Angels and Monsters: An Empirical Investigation of Potential Test Effectiveness and Efficiency Improvement from Strongly Subsuming Higher Order Mutation

Mark Harman and Yue Jia
CREST, University College London, UK

Pedro Reales Mateo and Macario Polo
Univ. of Castilla-La Macha, Spain

ABSTRACT

We study the simultaneous test effectiveness and efficiency improvement achievable by Strongly Subsuming Higher Order Mutants (SSHOMs), constructed from 15,792 first order mutants in four Java programs. Using SSHOMs in place of the first order mutants they subsume yielded a 35%-45% reduction in the number of mutants required, while simultaneously improving test efficiency by 15% and effectiveness by between 5.6% and 12%. Trivial first order faults often combine to form exceptionally non-trivial higher order faults; apparently innocuous angels can combine to breed monsters. Nevertheless, these same monsters can be recruited to improve automated test effectiveness and efficiency.

1. INTRODUCTION

Mutation testing seeds faults into the program under test to create 'mutants' [7, 24]. These mutants can be classified as being either First Order Mutants (FOMs) or Higher Order Mutants (HOMs). FOMs seed only a single fault, while HOMs are a generalizaion in which at least one fault is seeded. The space of higher order mutants is exponential, but search techniques have proved able to find many interesting HOMs, thereby avoiding the need for (an infeasible) full enumeration of the HOM space [21, 23].

There is empirical evidence that HOMs may also reduce the proportion of mutants that are equivalent to the original program from which they are constructed [27, 41]. It has also been shown that HOMs have the potential to reduce test effort [21, 23, 34, 42]. Higher order mutation has been applied to model based testing [4] and concurrency testing [32] as well as code-level mutation (as studied in this paper). A more detailed introduction to the higher order mutation paradigm can be found elsewhere [15].

Strongly Subsuming Higher Order Mutants (SSHOMs) are a particularly important subclass of HOMs that are only killed by a subset of the *intersection* of test cases that kill each of the first order mutants from which they are constructed [23].

By construction, an SSHOM is at least as good (and possibly superior to) any of its constituent FOMs, in the sense that a test case that kills it will be guaranteed to kill all FOMs (but not vice versa, whence the moniker 'strongly subsuming').

Although SSHOMs are theoretically at least as good as the FOMs they subsume, there has been no previous empirical investigation of this effect. Therefore, we have no data in the literature that reports the expected reduction in test effort than can be achieved (if any). Furthermore, though the use of SSHOMs cannot, by construction, reduce test effectiveness, we have no data on whether any improvement in test effectiveness can be achieved.

Most test optimization techniques (for example regression test selection and minimization [44, 50] and selective mutation [24, 38]) involve an inherent trade-off between effectiveness and efficiency. Test improvement approaches either make testing more effective at the expense of some (hopefully acceptable) reduction in its efficiency or they make testing more efficient with a concomitant reduction in effectiveness. The use of SSHOMs in place of FOMs offers a more attractive possibility: maybe we can simultaneously improve both effectiveness and efficiency. Adoption of the approach will therefore not require any trade offs between the competing objectives of efficiency and effectiveness.

More precisely, we distinguish two distinct reasons why an SSHOM study may be of interest to mutation testers, depending upon whether or not the test process has a fully automated oracle available:

Fully Automated: The entire mutation testing process is automated, including checking whether a test output is correct (the so-called 'oracle' problem [9, 17, 47]).

Partially Automated: Mutant generation and execution are automated, but the determination of whether the output is correct is not fully automated. There is no fully automated test oracle and so some of the burden of the oracle problem falls on the shoulders of the test engineer or other humans involved in the test process.

In the first of these two scenarios, full automation means that the number of mutants required will have a strong impact on the overall testing time. Therefore, in this fully automated scenario, both reductions in the numbers of mutants required *and* reductions in the number of test cases required will be of interest. In the second of the two scenarios, the test effort will be likely dominated by the human effort required to provide an oracle, so reductions the number of test cases to be examined will outweigh the value of any reductions in the numbers of mutants required.

In this paper we study the reduction in both the number of mutants and the number of test cases that result from the replacement of FOMs with the SSHOMs that subsume them. All current techniques for finding HOMs (including that used in this paper to find SSHOMs) require all FOMs to be executed as part of the HOM construction process. Therefore, any reduction in the number of mutants will only be of interest if we can find ways of statically predicting the FOMs that are more likely to combine to form SSHOMs. However, in the second scenario, where test effort is dominated by the oracle cost, any reduction in the number of tests required (while maintaining or improving test effectiveness) will be valuable to the tester (even using currently available techniques to construct HOMs).

We asses the efficiency and effectiveness of SSHOM-based testing, but we also want to understand the properties of subsumed FOMs: if subsumed FOMs are very hard to kill (they are 'monsters'), then we might expect SSHOMs constructed from them to also be monsters; *from monsters shall we expect to construct monsters*. By contrast, if we find that subsumed FOMs are easy to kill on average (they are 'angels'), then the SSHOM construction process transforms angels into monsters: it builds (hard-to-kill) monster faults from (easy-to-kill) angelic faults. Our findings concern the effects of using the set of Strongly Subsuming Higher Order Mutants (which we denote SSHOMs) in place of the set of first order mutants they subsume (which we denote FOMs). The primary contributions are:

Mutant Minimization: SSHOMs reduce the number of mutants to be considered by 35%. This rises to 45% if SSHOMs is minimized using greedy test suite minimization to produce a reduced set, SSHOMsRed.
Efficiency improvement: SSHOMs reduce test effort by 14% (15% for SSHOMsRed).
Effectiveness improvement: SSHOMs increase test effectiveness by 5.6% to 12.0% (similarly for SSHOMsRed).
Monsters from angels: SSHOMs are 'monster' faults that are exceptionally hard to detect, yet are constructed entirely from 'angel' faults that are easy to detect.
Unsubsumed coverage: SSHOMs subsume 45% - 75% of all FOMs. We therefore need to find techniques that increase SSHOM coverage (or to use hybrid FOM/SSHOM testing).

2. SEARCHING FOR SSHOMS

A Strongly Subsuming Higher Order Mutant (SSHOM) is defined as follows. Let Kill(S) return the set of test cases which kill all mutants in the set S. The higher order mutant H, constructed from a set of first order mutants $F_1, ..., F_n$, is an SSHOM[21] iff

$$Kill(H) \subseteq \bigcap_i Kill(F_i) \ \ and \ \ Kill(H) \neq \emptyset$$

Figure 1 depicts this definition of SSHOM in a Venn diagram. The test set denoted T_m kills the higher order mutant, H, while the test sets denoted T_i kill first order mutants F_i. If H is an SSHOM then T_m must be a non-empty subset of the intersection of T_i.

Our four subjects are Monopoly [33] and Chess [34] (two board games), and Validator and Clifrom the Apache project. They range in size from 1,190 Lines of Code (LoC) to 6,359 LoC (see Table 2). We use the Java mutation testing tool, Bacterio [33] and its set of Java mutants.

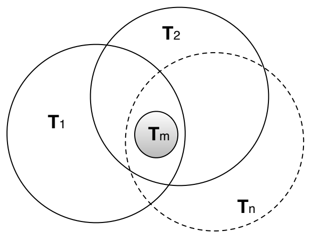

Figure 1: Generalized n^{th} order SSHOM scenario

Table 1: Quantitative information about Subject Systems

Subject System	#Tests	Statement coverage	First order mutation score
Monopoly	213	98.8%	83.8%
Chess	593	99.6%	88.6%
Validator	415	80.0%	73.6%
Cli	187	95.7%	77.6%

All 4 applications come with good quality test suites distributed with the applications and designed by third party testers (not the experimenters). We use these tests in our empirical study, because they are considered to be complete and adequate by their developers. Quantitative information concerning the test suites are summarised in Table 1. In the rest of this study we focus on the mutants that are killed by the test suites. Manual inspection of samples of the remaining unkilled mutants indicated that many are equivalent. In order to ensure high quality test data, we manually added test cases to the test suites provided, where we found unkilled yet non-equivalent mutants. However, as with all studies of equivalent mutants, we cannot be sure that we have detected all equivalent mutants, even if we were to manually inspect every one (which is seldom practical [19]).

Finding SSHOMs among the space of all higher order mutants requires a computational search process, such as those associated with the widely-studied area known as 'Search Based Software Engineering' (SBSE) [10, 16]. This is because the space of higher order mutants is exponentially large in n for n^{th} order mutants. We use a simple genetic algorithm to search for SSHOMs among the HOMs constructed within each of the methods of the program. It is recommended [18] that authors of SBSE work define clearly the representation and fitness function used in the search process. We define these two key 'SBSE ingredients' in the remainder of this section, along with the search operators we use and our termination procedure.
Representation: We represent a higher order mutant as a vector of pairs of integers. The length of the vector is the number of first order mutants that combine to form the HOM represented. Each vector element indexes the location of a first order mutant and the type of mutation to be applied. A single point crossover is applied to generate 'offspring'. It first chooses a cut point. Two higher order mutant 'parents' are then split at the cut point and the resulting four fragments are recombined to create two new 'children' in the standard manner for single point crossover [12]. Figure 2 illustrates this process.

Crossover can produce mutants that are not higher order, either because of repetition of a lower order mutants or because of repeated mutation at the same location. We avoid repetition of lower order mutants, but make no attempt to control for same-location mutations.

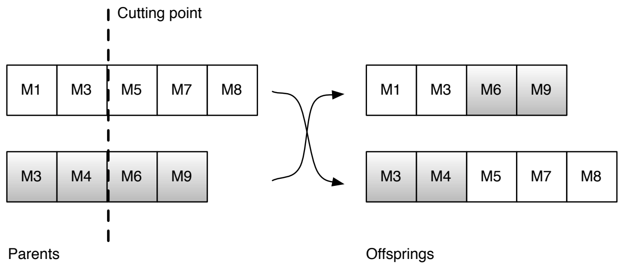

Figure 2: Crossover operator used in our search

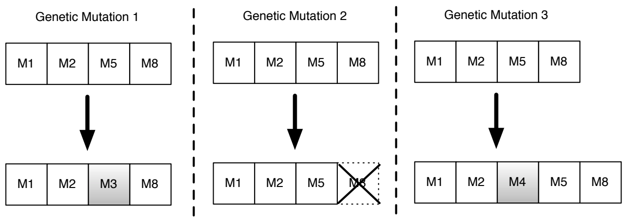

Figure 3: Genetic mutation operators used in our search

Such same-location mutants will not have increased fitness and, therefore, the genetic algorithm will not favour their selection. The genetic algorithm uses three genetic mutation operators, illustarted in Figure 3. The first operator randomly selects a FOM and replaces it with another FOM. The second operator randomly removes a selected FOM and the third operator adds a new randomly selected FOM to the current HOM.

Fitness function: Our fitness function simply seeks to reduce the number of tests that kill all FOMs, while increasing the number that kill the HOM: the fitness of an SSHOM, $H_{\{F_1,\ldots,F_n\}}$, with respect to a test suite T is defined as $|T| - |\text{Kill}(H_{\{F_1,\ldots,F_n\}})| + |\bigcap_{i=1}^{n} \text{Kill}(F_i)|$.

This function targets subsuming HOMs, but there is nothing in the computation that specifically favours *strongly* subsuming HOMs. In so doing, we are thus sampling (randomly) from the space of subsuming higher order mutants to find a set that is strongly subsuming. Because we do not aggressively seek strong subsumption, our results can be thought of as a lower bound on technical improvements that can be achieved using strongly subsuming HOMs. Our goal is not to introduce a new mutation testing *technique*. Rather, we seek to identify a set of SSHOMs so that we can empirically *study* them, thereby providing results against which future work on SSHOMs can compare.

Future work may focuses on carefully designed fitness functions that target strongly subsuming higher-order mutants with particular behaviour and properties. Indeed, such work is on-going. For example, recent work [40] has introduced a fitness function that targets strongly subsuming HOMs that test untested behaviour (that which is not tested by any of the constituent FOMs).

We used a 'reset' feature as a termination condition and also to avoid the search becoming unnecessarily trapped in local optima. Reset simply restarts the search with a new randomly generated population when the variance in fitness observed has fallen below 10% of its starting variance. The overall search terminates when no new SSHOMs are found after three restarts have been applied.

3. EMPIRICAL STUDY PROCEDURE

This section explains our research questions, motivating why we ask them, how they relate to one another and the experimental and inferential statistical procedures used to answer them.

We search for a set of SSHOMs for each of the four programs using the generic algorithms described in the previous section (Section 2). From the set SSHOMs, we construct a set SSHOMsRed using greedy minimization [43, 50]. This algorithm iteratively removes mutants from SSHOM, while there remain mutants that can be removed without affecting the set of FOMs subsumed.

We start by establishing a baseline set of data on the numbers of mutants and test cases involved in our study. Such baseline data can be useful, for example, in replication studies that involve different subjects and for which the sets of mutants and test cases may therefore differ. In particular, we are interested to establish baseline data for the sizes of the SSHOM and SSHOMsRed sets, since these are comparatively less well studied than first order mutants (the FOM set).

RQ1: Baseline Data: We decompose RQ1 into two sub research questions, one concerning baseline data on mutants and the other concerning baseline data on test suites.

RQ1.1: Baseline Data on Mutants: What are the numbers of FOMs, SSHOMs SSHOMsRed and the percentage of FOMs subsumed by the SSHOMs?

Once the SSHOMs were found, mutation adequate tests suites were constructed for each of the three sets of mutants: FOMs, SSHOMs and SSHOMsRed. The test suite construction process consisted of random selection from the pool of test cases provided with each of the four programs under study. By using tests from a third party, we remove one source of potential bias (selection of test cases by the experimenters). We also hope that the use of real-world test suites may make it more likely that our results will be realistic and, thereby more relevant to real-world testers.

To select, from this pool, a test suite to kill mutants in the set S, test cases were repeatedly randomly chosen until all mutants in S were killed. This process was repeated 30 times to cater for the stochastic nature of the selection process.

RQ1.2: Baseline Data on Test Suites: What are the average sizes of mutation adequate test suites (and the standard deviation) over the thirty tests suites selected to kill all mutants in the three classes of mutant for each of the four programs?

RQ1.1 and RQ1.2 also allow us to assess the degree of test effort reduction (in terms of the numbers of mutants and test cases required) that can be achieved using SSHOMs in place of the FOMs that they subsume. Having established baseline data for mutants and test cases, we move to the two research questions at the heart of our technical investigation.

That is, how much harder to kill are strongly subsuming higher order mutants (RQ2) and how much more effective are the test suites constructed from them (RQ3)?

RQ2: Higher order potency: How much harder to kill are mutants in SSHOM (and SSHOMsRed) than those in FOM?

Note that SSHOMs are theoretically at least as hard to kill as the FOMs from which they are constructed, by definition. Therefore, we need not ask whether SSHOMs *are* harder to kill; we know they are at least as hard to kill.

In our study, we seek to measure the significance (or otherwise) and size of this effect. In order visualize the difference in killability of FOMs and SSHOMs (and SSHOMsRed) we draw a graph we term the 'kill-growth' graph. We consider two types of kill growth graph. A Proportional Kill Growth (PKG) graph and an Absolute Kill Growth (AKG) graph. For a given test suite order TO and set of mutants M, the kill growth graph shows the growth in the absolute number (AKG) or proportion (PKG) of mutants from M killed by TO.

A point in the PKG graph at (x, y) signifies that after $x\%$ of the tests in TO have been executed, the proportion of mutants from M killed is y. A point in the AKG graph at (x, y) signifies that after $x\%$ of the tests in TO have been executed, the number of mutants from M killed is y. It is reasonable to use a PKG graph when the mutant sets are the same size. However, for differently sized sets of mutants, the AKG graph should also be reported, because it reflects the growth in the numbers of faults found.

We have 3 sets of 30 test suites (one set for each of the three classes of mutant). We combine all these sets of test suites into a single set of 90 test suites for this visualization. In so-doing we seek to visualise the typical expected average kill growth trends for each of the three classes of mutants, averaged over 90 trials. We have no information about the specific order in which a tester might execute test cases. Therefore, we report the average mutation score observed over all 90 test suites in the (arbitrary) order in which they are selected by the random test selection algorithm.

To assess the statistical significance of the difference between first order mutants and higher order mutants, we used a Wilcoxon Signed-Rank test with the widely-adopted 'standard' 95% significance level (α= 0.05). This α level is reduced to 0.0167, using the Bonferroni correction to cater for the fact that we perform three Wilcoxon tests ($0.0167 = 0.05/3$). The Bonferroni correction is known to be a conservative correction. It risks Type II errors (incorrectly accepting the null hypothesis) in order to reduce the risk of Type I errors (incorrectly rejecting the null hypothesis). Therefore, should we find that our results force us to reject the null hypothesis after this correction (i.e., there is no possibility of Type II error), then we can be confident that our results are strongly significant.

Knowing how much harder SSHOMs are to kill (compared to FOMs) is interesting to mutation testing researchers, but for testing practitioners the important question is

"How much more effective is SSHOM testing?"

This question is addressed by RQ3:
RQ3: Higher order test effectiveness: How much more effective are test suites constructed from SSHOMs compared to those constructed from FOMs?

We plot PKG graphs for each of the three types of test suite: those selected to kill each of the three types of mutants (FOMs, SSHOMs and SSHOMsRed). Since each test suite is selected to kill a particular class of mutants, S, there is a natural order in which to execute the test suites (unlike the union of all test cases used in answering RQ2): we order the test cases in each test suite using the so-called 'additional greedy' algorithm [51]. This chooses the next test to be executed as that which kills the greatest number of remaining unkilled mutants in the set S.

This greedy ordering ensures that we consider test cases in the order that respects the 'design' of the test suite; tests selected to kill FOMs, for example, are ordered so that they favour early achievement of 100% FOM coverage. In total, we have selected 30 different test suites for each of the three classes of mutants (FOMs, SSHOMs and SSHOMsRed), so we now have a total of 90 ordered test suites with which to experiment.

Since we have unpaired data, we use the Kruskal-Wallis test to analyse the variance of the three sets of data obtained from the test suites selected to kill each of the three kinds of mutants. The Kruskal-Wallis test is an unpaired and non-parametric test that determines whether a set of samples originate from the same distribution according to the variance of their ranks. We augment the Kruskal-Wallis test with a post-hoc pairwise comparison of the different classes of test (selected for FOMs, SSHOMs and SSHOMsRed) using Dunn's test [45].

We use Dunn's test because we have not only unpaired samples from populations of unknown distribution (therefore suggesting the use of a non-parametric test), but also samples of uneven sizes, which means that Wilcoxon's unpaired non-parametric test is not directly applicable). Dunn's test also includes a generalised version of the Bonferroni correction for multiple tests. Therefore, it the P values we report are adjusted to take account of the number of statistical significance tests we perform.

Having explored whether SSHOMs can improve testing by replacing subsumed FOMs, we wish to study whether there is anything more interesting about subsumed FOMs compared to the unsubsumed FOMs:
RQ4: Characteristics of Subsumed FOMs: What are the differences between subsumed and unsubsumed FOMs?

We decompose RQ2 into three subquestions as follows:
RQ4.1: Subsumed FOM killability: What is the killability of subsumed FOMs compared to the remaining FOMs?

To answer RQ4.1 we compare the two box plots constructed from the set of killability values (proportion of all test cases that kill a mutant) for subsumed and unsubsumed FOMs. However, a mutant may fail to be killed by a test case because the test case fails to cover the mutant. Mutants that reside in branches that are seldom executed will thus naturally tend to appear 'harder to kill' simply because they are harder to reach (it will be harder to execute the mutated code). Such reduced killability due to reduced executability is a property of the program under test, rather than any characteristic of the mutants that reside in it.

Therefore, in order to provide an assessment of mutant killability that takes account of reachability, we complement the answer to RQ4.1 with a repeated experiment in which only test cases that execute the mutant (covering test cases) are counted in determining a mutant's killability:
RQ4.2: Subsumed FOM killability by Covering Tests: What is the killability of subsumed FOMs compared to the remaining FOMs when restricted to covering tests?

To answer RQ4.2 we compare the two box plots constructed from the set of killability values for subsumed and unsubsumed FOMs using only test cases that cover the mutants. Finally, we examine the distribution of subsumed FOMs over all FOMs:
RQ4.3: Subsumed FOM distribution: How do the subsumed FOMs distribute over all mutants in the process of mutation testing?

Table 2: Number of SSHOMs found

Subject	LoC	FOMs	SS HOMs	SSHOMs Red	Subsumed FOMs
Monopoly	1,190	1,283	166	119	218; 18%
Chess	2,376	7,204	664	634	1,168; 16%
Validator	6,359	5,604	501	377	674; 12%
Cli	2,001	1,701	159	121	220; 13%
Total		15,792	1,490	1,251	2,280; 14%
Reduction		—	**35%**	**45%**	0%

Table 3: Number of Tests derived

Subject Systems		Mean number of tests for FOMs	SSHOMs	SSHOMsRed
Monopoly	Mean	29	27	28
	Std. dev.	3.9	3.1	2.7
Chess	Mean	190	158	157
	Std. dev.	6.7	4.8	3.8
Validator	Mean	73	65	64
	Std. dev.	4.3	3.6	2.8
Cli	Mean	32	28	27
	Std. dev.	2.9	2.7	2.8
Total (averaged) tests		324	278	276
Reduction in tests needed		0	14%	15%

We plot a Proportional Kill Growth (PKG) graph for all mutants and study the distribution of subsumed FOMs on this graph. This allows us to see how subsumed mutants distribute over the PKG. For example, we can see whether subsumed mutants tend to be killed earlier, later or evenly in the overall mutation testing process. Finally, for completeness, we ask how many of the unsubsumed FOMs would have been killed using the test suites selected to kill subsumed FOMs and their SSHOMs:

RQ4.4: Unsumbsumed coverage: How many unsubsumed FOMs do subsumed FOMs and their SSHOMs kill?

4. RESULTS

This section presents the results that answer the four research questions set out in the previous section (Section 3).

RQ1.1: Baseline Data on Mutants

Table 2 provides baseline data. It shows the numbers of FOMs generated from each of the four programs and the numbers of SSHOMs generated from the FOMs using the search process defined in Section 2. The table also gives the sizes of the reduced set SSHOMsRed produced by minimization. In the final column, the table reports the number of FOMs (and the percentages of all FOMs this denotes) that are subsumed by the SSHOM set.

As we can see from Table 2, 14% of the first order mutants are subsumed by SSHOMs. Also, these data allow us to quantify the degree of saving possible using subsuming higher order mutants. Over all programs, we found that the saving achievable using SSHOMs over the FOMs they subsume is 35%, increasing to 45% if we use the reduced set SSHOMsRed.

RQ1.2: Baseline Data on Test Suites

Table 3 shows the mean number and the standard deviation of the number of test cases selected for each of the four programs and for each of the three classes of mutant.

Table 3 shows that the number of test cases selected to achieve 100% FOM adequacy is larger than the number required to achieve 100% SSHOM adequacy, on average.

This suggests that SSHOMs can potentially reduce the number of tests required without losing test effectiveness. It is also noticeable that the number of tests derived from the complete set of SSHOMs and the reduced set (SSHOMsRed) are almost identical (each differing by at most one test case, on average, which is well within a single standard deviation).

This is to be expected, since the reduction the minimization algorithm is designed to reduce the number of mutants without affecting the effectiveness of the mutant set. We should, therefore, expect a similar number of test cases would be required to kill the minimized set of strongly subsuming higher order mutants as are required to kill the unreduced set.

For the test engineer's point of view, it is interesting to note that using tests selected to kill strongly subsuming HOMs results in a 14%-15% reduction in the number of test cases that need to be selected. In cases where there is no automated oracle, and the test engineer must play this role, this may be a welcome reduction in test effort, especially if it is accompanied by an increase in test effectiveness (a question explored in RQ3).

RQ2: Higher Order Mutant Potency

The AKG and PKG graphs for each of the three classes of mutants are depicted in Figure 4. In both AKG and PKG graphs, the average value of all ninety executions was calculated at each point. The AKG graphs show the number of mutants killed as more test cases are executed, while the PKG graph shows the proportion of all mutants in the class (FOM, SSHOM or SSHOMsRed) that are killed. The dashed line (representing the mutation score obtained with FOMs) is always above other lines, which indicates FOMs are killed faster than SSHOMs independently of the test cases used.

From Table 2 we know that the average saving in numbers of mutants needed, when SSHOMs are used in place of the FOMs is 35% (for SSHOMs), rising to 45% for the reduced set of SSHOMs. However, though fewer SSHOMs are needed, they are also harder to kill. These growth graphs, provide a visualization this reduced killability.

As can be seen from Figure 4, FOMs are killed at a greater rate than SSHOMs in absolute terms. There are also proportionately more FOMs killed than SSHOMs at every stage of the test process until the very final stages of testing, when almost all test cases have been executed. As a result, we can be reasonably sure that, were testing to cease at any stage, a greater number and proportion of FOMs would be killed than SSHOMs.

To examine the observation in more detail, statistical analyses were carried out as explained in Section 3. Table 4 shows the results of a (paired) Wilcoxon signed-rank test. In Table 4, the p values obtained are all lower than 0.00049, indicating strong significance at the 0.05 level (after accounting for multiple testing with the Bonferroni correction).

Each statistical test involves 90 sets of paired data points. There is one set of paired points for each set of test data. Each paired data element is the mutation score reached by the two mutation techniques being compared, after the same number of test cases has been executed.

To interpret the Cohen's d effect size measurement we use a standard range of effect sizes from 'small' to 'large' [5]. The effect size is 'medium' between FOMs and both SSHOMs and the reduced set SSHOMsRed.

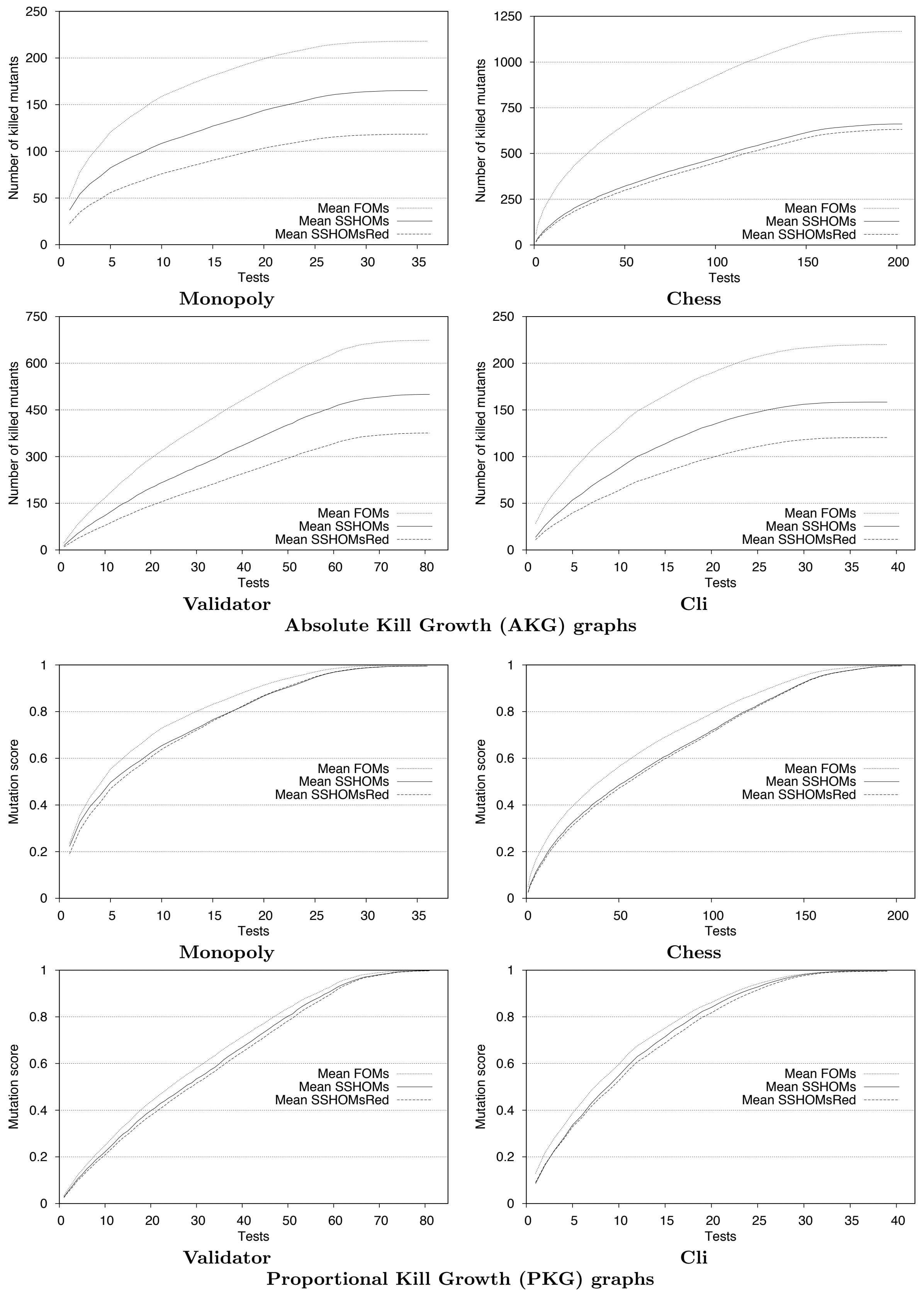

Figure 4: **The Potency of Strongly Subsuming Higher Order Mutants**: Absolute/Proportional Kill Growth ({A/P}KG) graphs showing the {absolute/proportional} numbers of mutants killed by all test cases for each of the three classes of mutants: First Order Mutants (FOMs), Strongly Subsuming Higher Order Mutants (SSHOMs) and Reduced SSHOMs (SSHOMsRed). The graphs reveal that the rate at which SSHOMs are killed is much lower than for FOMs in both absolute terms and also proportionally (relative to the numbers of mutants in each class). We also observe a noticeable further saving in the absolute numbers of mutants using the reduced set of SSHOMs for three out of the four programs studied.

Table 4: Wilcoxon Signed-Rank test results

	Comparing FOMs with SSHOMs	Comparing FOMs with SSHOMsRed	Comparing SSHOMs with SSHOMsRed
Z	-134.477	-139.946	-97.386
Sig.	0.000	0.000	0.000
Effect Size	0.586	0.610	0.424

The reduction in mutants achieved by mimimizing the SSHOMs set to SSHOMsRed is also significant and the effect, though not as large as the effect size for FOMs, is still appreciable. In this case, the effect size is typically regarded as between 'small' and 'medium'. We conclude that using strongly subsuming HOMs in place of the FOMs they subsume significantly reduces the number of mutants required and that this is a statistically medium sized effect.

RQ3: Higher order test effectiveness

From RQ2 we know how much harder it is to kill higher order mutants compared to first order mutants. Furthermore, from RQ1, we know that the test suites selected to kill SSHOMs contain 15% fewer tests than those selected to kill FOMs (yet they kill all FOMs and SSHOMs). This is interesting to a mutation tester, but for a software engineer, interested in testing in general, the more important question will be whether the test suites selected to achieve coverage of higher order mutants are, in some sense 'better', not merely smaller, than those test suites selected for the more traditional, first order, mutation criteria.

As with all testing techniques, the determination of which technique is better can vary depending on context. However, in mutation testing there is a way to order two test suites, A and B according to their mutation score on some set of mutants M. Suppose test suite T_{SSHOM} is selected to kill mutants in $SSHOM$ and test suite T_{FOM} is selected to kill mutants from the set FOM, then we can compare each test suite to see how good the suite is at killing mutants used to select the other kind of mutants. However, the test suites may have different size, so a direct comparison of scores is not appropriate. Rather, we compare the growth in mutant scores as we order the test cases, as before, thereby catering for differing sizes of test suite.

Figure 5 shows the results of comparing the mutation score achieved by the tests selected to kill the mutants in FOMs, SSHOMs, SSHOMsRed.

As we can see from this figure, for all four subjects, the results show that, though test suites derived from SSHOMs contain fewer test cases, they nevertheless, achieve both higher mutation score overall and, more importantly, do so faster then test suites derived from FOMs.

We also observe, in all cases, that the test suites selected to kill FOMs fail to reach 100% mutant score. This is because there are HOMs SSHOMs that are not killed by *any* of the test cases select to kill FOMs: over all thirty test suites selected to kill all FOMs, 2.2, 7.3, 5 and 1.7 of the HOMs remain unkilled, on average, for the programs Monopoly, Chess, Validator and Cli respectively.

This provides additional evidence that confirms the observation that test suites selected to kill FOMs are inferior to test suites selected to kill SSHOMs. It also reveals that, for all programs studied, some of the SSHOMs are, indeed, very 'stubborn'; hard to kill yet not equivalent [19].

Table 5: Kruskal-Wallis test results

Test set selected to kill	N	Mean rank	
FOMs	29205	38387	$\chi^2 = 117.872$
SSHOMs	24990	40258	
SSHOMsRed	24879	40164	$p < 0.000$
Total	79074		

Table 6: Dunn's method results

Kind of mutants from which tests are selected	Z	Sig.	Adj. Sig.	Effect Size
FOMs–SSHOMs	-9.513	0.000	0.000	0.041
FOMs–SSHOMsRed	-9.026	0.000	0.000	0.039
SSHOMs–SSHOMsRed	0.458	0.647	1.000	0.002

Table 5 presents the results of the Kruskal-Wallis test. In this table N is the total number of test executions (tests $\times$ mutants) required to achieve 100% mutation score for each of the three criteria (over thirty repeated samples of test cases). Since these test suites are unpaired, their sizes differ; more test cases were executed according to the FOM criteria. The result (Sig.<0.000) indicates a strongly significant difference exists between the three test suite executions; they are very unlikely to be sampled from the same population distribution.

The Kruskal-Wallis test was run to determine whether there were significant differences between tests generated from the set of FOMs, SSHOMs and SSHOMsRed. Pairwise comparisons were performed using Dunn's method (as explained in Section 3). This post-hoc analysis, the results of which are shown in Table 6, reveals statistically significant differences in mutation score between the tests selected to kill FOMs and those selected to kill SSHOMs ($p = 0.000$) and also between the tests selected to kill FOMs and those selected to kill SSHOMsRed ($p = 0.000$).

However there was no statistical difference between the tests selected to kill SSHOMs and those selected to kill SSHOMsRed ($p = 0.647$). The effect size between the FOM tests and the SSHOM tests and FOM tests and the SSHOMsRed tests was assessed as 'small' according to the Cohen's d measurement.

We conclude that a small, but significant, reduction in the numbers of test cases required for mutation adequacy can be obtained using strongly subsuming higher order mutants in place of the first order mutants that they cover. Furthermore, though we found in RQ2 that minimization had a significant effect on the numbers of mutants in the SSHOM sets, it has no significant effect on the number of test cases required to achieve mutation adequacy.

Overall, we find a 5.6% average improvement in test effectiveness using tests selected to kill SSHOMs in place of test selected to kill FOMs. That is, it requires 5.6% fewer tests on average to kill all mutants (FOMs and HOMs) using tests selected to kill the SSHOMs compared to those selected to kill the FOMs.

This figure might downplay the potential improvement in test effectiveness, because it is computed as an average over all points in the test process, thereby including cases such as those near the 'end' of the process, when almost all tests have been executed in any case.

If we consider, instead, the maximum effectiveness improvement achieved over the test process, this gives a contrasting upper bound on the effectiveness improvement that can be expected.

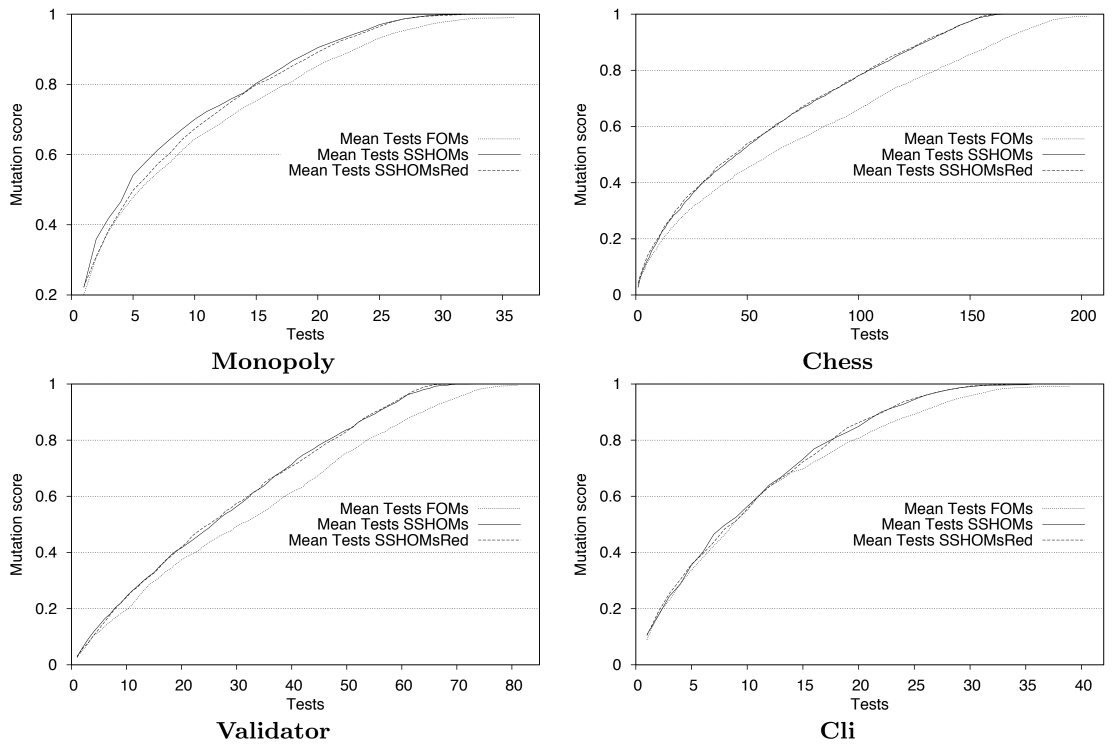

Figure 5: **The Quality of Tests Selected to Kill Strongly Subsuming Higher Order Mutants**: The four figures show, for each program studied, the growth in mutants killed, over all mutants (subsumed FOMs and the HOMs that strongly subsume them) for each of the three kinds of test suite. The graphs visualise the increased test quality (in terms of mutant score) for test suites selected to kill strongly subsuming mutants.

The maximum saving achieved using tests selected to kill SSHOMs was 6.6%, 12.4%, 11.3% and 5.5% for the programs Monopoly, Chess, Validator and Cli respectively. These improvements in effectiveness are obtained while simultaneously reducing the number of test cases required by 14% and the number of mutants by 35%.

We also found that mimization of SSHOM test suites has only a very small impact on effectiveness. Though, as reported in RQ2, minimization of the set SSHOMs to SSHOMsRed, improves the reduction in mutants from 35% to 45%, it also decreases the improvement in test effectiveness that can be achieved by a (small) amount.

That is, though we found an average improvement in effectiveness of 5.6% with SSHOM, this was reduced (by only 0.2%) to 5.4% when using SSHOMsRed. Similarly, we found (only) a 0.5% reduction in the maximum effectiveness improvement (overall all four programs). This is to be expected, since minimization seeks to remove only 'redundant' test cases [50].

RQ4: Characteristics of Subsumed FOMs

RQs 1, 2 and 3 show that subsumed FOMs can be combined to make SSHOMs that reduce test effort by approximately 14%-15% while simultaneously improving test effectiveness by 5.6%-12%. We next investigate whether there are any special properties of the subsumed FOMs we have been able to find compared to the unsubsumed FOMs.

RQ4.1: Subsumed FOM killability

Figure 6 shows the killability of subsumed and unsubsumed mutant sets for each of the four programs studied.

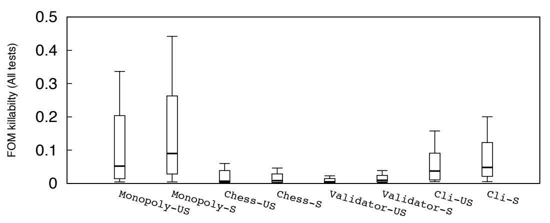

Figure 6: Killability distributions for subsumed(–S) and unsubsumed (–US) mutants over all test cases

As can be seen, the mutants of programs Chess and Validator are noticeably harder to kill (in general) than those of the other two programs. The average killability of the subsumed FOMs appears to be similar (or higher) than that of the unsubsumed FOMs in all cases. In all cases except Chess, this difference is statistically significant (according to a Wilcoxon test) but the effect size is small in all cases (Cohen's d is 0.120, 0.210 and 0.117 for Monopoly, Validator and Cli respectively).

RQ4.2: Subsumed FOM killability by Covering Tests

Figure 7 shows the killability of subsumed and unsubsumed mutant sets for each of the four programs studied, where killability is restricted to test cases the cover the mutant in question. That is, a mutant is only deemed to be unkilled when a test case reaches (and therefore executes) it but fails to kill it. This factors out the effects of programs that contain branches that are hard to reach.

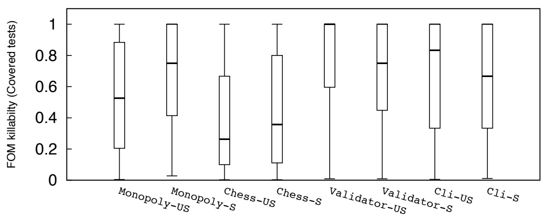

Figure 7: Killability distributions for subsumed(–S) and unsubsumed (–US) mutants with tests that reach (cover) them

Table 7: Unsubsumed FOMs Killed by test selected to kill subsumed FOMs and their SSHOMs

Subject	Tests to kill	%age unsubsumed FOMs killed
Chess	FOMs	61
	SSHOMs	60
	SSHOMsRed	60
Cli	FOMs	56
	SSHOMs	56
	SSHOMsRed	55
Monopoly	FOMs	64
	SSHOMs	64
	SSHOMsRed	64
Validator	FOMs	35
	SSHOMs	34
	SSHOMsRed	34

In this case, comparing Figures 6 and 7 we can see immediately that the apparent reduced killability of the mutants for programs Chess and Validator was largely a product of these programs containing mutants that are seldom executed. As can be seen, for two programs (Monopoly and Chess), subsumed mutants are, on average, a little easier to kill by tests that execute them, while in the other two cases (Validator and Cli), the subsumed mutants are a little harder to Kill. The differences are statistically significant (according to the Wilcoxon test), but, once again, the effect size is small according to Cohen's d measure (0.154, 0.053, 0.210 and 0.055 for Monopoly, Chess, validator and Cli, respectively).

RQ4.3: Subsumed FOM distribution over all FOMs

The Proportional Kill Growth (PKG) graphs for all mutants are depicted in Figure 8. The PKG graphs are drawn as a faint solid line on which we superimpose an indication (with short lines and whiskers) of the proportion of subsumed FOMs that are killed. The PKG graphs are averaged over all test suites and so the locations of the whisker plots give an indication of the expected proportion of subsumed FOMs that would be killed as testing progresses. As can be seen, there are different subsumed FOM distribution patterns for each of the programs, but overall Subsumed FOMs are certainly no harder to kill than that other FOMs.

RQ4.4: Unsumbsumed coverage: How many unsubsumed FOMs do subsumed FOMs and their SSHOMs kill?

Table 7 shows the percentage of unsubsumed FOMs killed by tests selected to kill each of subsumed FOMs, SSHOMs and the reduced set of SSHOMs. As can be seen, even though subsumed FOMs are only 15% of all mutants, the test suites that kill them can kill 34% to 60% of the unsubsumed FOMs (45% to 75% of all FOMs).

Overall, in answer to RQ4, we find that in all programs, a large number of subsumed FOMs are easy to kill. From a software engineer's point of view they are 'angel' faults; easy to detect.

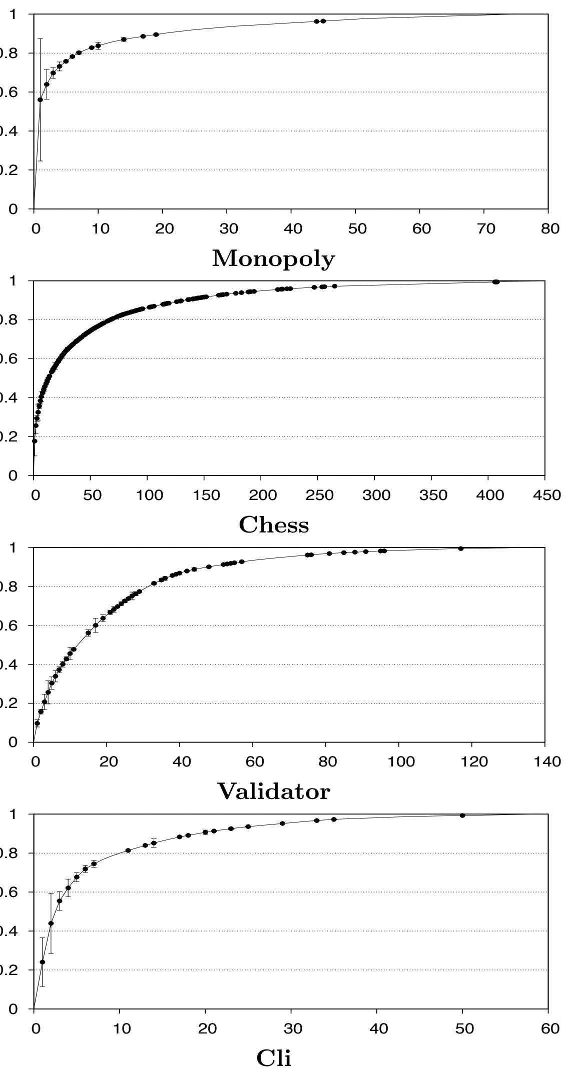

Figure 8: Distributions of killed subsumed FOMs over all FOMs killed as testing progresses

However, we also found that there exist higher order combinations of these angels that are, from a software engineer's point of view, 'monsters': the entire test suite cannot detect them. Nevertheless, scary though they may first appear, we also know (from the answers to RQs 1, 2 and 3) that we can exploit these monsters to help simultaneously improve test effort and effectiveness.

5. THREATS TO VALIDITY AND FUTURE WORK

Our study has the typical threats to validity that impinge on research investigations reporting on a set of studied programs; we cannot be sure that our results will generalise to wider classes of programs and our results are obviously specific to Java programs. We sought to take care of construct threats that may otherwise have emerged due to multiple inferential statistical testing and uncertainties about the distributions from which we sampled. However, another important potential threat arises because of the need to use 'hand constructed' test suites.

We wanted the pool of test cases from which we selected test cases to be both realistic and to exhibit high coverage so that we would not observe spurious effects that might have otherwise arisen by limiting our study to only those mutants that can be easily killed. Though there have been advances on automated test case generation, tools remain unable to achieve high coverage in all cases [29]. Therefore, we used the human-designed test suites (that are provided with the four programs we studied). More work is require to assess the impact of SSHOM-testing on, for example, tests constructed by automated test generation tools or those that are considered to be of poor quality or low coverage by their developers.

Our results are primarily of interest to mutation testing researchers and designers of mutation testing systems[1]. For example, if further research can identify procedures that find more SSHOMs and, thereby, subsume a larger set of FOMs, then the effectiveness and efficiency improvements we report would have a significant impact, reducing the oracle cost and increasing test effectiveness. Future work may also consider whether static analyses can be adapted to predict the combinations of FOMs that would likely lead to SSHOMs. If so, then the 45% reduction in the numbers of mutants required would have a significant effect on the cost of mutation testing (even when oracle testing can be fully automated).

6. RELATED WORK

Mutation testing has been used to assess test quality for a wide range of systems [3, 48, 49], to compare test adequacy criteria [2, 11, 36, 35], and to generate tests [9, 14] (and test oracle [9, 13, 47]). It has also be used to guide test generation and selection using non-mutation based techniques [8, 20, 52]. There are several publicly available tools for mutation testing [6, 22, 46, 31] and mutation faults have been demonstrated to be reasonable simulations of real faults [1, 26]. A recent comprehensive survey of mutation testing is provided by Jia and Harman [24].

There has recent interest in mutant subsumption as a means of improving mutation effectiveness by removing subsumed mutants from consideration for first order mutation testing [25, 28]. The idea of subsuming higher order mutation was first proposed by Jia and Harman [21, 23]. They introduced the concept of subsuming and strongly subsuming HOMs and suggested to combine higher order mutants and non-trivial first order mutants together for mutation testing. The present paper is the fist to report results strongly subsuming mutants and their relationship to the first order mutants they subsume.

Interest in the higher order mutation testing paradigm has grown in recent years, with many results [4, 27, 32, 34, 39, 41, 42]. We summarise these results and their relationship to our work in the remainder of this section.

Polo et al. [42] investigated second order mutants. They proposed different algorithms to combine first order mutants to generate second order mutants. By applying the second order mutants, test effort was reduced by 50%, without great loss of test effectiveness. However, Polo et al. did not use search based optimization and so they were limited to small number of lower order mutants.

[1] All results can be found at the paper's companion website: `http://www0.cs.ucl.ac.uk/staff/Y.Jia/projects/ase14/`

Mateo et al. [34] extended this work with additional combination strategies for second order mutants. Neither of these papers used strong subsumption, so they achieve greater efficiency improvements than those reported in the present paper, but with a (small) decrease in effectiveness rather than a (small) increase in effectiveness.

Jia and Harman [21] and Omar et al. [39] studied the way in which a HOM could combine FOMs in subtle ways. While Jia and Harman generated subtle HOMs for the C language, Omar et al. extended this to Java and AspectJ. Fault subtlety comes from the way in which the HOM's constituent FOMs are mutually partially masking. Langdon et al. [30] found HOMs using a grammar-based, bi-objective, strongly typed genetic programming system. Their two objectives are killability and diversity, which found realistic HOMs that are harder to kill than any of the FOMs in four C subject systems. However, they neither investigated nor reported on strong subsumption.

Higher order mutation may also reduce the effects of the equivalent mutant problem [19], in which a mutant is syntactically different yet semantically identical to the original program from which it is constructed. Offutt [37] was the first to suggest that HOMs might be less likely to be equivalent than FOMs. This observation was also confirmed in the subsequent work of Papadakis and Malevris on second order mutation [41]. More recently, Kintis et al. also showed that impact analysis (with second order mutants) can be used to detect first order equivalent mutants [27]. In the present paper, we are concerned only with non-equivalent mutants, since we wish to study strongly subsuming higher order mutants and the first order mutants that they subsume.

7. CONCLUSIONS AND FUTURE WORK

This paper studied the SSHOMs found in four real world Java systems. We found that, using SSHOMs in place of the FOMs they subsume results in a requirement for many fewer mutants (up to 45% fewer) and that test efficiency can be improved by approximately 14%, while test effectiveness is simultaneously improved by between 5.6% and 12%.

We also found that SSHOMs can be exceptionally hard to detect. These SSHOMs are 'monster faults' that remain undetected by the entire test suite selected to kill all FOMs, yet their constituent FOMs are 'angelic faults' that are easy to kill and therefore offer little resistance to testing. Nevertheless, despite their potency, the SSHOMs we found do still need to be combined with *some* (unsubsumed) FOMs in order to achieve full mutation adequacy.

Our fitness function merely samples strongly subsuming HOMS from a set of subsuming HOMs. We do not specifically search for strong subsumption, merely subsumption, so our results form a lower bound on the improvement in effectiveness and efficiency that might accrue from strong subsumption.

Though we have demonstrated that monster faults can be constructed from angelic faults, we have surely not found the best construction algorithm. We hope and believe that future work will see the benefits of HOMs migrating from research to practice with wider incorporation of higher order mutation in testing tools. We also expect future research will find better targeted fitness functions that seek out various forms of strong subsumption.

8. REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is Mutation an Appropriate Tool for Testing Experiments? In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, pages 402 – 411, St Louis, Missouri, 15-21 May 2005.

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. *IEEE Transactions on Software Engineering*, 32(8):608–624, August 2006.

[3] A. Bartel, B. Baudry, F. Munoz, J. Klein, T. Mouelhi, and Y. L. Traon. Model driven mutation applied to adaptive systems testing. In *Mutation Testing Workshop*, pages 408–413, 2011.

[4] F. Belli, N. Güler, A. Hollmann, G. Suna, and E. Yoldoz. Model-Based Higher-Order Mutation Analysis. In *Advances in Software Engineering*, volume 117 of *Communications in Computer and Information Science*, pages 164–173. Springer Berlin Heidelberg, 2010.

[5] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences (second ed.)*. Lawrence Erlbaum Associates, New Jersey, 1988.

[6] M. E. Delamaro, J. C. Maldonado, and A. Vincenzi. Proteum/IM 2.0: An Integrated Mutation Testing Environment. In *Proceedings of the 1st Workshop on Mutation Analysis (MUTATION'00)*, pages 91–101, San Jose, California, 6-7 October 2001. published in book form, as *Mutation Testing for the New Century*.

[7] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practical programmer. *IEEE Computer*, 11:31–41, 1978.

[8] G. Fraser and F. Wotawa. Using model-checkers for mutation-based test-case generation, coverage analysis and specification analysis. In *International Conference on Software Engineering Advances (ICSEA 2006)*, pages 1–16, 2006.

[9] G. Fraser and A. Zeller. Mutation-driven generation of unit tests and oracles. In *International Symposium on Software Testing and Analysis (ISSTA 2010)*, pages 147–158, Trento, Italy, 2010. ACM.

[10] F. G. Freitas and J. T. Souza. Ten years of search based software engineering: A bibliometric analysis. In 3^{rd} *International Symposium on Search based Software Engineering (SSBSE 2011)*, pages 18–32, 10th - 12th September 2011.

[11] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov. Comparing non-adequate test suites using coverage criteria. In M. Pezzè and M. Harman, editors, *International Symposium on Software Testing and Analysis (ISSTA 2013)*, pages 302–313, Lugano, Switzerland, 2013. ACM.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[13] B. J. M. Grün, D. Schuler, and A. Zeller. The Impact of Equivalent Mutants. In *Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION'09)*, pages 192–199, Denver, Colorado, 1-4 April 2009. IEEE Computer Society. published with *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation Workshops*.

[14] M. Harman, Y. Jia, and B. Langdon. Strong higher order mutation-based test data generation. In 8^{th} *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11)*, pages 212–222, New York, NY, USA, September 5th - 9th 2011. ACM.

[15] M. Harman, Y. Jia, and W. B. Langdon. A manifesto for higher order mutation testing. In 5^{th} *International Workshop on Mutation Analysis (Mutation 2010)*, Paris, France, April 2010.

[16] M. Harman, A. Mansouri, and Y. Zhang. Search based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, November 2012.

[17] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. A comprehensive survey of trends in oracles for software testing. Technical Report Research Memoranda CS-13-01, Department of Computer Science, University of Sheffield, 2013.

[18] M. Harman, P. McMinn, J. Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In B. Meyer and M. Nordio, editors, *Empirical software engineering and verification: LASER 2009-2010*, pages 1–59. Springer, 2012. LNCS 7007.

[19] M. Harman, X. Yao, and Y. Jia. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In 36^{th} *International Conference on Software Engineering (ICSE 2014)*, Hyderabad, India, June 2014. To Appear.

[20] S.-S. Hou, L. Zhang, T. Xie, H. Mei, and J.-S. Sun. Applying Interface-Contract Mutation in Regression Testing of Component-Based Software. In *Proceedings of the 23rd International Conference on Software Maintenance (ICSM'07)*, pages 174–183, Paris, France, 2-5 October 2007.

[21] Y. Jia and M. Harman. Constructing Subtle Faults Using Higher Order Mutation Testing. In *Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation (SCAM'08)*, pages 249–258, Beijing, China, 28-29 September 2008.

[22] Y. Jia and M. Harman. Milu: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In 3^{rd} *Testing Academia and Industry Conference - Practice and Research Techniques (TAIC PART'08)*, pages 94–98, Windsor, UK, August 2008.

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

[24] Y. Jia and M. Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions of Software Engineering*, 37(5):649–678, 2011.

[25] R. Just, M. D. Ernst, and G. Fraser. Efficient mutation analysis by propagating and partitioning infected execution states. In *International Symposium on Software Testing and Analysis (ISSTA 2014)*, San Jose, CA, USA, 2014. To appear.

[26] 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? Technical Report UW-CSE-14-02-02, University of Washington, 2014.

[27] M. Kintis, M. Papadakis, and N. Malevris. Isolating first order equivalent mutants via second order mutation. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, ICST '12, pages 701–710, Washington, DC, USA, 2012. IEEE Computer Society.

[28] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng. Mutant subsumption graphs. In 10th *Mutation Testing Workshop (Mutation 2014)*, Cleveland Ohio, USA, March 2014. To appear.

[29] K. Lakhotia, P. McMinn, and M. Harman. Automated test data generation for coverage: Haven't we solved this problem yet? In 4th *Testing Academia and Industry Conference — Practice And Research Techniques (TAIC PART'09)*, pages 95–104, Windsor, UK, 4th–6th September 2009.

[30] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, 83(12):2416–2430, 2010.

[31] Y.-S. Ma, A. J. Offutt, and Y.-R. Kwon. MuJava: a Mutation System for Java. In *Proceedings of the 28th international Conference on Software Engineering (ICSE '06)*, pages 827–830, Shanghai, China, 20-28 May 2006.

[32] P. Madiraju and A. S. Namin. ParaMu - A Partial and Higher-Order Mutation Tool with Concurrency Operators. In *Proceedings of the 6th International Workshop on Mutation Analysis (Mutation 2011)*, Berlin, Germany, March 2011.

[33] P. R. Mateo and M. P. Usaola. Mutant execution cost reduction: Through music (mutant schema improved with extra code). In *Proceedings of the 7th Workshop on Mutation Analysis (MUTATION'12)*, pages 664–672, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[34] R. P. Mateo, P. U. Macario, F. Alemán, and J. Luis. Validating second-order mutation at system level. *IEEE Transactions of Software Engineering*, 39(4):570 – 587, April 2013.

[35] A. S. Namin and J. H. Andrews. The Influence of Size and Coverage on Test Suite Effectiveness. In *Proceedings of the18th International Symposium on Software Testing and Analysis (ISSTA'09)*, pages 57–68, Chicago, Illinois, USA, 19-23 July 2009.

[36] A. S. Namin, J. H. Andrews, and D. J. Murdoch. Sufficient Mutation Operators for Measuring Test Effectiveness. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 351–360, Leipzig, Germany, 10-18 May 2008.

[37] A. J. Offutt. Investigations of the Software Testing Coupling Effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, January 1992.

[38] A. J. Offutt, G. Rothermel, and C. Zapf. An experimental evaluation of selective mutation. In 15th *International Conference on Software Engineering (ICSE 1993)*, pages 100–107. IEEE Computer Society Press, Apr. 1993.

[39] E. Omar, S. Ghosh, and D. Whitley. Constructing subtle higher order mutants for Javer and AspectJ programs. In *International Symposium on Software Reliability Engineering (ISSRE'13)*, pages 340–349. IEEE, 2013.

[40] E. Omar, S. Ghosh, and D. Whitley. Comparing search techniques for finding subtle higher order mutants. In *Conference on Genetic and Evolutionary Computation (GECCO 2014)*, pages 1271–1278. ACM, 2014.

[41] M. Papadakis and N. Malevris. An Empirical Evaluation of the First and Second Order Mutation Testing Strategies. In 5th *Mutation Testing Workshop (Mutation'10)*, Paris, France, 2010.

[42] M. Polo, M. Piattini, and I. Garcia-Rodriguez. Decreasing the Cost of Mutation Testing with Second-Order Mutants. *Software Testing, Verification and Reliability*, 19(2):111 – 131, June 2008.

[43] G. Rothermel, M. Harrold, J. Ronne, and C. Hong. Empirical studies of test suite reduction. *Software Testing, Verification, and Reliability*, 4(2):219–249, December 2002.

[44] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, Aug. 1996.

[45] N. J. Salkind. *Encyclopedia of Measurement and Statistics*. Sage publications, Inc., California, USA, 2007.

[46] D. Schuler and A. Zeller. Javalanche: efficient mutation testing for java. In 7th *joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2009)*, pages 297–298, 2009.

[47] M. Staats, G. Gay, and M. P. E. Heimdahl. Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing. In 34th *International Conference on Software Engineering (ICSE 2012)*, pages 870–880, 2012.

[48] J. Tuya, M. J. S. Cabal, and C. de la Riva. Mutating Database Queries. *Information and Software Technology*, 49(4):398–417, April 2007.

[49] R. H. Untch, A. J. Offutt, and M. J. Harrold. Mutation Analysis Using Mutant Schemata. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'93)*, pages 139–148, Cambridge, Massachusetts, 1993.

[50] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. *Journal of Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[51] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei. Bridging the gap between the total and additional test-case prioritization strategies. In 35th *International Conference on Software Engineering (ICSE 2013)*, pages 192–201, San Francisco, CA, USA, May 2013.

[52] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. Regression mutation testing. In *International Symposium on Software Testing and Analysis, (ISSTA 2012)*, pages 331–341, 2012.