Therefore, the inflation effect of the mutation score, using random mutants is important since the more mutants considered the more "blunt" becomes the test effectiveness measure.

Ideally, what we need is a metric that can distinguish among all the different mutant sets. Our results demonstrate that subsuming mutants form a good approximation to this ideal case since they fairly discriminate among the different mutants sets.

5.3 Correlation Analysis, Answering RQ3

Up to this point, our analysis has shown that subsumed mutants are prevalent. When mutation is used to guide the test generation process, the consequences of this issue, apart from the misleading measurement, does not seem to be important. However, in literature, mutation is used as a comparison basis between testing methods. In this specific case, the existence of so many non-contributing mutants can have a distorting effect in the effectiveness calculation. Thus, it is possible that one can receive an unfair advantage simply by killing these mutants. This is the subject of our third research question, where we investigate whether the subsumed mutants can have an influence on the measured comparisons.

We investigate this issue by computing the strength of the relation between mutation score and subsuming mutation score. A strong relation, i.e., a strong correlation, suggests that the two measurements have quite similar trend, i.e., to increase or decrease by a relatively similar amount. We also control for test suite size to investigate the impact of test suite effectiveness on the reported correlations. Table 2 records the results for the three correlation coefficients. Recall that all values are statistically significant. These results demonstrate that most of the correlations are fairly weak, between 0.2 - 0.6, with very few exceptions (for Gzip and large test suite sizes). Another interesting point is that all three coefficients agree among themselves. Thus, they provide confidence on the reported relations.

An important aspect of these results is the influence of test suite size on the correlations. This sometimes, in the cases of Grep and Sed, does not make a difference, while in the other cases, Flex, Make and Gzip, it does. Gzip seems to be an outlier, but this can be explained by the high mutation score achieved by the tests applied to Gzip, i.e., all the sets kill almost all the mutants. However, even in such excessive cases like this, many correlation coefficients have only medium level correlation.

Overall, our results suggest that mutation score (MS) is influenced by subsumed mutants. Hence, MS might not be a reliable indicator of the test effectiveness. Even in the cases that there is a strong correlation, there remain threats to commit subsequent error. We investigate this issue in the following sections.

5.4 Type I Error Analysis, Answering RQ4

A Type I error, also known as "false positive", refers to the case of rejecting, incorrectly, a true null hypothesis. In this research question, RQ4, we simulate a scenario where a researcher compares the quality of the test sets that were produced by two test techniques and uses mutation testing to identify the most effective technique. Our goal is to estimate the chances of getting a Type I error due to the presence of subsumed mutants. As explained before, we randomly selected groups of test suite sets, of equal size, having statistically significant difference according to traditional MS and measured the number of errors that this process would entail with respect to the MS^*.

Our results are depicted in Figure 2, which records, for every subject program (Figure 2(a) to 2(e)), the ratio of Type I errors committed when using different test suite sizes. We observe that, with the exception of Gzip, test size does not reduce the errors. This finding suggests that subsumed mutants influence the results that would be reported even when the examined sets of tests can kill most of the mutants under analysis.

The overall picture (Figure 2(f) records the average on all the programs and test sizes) is in line with the data reported for the previous research question; the chances to get a correct solution, according to both measures, is proportional to the correlation coefficients presented in RQ3. Finally, and perhaps more importantly, the chance of committing Type I error is high: approximately 62%, on average. This raises concerns regarding the validity of the conducted experiments that use mutation-based test assessment.

5.5 Sensitivity Analysis, Answering RQ5

In this section we go beyond statistical significance and perform a sensitivity analysis on the Type I errors with respect to increasing effect size. We thus try to identify the differences in mutation score, between the compared groups, that have a reduced chance of committing an error due to the increases in the effect size. To do so we constructed cases where we control for size and there is statistical significance difference between the studied groups while their average effect size, mutation score difference, is approximately 2%, 4%, 6%, 8% and 10%.

Sensitivity analysis results are presented in Figure 3. The plots present the level of agreement, i.e., chance to avoid making an error, between the two measurements, mutation score and subsuming mutation score, when the groups have a difference of 2%-10%. As expected, when the effect size

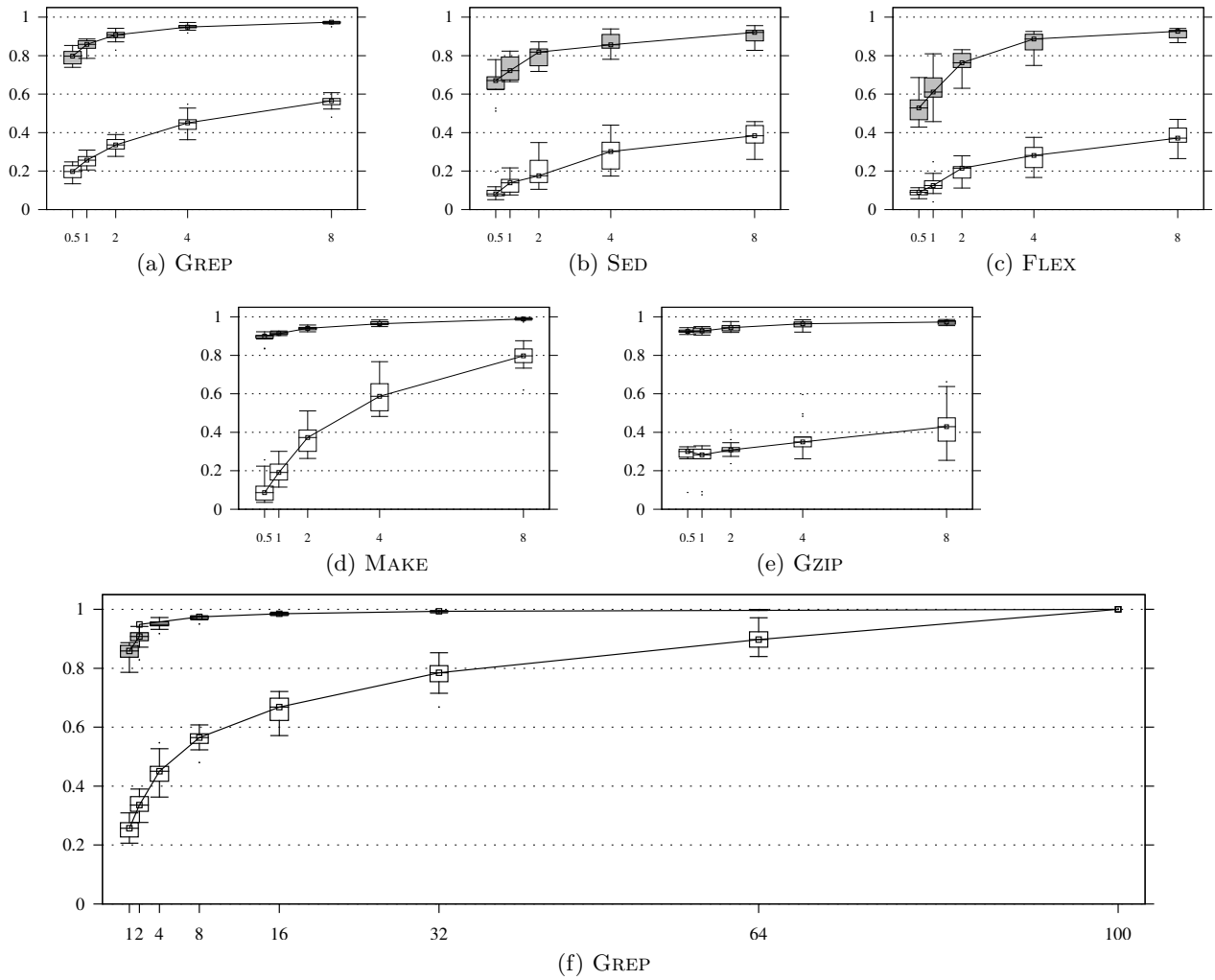

Figure 1: RQ2: Inflation effect, the MS is an undiscriminating measurement. Figures 1(a) to 1(e) present the curves and the distribution of the scores obtained when killing the 0.5%, 1%, 2%, 4% and 8% (x-axis) of randomly selected mutants. The y-axis in the upper (gray boxes) curve records the ratio of killed mutants over all mutants (MS), while the lower curve (white boxes) the ratio of subsuming mutants killed (MS^*). Figure 1(f) shows an example of the "whole picture", when considering (killing) up to 100% of the mutants.

increases the disagreement reduces. However, it is surprising that, even in the case where there is 10% difference, the chances for error remain high. For Make and Gzip, the chances appear lower for differences above 4%. Since we control for size, it was not possible to obtain differences between the two sets higher than 8% for Gzip and 10% for Make. These results suggest that even when having large effect size differences and controlling for test suite size, mutation-based test assessment is still prone to errors.

6. DISCUSSION

6.1 Research Impact

To estimate the impact of our findings on testing studies, we measured the extent to which recent research is vulnerable to the threats to validity that we have identified. Thus, we gathered all the papers from the last two years in the three leading software conferences, ISSTA, (ESEC)FSE and ICSE and counted the number of them that concern software testing, mutation testing and mutation-based test assessment. The last category is the one that is vulnerable to the validity threat we identified.

Our procedure was to gather all papers and keep only those having more than 6 pages. This process resulted in 389 papers. We then read and categorized these papers as presented in Table 3. Overall, 91 papers were concerned with software testing, of them 25 used mutation testing. Among the 25 papers that used mutation, 17 involved mutation-based test assessment none of which consider mutant subsumption. Thus, approximately 27% of the testing papers involve mutation testing and 68% of those papers are potentially vulnerable to the subsumed mutant threat.

6.2 Threats to Validity

Our study concerns Type I errors as a potential threat to validity in previous work. All these threats are construct validity threats, since they question the mutation score metric,

Table 2: RQ3: Correlation analysis; indicate that the proportion of all mutants killed (MS) is not generally strongly correlated with the proportion of subsuming mutants killed (MS^*). For each considered number of test cases and program, the mutation score (MS), Kendall (τ), Pearson (r) and Spearman (ρ) correlation coefficients between the mutation score (MS) and subsuming mutation score (MS^*) are presented.

	Grep				Sed				Flex				Make				Gzip			
No. Tests	MS	τ	r	ρ	MS	τ	r	ρ	MS	τ	r	ρ	MS	τ	r	ρ	MS	τ	r	ρ
3	0.705	0.4	0.52	0.54	0.732	0.42	0.5	0.54	0.641	0.24	0.33	0.32	0.798	0.51	0.58	0.63	0.962	0.49	0.29	0.54
6	0.784	0.34	0.48	0.47	0.794	0.35	0.45	0.46	0.769	0.29	0.38	0.39	0.852	0.63	0.61	0.76	0.973	0.52	0.48	0.59
9	0.824	0.22	0.35	0.31	0.821	0.36	0.46	0.47	0.822	0.33	0.42	0.44	0.880	0.71	0.57	0.84	0.979	0.54	0.51	0.61
12	0.847	0.24	0.37	0.34	0.836	0.37	0.47	0.49	0.862	0.35	0.44	0.46	0.891	0.73	0.52	0.86	0.983	0.58	0.54	0.65
15	0.864	0.19	0.29	0.27	0.851	0.36	0.44	0.47	0.882	0.39	0.46	0.50	0.899	0.76	0.54	0.89	0.987	0.63	0.56	0.71
18	0.875	0.17	0.25	0.24	0.860	0.37	0.46	0.49	0.899	0.46	0.51	0.59	0.906	0.78	0.56	0.91	0.988	0.66	0.53	0.73
21	0.885	0.21	0.29	0.29	0.869	0.37	0.46	0.49	0.910	0.44	0.46	0.55	0.910	0.79	0.63	0.92	0.991	0.73	0.55	0.79
24	0.892	0.22	0.32	0.31	0.875	0.38	0.46	0.50	0.920	0.47	0.45	0.57	0.913	0.79	0.74	0.92	0.992	0.74	0.52	0.80
27	0.899	0.20	0.27	0.28	0.885	0.40	0.49	0.52	0.925	0.56	0.52	0.68	0.916	0.80	0.75	0.92	0.993	0.78	0.55	0.83
30	0.905	0.24	0.34	0.34	0.890	0.40	0.48	0.52	0.931	0.55	0.47	0.66	0.919	0.80	0.93	0.92	0.995	0.83	0.57	0.88
33	0.909	0.25	0.36	0.35	0.896	0.43	0.52	0.55	0.937	0.58	0.47	0.68	0.921	0.81	0.93	0.93	0.995	0.89	0.60	0.92
36	0.913	0.24	0.35	0.34	0.900	0.40	0.50	0.52	0.941	0.57	0.42	0.68	0.923	0.81	0.94	0.93	0.996	0.91	0.61	0.94
39	0.000	0.26	0.37	0.36	0.918	0.39	0.49	0.50	0.944	0.62	0.48	0.73	0.904	0.81	0.93	0.93	0.926	0.92	0.63	0.95
42	0.920	0.28	0.39	0.38	0.908	0.46	0.58	0.59	0.949	0.64	0.46	0.73	0.928	0.81	0.93	0.93	0.997	0.92	0.61	0.95
45	0.924	0.26	0.37	0.36	0.913	0.41	0.52	0.54	0.951	0.68	0.47	0.78	0.929	0.82	0.94	0.93	0.997	0.96	0.65	0.98
48	0.927	0.28	0.40	0.39	0.915	0.43	0.53	0.55	0.953	0.70	0.52	0.80	0.931	0.81	0.94	0.93	0.997	0.96	0.70	0.98
51	0.928	0.28	0.40	0.39	0.920	0.41	0.51	0.52	0.956	0.69	0.47	0.79	0.934	0.82	0.94	0.94	0.998	0.98	0.71	0.99
54	0.931	0.33	0.43	0.45	0.921	0.42	0.51	0.54	0.956	0.72	0.53	0.83	0.936	0.83	0.95	0.94	0.998	0.98	0.79	0.99
57	0.935	0.29	0.41	0.40	0.923	0.46	0.56	0.58	0.958	0.69	0.54	0.79	0.937	0.83	0.94	0.94	0.998	0.99	0.75	0.99
60	0.936	0.30	0.40	0.41	0.927	0.42	0.53	0.53	0.960	0.75	0.54	0.85	0.939	0.83	0.95	0.94	0.998	0.99	0.77	1.00
63	0.938	0.29	0.39	0.40	0.930	0.42	0.52	0.53	0.961	0.74	0.57	0.84	0.941	0.83	0.95	0.94	0.998	0.99	0.93	1.00
66	0.939	0.28	0.37	0.39	0.932	0.44	0.54	0.56	0.963	0.76	0.55	0.86	0.943	0.83	0.95	0.94	0.999	0.99	0.79	1.00
69	0.943	0.31	0.41	0.42	0.937	0.41	0.51	0.53	0.964	0.77	0.58	0.86	0.945	0.83	0.95	0.94	0.998	0.99	0.87	1.00
72	0.943	0.33	0.45	0.46	0.937	0.47	0.57	0.59	0.963	0.77	0.61	0.86	0.946	0.82	0.94	0.94	0.999	1.00	0.82	1.00
75	0.945	0.33	0.44	0.45	0.938	0.43	0.53	0.54	0.965	0.77	0.66	0.86	0.947	0.83	0.95	0.94	0.999	1.00	0.87	1.00
78	0.947	0.34	0.45	0.47	0.940	0.41	0.50	0.52	0.966	0.77	0.64	0.86	0.949	0.83	0.95	0.94	0.999	1.00	0.84	1.00
81	0.948	0.35	0.44	0.47	0.944	0.44	0.53	0.56	0.967	0.74	0.60	0.83	0.950	0.84	0.95	0.95	0.999	1.00	0.98	1.00
84	0.949	0.31	0.41	0.43	0.945	0.46	0.55	0.58	0.967	0.79	0.68	0.88	0.952	0.83	0.95	0.94	0.999	1.00	0.91	1.00
87	0.952	0.35	0.45	0.47	0.946	0.45	0.53	0.56	0.968	0.77	0.70	0.87	0.953	0.84	0.95	0.95	0.999	1.00	1.00	1.00
90	0.953	0.34	0.43	0.46	0.949	0.45	0.53	0.57	0.969	0.76	0.70	0.85	0.955	0.84	0.95	0.95	0.999	1.00	0.91	1.00

but they are also internal validity threats since the confounding effects of subsumed mutants can determine our results (instead of all mutants). Namin and Kakarla [34] investigated threats regarding mutation operators, test suite size, and programming languages, which are all orthogonal to the threats we investigate here. Naturally, it is our duty to draw the reader's attention to the threats to the validity of our own findings.

6.2.1 Construct Validity

These threats regard the used measures, i.e., whether they are representative or not. Regarding mutation score and subsuming mutation score measures, we measure the test case ability to test thoroughly the code under assessment according to the mutation testing criterion. We demonstrate both the need and the consequences of not using these measure. This practice is also in line with previous research [28, 38, 39] and our results, ratios of subsuming mutants, are similar to those reported in their preliminary evaluations. Ideally, we would need to back our results with some "ground truth" data supporting the use of subsuming mutants. However, since the chances of committing Type I errors are high, a large variation between the two measurements is expected. This is a problem even in the case that the ground truth suggest against the subsuming mutants. Moreover, subsuming mutants form a measure free of redundant requirements, with respect to the used test suite.

Another threat might be due to the employed mutants, which could be non-representative of the "ideal" mutation testing criterion. We carefully choose mutants following the latest advances on mutation testing as detailed in Section 4.4. Additionally, these mutants were also used in the well-known mutation testing studies of Andrews *et al.* [5, 6].

6.2.2 Internal Validity

These are potential threats from our conclusions and our measurements. We used tests that are non-adequate and this might influence our results. However, non-adequate tests suites are widely used in practice [15] and the identification of equivalent and subsuming mutants is infeasible in real-world projects. Therefore, our results remain relevant to current mutation-based test assessment practices. Another issue is what is truly an "adequate test suite", future research will deepen our understanding on this respect.

Other issues might arise due to the tool we used. This might have errors that can affect our findings. We carefully checked our implementation and performed a manual evaluation of our results on some smaller programs. Also, Coccinelle is a tool widely used by both practitioners and researchers [1].

The random samples we studied involve a certain degree of stochastic data and consequently equivalently stochastic results. To reduce the influence of the random effects, we repeated our experiments a large number of times, 30 or 1,000 times each, and used statistical correlation techniques.

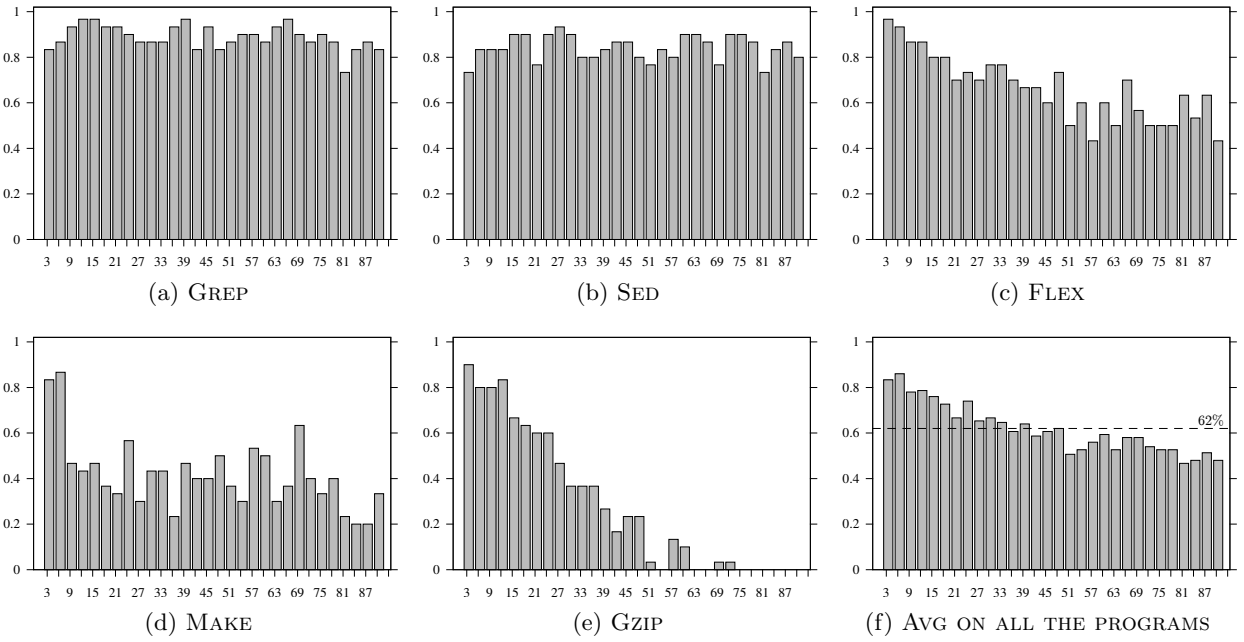

Figure 2: RQ4: Subsumed mutants in arbitrary sets of mutation-based test assessment experiments lead to Type I errors. For each subject program, and for all of them together, the bars record the chance of committing a Type I error per considered test suite size. The average on all the programs and sizes is 62%.

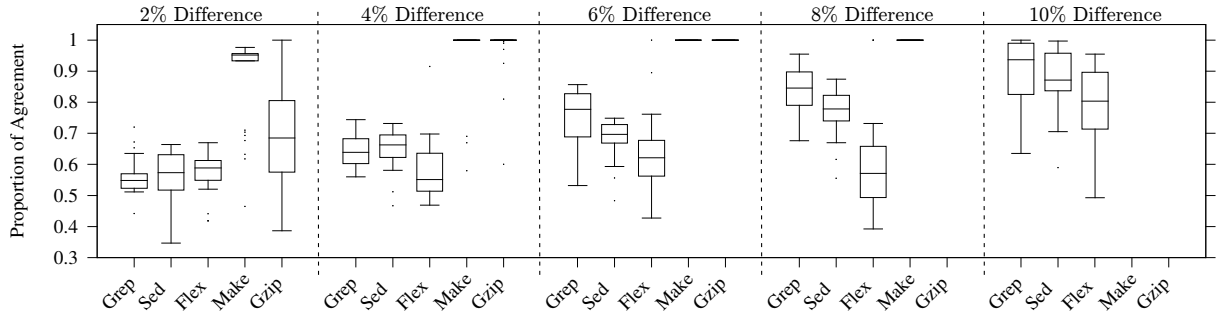

Figure 3: RQ5: Sensitivity analysis. Errors remain even as effect size increases. For each program, the boxplots represent the level of agreement, chance to avoid an error, between the mutation score (MS) and the subsuming mutation score (MS^*) when comparing test suites of the same size that have mutation score differences, on average, 2%, 4%, 6%, 8% and 10%. Since test size is controlled for, it was not possible to construct differences of more than 8% for Gzip and 10% for Make.

6.2.3 *External Validity*

These threats regard the generalizability of our results. We used 5 widely used in real-world projects. Still, there are many factors, e.g., test suites, programs and tools, that can affect our findings. Though, our goal is to show potential weaknesses of studies conducted under similar settings.

7. RELATED WORK

In literature two types of seeded faults appear; the machine-generated ones (mutants) and the manually seeded faults (introduced by humans based on their experience). Hutchins *et al.* [20] performed one of the first studies with manually seeded faults using the well-known Siemens suite. Although manually seeded faults are typically much fewer and hence less likely to have severe subsumption problems, the quality of these faults depends on the knowledge and expertise of the person that makes the seeding. For instance, the study of Hutchins *et al.* [20] only aimed at seeding faults that were revealed by a small fraction of tests and thus, completely ignored the subsumed mutant threat.

Mutation analysis was initially introduced as a test method helping the generation of test cases [10]. However, in recent years it has been proven to be quite powerful and capable of supporting various software engineering tasks [39]. In particular, mutants have been used to guide test generation [14, 16, 43], test oracle generation [14], to assist the debugging activities [19, 46], to evaluate fault detection ability [6, 24] and to support regression testing activities like test selection and prioritization [13, 48, 53]. The method has also been applied to behavioral models [2, 11], software product lines [17] and combination strategies [41].

Table 3: Subsumed mutant threats in the last two years of ISSTA, (ESEC)FSE, and ICSE testing studies. The percentages represent the ratios of the testing studies that use mutation and those on mutation vulnerable to the subsumed mutant threat.

Conf.	Year	Testing	Mutation Testing		Threatened	
ISSTA	'14	17	3	18%	1	33%
	'15	17	3	18%	3	100%
(ESEC)	'14	11	3	27%	2	67%
FSE	'15	14	5	36%	4	80%
ICSE	'14	12	6	50%	5	83%
	'15	20	5	25%	2	40%
Total		91	25	27%	17	68%

Equivalent mutants form one of the main problems of mutation testing [18, 31, 47]. Despite the efforts made by the community, e.g., [23], and recent advances [7, 29, 42, 47], the problem is remains [42]. A similar situation arises when considering mutation-based test generation [14, 16, 43].

Mutation is popular [23] thanks to mutants' ability to simulate the behaviour of real faults [6, 24]. Also, many modern mutation tools are integrated with build systems and development tools, thus making their application easy [9]. Previous research has also suggested that mutation testing has the ability to probably subsume and to reveal more faults than most of the other white-box testing criteria [37, 39] and hence potentially provide testers with substantial benefits.

From its incipient, it was evident that the number of mutants was a practical problem of mutation testing. Therefore, researchers tried to identify smaller yet representative subsets. An initial reduction was made based on the coupling effect hypothesis, which states that test cases revealing simple mutants can also reveal complex ones [10, 35].

A simple but effective way to reduce the number of mutants is random sampling [44]. Although, sampling can provide a range of trade-offs, Papadakis and Malevris [44] provided evidence that mutant sampling ratios of 10% to 60% have a loss on fault detection from 6% to 26%. Similar results have been shown in the study of Wong *et al.* [51].

There are other mutant reduction strategies that fall in the category of selective mutation [36]. Selective mutation tries to reduce the arbitrariness of random sampling by using only specific types of mutants. However, Zhang *et al.* [52] showed that in practice random mutant sampling performs comparably well with the selective mutation.

The studies of Tai [49, 50] focused on reducing the number of the mutants with respect to two fault-based testing strategies. This was achieved by constraining and restricting the mutants produced by the relational and logical mutation operators to reduce the number of subsumed mutants. Thus, their suggestion was to restrict the mutant instances of the relational and logical operators. In later studies, Kaminski *et al.* [26, 27] studied the relational mutation operator and came to a similar conclusion. Hence, they proposed reducing the number of the subsumed mutants by restricting the mutant instances of the relational operator.

Using minimized constraint mutant instances to efficiently generate mutation-based test cases has been proposed by Papadakis and Malevris [45]. Thus, when aiming at generating test cases, there is no point of targeting subsumed mutants instead of the subsuming ones. On the same lines, Just *et al.* [25] demonstrated that by constraining the relational and logical operators, it is possible to reduce the number of subsumed mutants.

The first study that used subsuming mutants to compare testing techniques, is that of Kintis *et al.* [28]. Kintis *et al.* introduced the notion of disjoint mutants and demonstrated that the majority of mutants produced by the MuJava tool are jointly killed, i.e., they are subsumed. The same study also shown that hard-to-kill mutants are prone to the mutant subsumption problem. Amman *et al.* [3] defined and analysed the minimal mutant sets. Their results agree with that of Kintis *et al.* and demonstrate that the majority of mutants used by the MuJava and Proteum mutation testing tools are subsumed. Later, Kurtz *et al.* [30] used static analysis techniques, such as symbolic execution to identify subsuming mutants.

More recently, Papadakis *et al.* [42] used compilers to eliminate trivially duplicated mutants, i.e., a special form of subsumed mutants. Recall, that these mutants are mutually equivalent but not with the original program. In the same study, it is reported that 21% of all mutants is duplicated and can be removed by using the compiler optimization techniques of the GCC compiler.

All the approaches discussed in this section aimed at reducing the application cost of mutation testing. Those that considered mutant subsumption either identified the fact that most of the mutants are trivial/subsumed or used some form of redundancy elimination to reduce cost. Therefore, none of them studied the distorting effects of including subsumed mutants, nor the use of subsuming mutants as a way to strengthening experiments, i.e., by reducing the chances of committing Type I errors.

8. CONCLUSION

This paper presents experimental evidence for the threat to validity occasioned by the inclusion of subsumed mutants in mutation-based test assessment studies. This evidence highlights the need to use mutation-based assessment metrics based solely on subsuming mutants. We are not the first authors to advocate the use of such a mutation assessment metric, but we are the first to investigate the scientific cost of ignoring this advice. We show that a large number of studies are vulnerable to this threat to validity. Based on our findings we recommend that all such future studies should, first, identify and discard as many subsumed mutants as possible.

In future, we plan to investigate whether mutant subsumption can also lead to type II errors and whether the choice of mutant operators can affect our results, (here we only consider random mutant selection). We also plan to validate further our findings using more and real data. Finally, although our results show that an arbitrary experiment has a high chance of committing a type I error, this might vary for particular experiments. Therefore, we plan to replicate previous research to address the subsumed mutant threat.

9. ACKNOWLEDGMENTS

Mike Papadakis is supported by the National Research Fund, Luxembourg, INTER/MOBILITY/14/7562175 and by Microsoft Azure Grant 2015. Mark Harman and Yue Jia are supported by EPSRC grant Dynamic Adaptive Automated Software Engineering (DAASE: EP/J017515).

10. REFERENCES

[1] Coccinelle: A program matching and transformation tool for systems code. http://coccinelle.lip6.fr/papers.php.

[2] B. K. Aichernig, J. Auer, E. Jöbstl, R. Korosec, W. Krenn, R. Schlick, and B. V. Schmidt. Model-based mutation testing of an industrial measurement device. In *Tests and Proofs - 8th International Conference TAP*, pages 1–19, 2014.

[3] P. Ammann, M. E. Delamaro, and J. Offutt. Establishing theoretical minimal sets of mutants. In *IEEE International Conference on Software Testing, Verification and Validation, ICST*, pages 21–30, 2014.

[4] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[5] J. H. Andrews, L. C. Briand, and Y. Labiche. Is Mutation an Appropriate Tool for Testing Experiments? In *ICSE*, pages 402 – 411, 2005.

[6] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. *IEEE Trans. Softw. Eng.*, 32(8):608–624, 2006.

[7] S. Bardin, M. Delahaye, R. David, N. Kosmatov, M. Papadakis, Y. L. Traon, and J. Marion. Sound and quasi-complete detection of infeasible test requirements. In *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015*, pages 1–10, 2015.

[8] T. A. Budd and D. Angluin. Two Notions of Correctness and Their Relation to Testing. *Acta Informatica*, 18(1):31–45, 1982.

[9] M. Delahaye and L. du Bousquet. Selecting a software engineering tool: lessons learnt from mutation analysis. *Softw., Pract. Exper.*, 45(7):875–891, 2015.

[10] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, Apr. 1978.

[11] X. Devroey, G. Perrouin, M. Papadakis, P.-Y. Schobbens, and P. Heymans. Featured Model-based Mutation Analysis. In *International Conference on Software Engineering, ICSE*, Austin, TX, USA, 2016.

[12] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[13] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Trans. Software Eng.*, 32(9):733–752, 2006.

[14] G. Fraser and A. Zeller. Mutation-driven generation of unit tests and oracles. *IEEE Trans. Software Eng.*, 38(2):278–292, 2012.

[15] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov. Guidelines for coverage-based comparisons of non-adequate test suites. *ACM Trans. Softw. Eng. Methodol.*, 24(4):22, 2015.

[16] M. Harman, Y. Jia, and W. B. Langdon. Strong higher order mutation-based test data generation. In *19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference*, pages 212–222, 2011.

[17] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In *IEEE International Conference on Software Testing, Verification and Validation, ICST Workshops Proceedings*, pages 188–197, 2013.

[18] R. M. Hierons, M. Harman, and S. Danicic. Using program slicing to assist in the detection of equivalent mutants. *Softw. Test., Verif. Reliab.*, 9(4):233–262, 1999.

[19] S. Hong, B. Lee, T. Kwak, Y. Jeon, B. Ko, Y. Kim, and M. Kim. Mutation-based fault localization for real-world multilingual programs (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 464–475, 2015.

[20] M. Hutchins, H. Foster, T. Goradia, and T. J. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering*, pages 191–200, 1994.

[21] L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 435–445, 2014.

[22] Y. Jia and M. Harman. Higher Order Mutation Testing. *Journal of Information and Software Technology*, 51(10):1379–1393, October 2009.

[23] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37(5):649 –678, sept.-oct. 2011.

[24] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 654–665, 2014.

[25] R. Just, G. M. Kapfhammer, and F. Schweiggert. Do redundant mutants affect the effectiveness and efficiency of mutation analysis? In *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012*, pages 720–725, 2012.

[26] G. Kaminski, P. Ammann, and J. Offutt. Better predicate testing. In *Proceedings of the 6th International Workshop on Automation of Software Test, AST*, pages 57–63, 2011.

[27] G. Kaminski, P. Ammann, and J. Offutt. Improving logic-based testing. *Journal of Systems and Software*, 86(8):2002–2012, 2013.

[28] M. Kintis, M. Papadakis, and N. Malevris. Evaluating mutation testing alternatives: A collateral experiment. In *APSEC*, pages 300–309, 2010.

[29] M. Kintis, M. Papadakis, and N. Malevris. Employing second-order mutation for isolating first-order equivalent mutants. *Softw. Test., Verif. Reliab.*, 25(5-7):508–535, 2015.

[30] B. Kurtz, P. Ammann, and J. Offutt. Static analysis of mutant subsumption. In *Eighth IEEE International Conference on Software Testing, Verification and*

Validation, ICST 2015 Workshops, Graz, Austria, April 13-17, 2015, pages 1–10, 2015.

[31] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Jozala. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Trans. Software Eng.*, 40(1):23–42, 2014.

[32] A. S. Namin and J. H. Andrews. The influence of size and coverage on test suite effectiveness. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, pages 57–68, 2009.

[33] A. S. Namin, J. H. Andrews, and D. J. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 351–360, 2008.

[34] A. S. Namin and S. Kakarla. The use of mutation in testing experiments and its sensitivity to external threats. In *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA*, pages 342–352, 2011.

[35] A. J. Offutt. The Coupling Effect: Fact or Fiction. *ACM SIGSOFT Software Engineering Notes*, 14(8):131–140, December 1989.

[36] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An Experimental Determination of Sufficient Mutant Operators. *ACM T. Softw. Eng. Meth.*, 5(2):99–118, April 1996.

[37] A. J. Offutt, J. Pan, K. Tewary, and T. Zhang. An Experimental Evaluation of Data Flow and Mutation Testing. *Software Pract. Exper.*, 26(2):165–176, 1996.

[38] A. J. Offutt and J. M. Voas. Subsumption of condition coverage techniques by mutation testing. 1996.

[39] J. Offutt. A mutation carol: Past, present and future. *Information & Software Technology*, 53(10):1098–1107, 2011.

[40] Y. Padioleau, J. L. Lawall, R. R. Hansen, and G. Muller. Documenting and automating collateral evolutions in linux device drivers. In *Proceedings of the 2008 EuroSys Conference, Glasgow, Scotland, UK, April 1-4, 2008*, pages 247–260, 2008.

[41] M. Papadakis, C. Henard, and Y. L. Traon. Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing. In *IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, Cleveland, Ohio, USA*, pages 1–10, 2014.

[42] M. Papadakis, Y. Jia, M. Harman, and Y. L. Traon. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 936–946, 2015.

[43] M. Papadakis and N. Malevris. Automatic mutation test case generation via dynamic symbolic execution. In *IEEE 21st International Symposium on Software Reliability Engineering, ISSRE 2010, San Jose, CA, USA, 1-4 November 2010*, pages 121–130, 2010.

[44] M. Papadakis and N. Malevris. An empirical evaluation of the first and second order mutation testing strategies. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010, Workshops Proceedings*, pages 90–99, 2010.

[45] M. Papadakis and N. Malevris. Mutation based test case generation via a path selection strategy. *Information & Software Technology*, 54(9):915–932, 2012.

[46] M. Papadakis and Y. L. Traon. Metallaxis-fl: mutation-based fault localization. *Softw. Test., Verif. Reliab.*, 25(5-7):605–628, 2015.

[47] D. Schuler, V. Dallmeier, and A. Zeller. Efficient mutation testing by checking invariant violations. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, pages 69–80, 2009.

[48] A. Shi, A. Gyori, M. Gligoric, A. Zaytsev, and D. Marinov. Balancing trade-offs in test-suite reduction. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 246–256, 2014.

[49] K. Tai. Predicate-based test generation for computer programs. In *Proceedings of the 15th International Conference on Software Engineering, Baltimore, Maryland, USA, May 17-21, 1993.*, pages 267–276, 1993.

[50] K.-C. Tai. Theory of Fault-based Predicate Testing for Computer Programs. *IEEE Transactions on Software Engineering*, 22(8):552–562, August 1996.

[51] W. E. Wong and A. P. Mathur. Reducing the Cost of Mutation Testing: An Empirical Study. *J. Syst. Software*, 31(3):185–196, December 1995.

[52] L. Zhang, S. Hou, J. Hu, T. Xie, and H. Mei. Is operator-based mutant selection superior to random mutant selection? In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 435–444, 2010.

[53] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. Regression mutation testing. In *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, pages 331–341, 2012.

[54] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.