Analyzing the Validity of Selective Mutation with Dominator Mutants

Bob Kurtz
Software Engineering
George Mason University
Fairfax VA, USA
rkurtz2@gmu.edu

Paul Ammann
Software Engineering
George Mason University
Fairfax VA, USA
pammann@gmu.edu

Jeff Offutt
Software Engineering
George Mason University
Fairfax VA, USA
offutt@gmu.edu

Márcio E. Delamaro
Instituto de Ciências Matemáticas e de Computaçao
Universidade de São Paulo
São Carlos, SP, Brazil
delamaro@icmc.usp.br

Mariet Kurtz
The MITRE Corporation
McLean VA, USA
mkurtz@mitre.org

Nida Gökçe
Department of Statistics
Muğla Sıtkı Koçman University
Muğla, Turkey
nidagokce@yahoo.com

ABSTRACT

Various forms of selective mutation testing have long been accepted as valid approximations to full mutation testing. This paper presents counterevidence to traditional selective mutation. The recent development of dominator mutants and minimal mutation analysis lets us analyze selective mutation without the noise introduced by the redundancy inherent in traditional mutation. We then exhaustively evaluate all small sets of mutation operators for the Proteum mutation system and determine dominator mutation scores and required work for each of these sets on an empirical test bed. The results show that *all* possible selective mutation approaches have poor dominator mutation scores on at least some of these programs. This suggests that to achieve high performance with respect to full mutation analysis, selective approaches will have to become more sophisticated, possibly by choosing mutants based on the specifics of the artifact under test, that is, specialized selective mutation.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

Mutation analysis, subsumption, dominator mutants

1. INTRODUCTION

Mutation testing [9] is a test criterion that generates a set of alternate programs, called *mutants*, and then challenges the tester to design tests to detect the mutants. Tests that cause a mutant to behave differently from the original program are said to detect, or *kill*, the mutant. Some mutants behave exactly the same as the original on all inputs. These are called *equivalent mutants* and cannot be killed. Mutants are generated by mutation operators. A mutation operator is a rule that generates variants of a given program based on the occurrence of particular syntactic elements.

Early on, researchers observed that mutation operators produced far more mutants than necessary [30]. One response to this observation was selective mutation, which deliberately limits the number of mutation operators to a small, carefully chosen set. In selective mutation, a reduced set of mutation operators is chosen to generate a reduced number of mutants. These mutants are intended to represent the larger body of potential mutants that could be generated by additional operators.

One problem with measuring the effectiveness of selective mutation is the very redundancy that selective mutation is intended to tame. Specifically, the redundant mutants introduce noise into mutation scores. For example, some mutants are killed by almost any test. Hence, eliminating such mutants from consideration does not affect which tests are chosen, but does result in a different mutation score. In other words, mutation scores can be inflated by redundant mutants, making the mutation score harder to interpret. This same problem of diminished utility of score due to redundancy has long been recognized in statement and branch coverage metrics [5].

Minimal mutation, a recent development, precisely defines redundancy among mutants by identifying *dominator* mutants. Prior research [2, 22] shows that *dominator mutation scores* are not consistent with traditional mutation scores for some subsets of mutation operators [2]. Papadakis *et al.* [31] confirmed this observation in a more rigorous study involving larger real-world programs. This inconsistency motivates us to revisit selective mutation using dominator mutation score as a key measure.

Selective mutation hypothesizes that there is a reduced set of mutation operators that, when applied to all programs, consistently elicits test suites that are nearly as effective as test suites based on all mutation operators. Any example

FSE'16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2950322

where selective mutation fails to work consistently challenges this underlying hypothesis. Proper experimental design suggests that we start with simple test subjects, and expand to larger more complex test subjects only if the hypothesis holds true in the simple cases. The Siemens suite [15] of programs is a well-known set of such simple test subjects, with the advantage of extensive test suites. In addition, the Siemens programs are written in C, which suggests the use of the Proteum [6] mutation tool. Proteum is especially well-suited to this experiment because it has a broad set of mutation operators compared to other tools.

The main result of the paper is negative: a selective approach that is highly effective for one program will almost certainly perform poorly on another program. This variance is extremely undesirable: test engineers need to be confident that the analysis techniques they apply have predictable results. One way out of this difficulty might be for selective mutation analysis techniques to be specialized to the program under test, in essence choosing a small yet effective set of mutants for that particular program. The goal of this paper is to demonstrate the need for specialized selective sets; actually finding those sets is out of current scope.

The contributions of this paper are:

1. A definition of *normalized work* to allow comparison of different selective mutation operator sets based on the effort required to develop a test suite.

2. A deterministic method for finding worst-case mutation scores based on mutant subsumption relationships.

3. An evaluation that shows that all one-size-fits-all approaches to selective mutation performs poorly on some programs.

4. A confirmation using dominator mutation scores that random mutant selection and statement deletion perform as badly as traditional selective mutation.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes an approach to measure redundancy and equivalency within a set of mutants and proposes a metric for measuring work required to develop a mutation-adequate test suite. Section 4 introduces our research questions, which are explored in Sections 5, 6, 7, and 8. Section 9 discusses threats to validity. Section 10 summarizes our conclusions.

2. RELATED WORK

The large number of mutants generated by mutation testing has long been a recognized problem. Mathur [25] determined that the complexity of mutation testing is $O(n^2)$, where n is the size of the program under test, and introduced the idea of *constrained mutation* to reduce that complexity to $O(n)$ by reducing the number of mutation operators to create fewer mutants. Offutt *et al.* [29, 30] took an empirical approach to defining an appropriate set of selective mutation operators, and proposed the *E-selective* set of five operators[1] based on achieving a mutation score of 0.99 or higher over ten small programs. Wong *et al.* [35, 36] evaluated combinations of mutation operators for efficiency and effectiveness. This paper reassesses the performance of *E-Selective* mutation using dominator mutation score.

Barbosa *et al.* [3] applied a well defined set of guidelines to obtain a sufficient set of mutation operators that would substantially reduce the computational cost of mutation testing without losing effectiveness. They applied such guidelines in two experiments with two different sets of C programs. They obtained reduced sets of mutant operators that would produce effective test cases, but these sets of sufficient operators were substantially diverse between the experiments, showing that it was not possible to select a single set of operators that was optimal for both programs. Namin *et al.* [26, 27, 28] analyzed the Siemens suite programs using variable reduction techniques to identify three high-performing operator sets using between seven and 13 operators. Delamaro *et al.* [8] defined a growth model for mutation operator selection, adding operators using a greedy algorithm until a mutation score of 1.00 was achieved, and concluded that there is no single way to select the best set of operators for any particular program.

Taking mutation operator reduction to an extreme, Untch [34] evaluated the performance of the statement deletion (SDL) operator on its own and found it to be competitive with the operator sets found by Namin. Deng *et al.* [11] applied the SDL operator to 40 classes written in Java using the muJava tool [24] and found that SDL achieved a mutation score close to that of Offutt's *E-selective* operators while generating approximately 80% fewer mutants. Delamaro *et al.* [7] evaluated the SDL operator against programs written in C using Proteum and confirmed Deng's findings.

Kaminski *et al.* [18, 19] were the first to consider mutation operators at the next level of detail, recognizing that the relational operator replacement (ROR) mutation operator has many sub-operators (replacing '>' with '<', '!=', etc.). They showed that, for any given relational operator, three mutants will always weakly subsume the other four mutants, making them redundant. Lindström and Márki [23] later showed that this subsumption does not always hold under strong mutation. Just, Kapfhammer, and Schweiggert [16] performed a similar analysis for the conditional operator replacement operators, and Yao *et al.* [38] found similar results for the arithmetic operator replacement mutation operators. Just and Schweiggert [17] identified seven sub-operators that are needed to form a sufficient set of non-redundant mutants, analyzed a set of real-world programs, and determined that redundant mutants cause an inflated mutation score that fails to accurately reflect the effectiveness of a test suite. The performance of selective mutation approaches at the sub-operator level, while certainly of interest, is beyond the scope of this paper.

Other researchers have examined whether selective mutation is more effective than random sampling of similar numbers of mutants. Acree [1] and Budd [4] separately concluded that executing tests that kill a randomly-selected 10% of mutants could provide results close to executing tests that kill the full set of mutants. Wong and Mathur [37] demonstrated similar results and found that adding randomly-selected mutants beyond 10% yielded comparatively small improvements. More recently, Zhang *et al.* [39] explored selective mutation and random selection using the Proteum mutation tool and the Siemens suite programs and also found

[1]The *E-selective* operators, derived from Mothra [10], are absolute value insertion (ABS), arithmetic operator replacement (AOR), logical connector replacement (LCR), relational operator replacement (ROR), and unary operator insertion (UOI).

no appreciable difference in performance between selective mutation and random selection.

Gopinath *et al.* [13] expanded this investigation using a much larger body of open-source code and compared several different mutation selection strategies with random selection, again finding that random selection performs as well as any other strategy. In a later paper, Gopinath *et al.* [12] took a different approach to dealing with the large number of mutants, and showed that determining the mutation score based on as few as 1,000 randomly-selected mutants provides an estimate of quality of a test suite in terms of mutation score. It seems counterintuitive that a targeted approach to mutant selection should perform no better than a random approach.

While part of the explanation might be the effect of mutant redundancy on mutation score [22], this paper suggests that an additional reason might be that existing mutation selection strategies, including random selection, perform poorly when evaluated in terms of dominator score.

In our previous paper [22] we showed that dominator mutation score was a superior metric to traditional mutation score for determining how much testing work has been done and how much remains. We also classified sets of mutants based on *redundancy* and *equivalency* and examined the effect on the ability to determine test completeness. In this paper, we use the same characterization for sets of mutants generated by selective operators.

3. DOMINATOR MUTANTS, REDUNDANCY, AND EQUIVALENCY

3.1 Subsumption and Dominator Mutants

In previous papers we defined a method for determining the relationship between mutants based on their behavior with respect to a set of tests [2, 20]. Given a finite set of mutants M and a finite set of tests T, mutant m_i is said to *dynamically subsume* mutant m_j if some test in T kills m_i and every test in T that kills m_i also kills m_j. Where two mutants m_i and m_j in M are killed by exactly the same tests in T, we say that m_i and m_j are *indistinguished*.

We capture the subsumption relationship among mutants with a directed graph, the *Dynamic Mutant Subsumption Graph* or DMSG [20]. Each node in the DMSG represents a maximal set of indistinguished mutants, and each edge represents the dynamic subsumption relationship between two sets of mutants. More specifically, if m_i dynamically subsumes m_j, then there is an edge from the node containing m_i to the node containing m_j.

Consider a set of ten mutants, each named for the mutation operator that created it followed by a sequence number. These mutants are tested using a set T of four tests, where the mutants are killed by tests as shown in the score function in Table 1.

We can see that mutants OABN_1 and OEAA_1 are indistinguished, since they are killed by exactly the same test set ($\{t_1\}$). Similarly, mutants OEAA_2 and OLBN_1 are indistinguished, as are OEAA_3 and VLSR_1. We can also see that every test that kills CCDL_1 also kills OEAA_2, OEAA_3, OLBN_1, and VLSR_1. Consequently, CCDL_1 subsumes these mutants. We can determine all of the subsumption relationships between these mutants and construct the resulting DMSG as shown in Figure 1. Mutants that are not subsumed by any other mutants are called *dominator mutants*; these are listed in the graph in dominator nodes, which we denote with a double-line.

Table 1: Example score function

	t_1	t_2	t_3	t_4
CCDL_1			t	
CCDL_2				
OABN_1	t			
OALN_1	t			t
OBBN_1				t
OEAA_1	t			
OEAA_2	t		t	
OEAA_3	t	t	t	t
OLBN_1	t		t	
VLSR_1	t	t	t	t

Figure 1 has three dominator nodes; a mutant from each forms a dominator mutant set. In this example, the three mutants {CCDL_1, OABN_1, OBBN_1} form a dominator mutant set, as do the three mutants {CCDL_1, OEAA_1, OBBN_1}. Because each dominator set contains one mutant from each dominator node, all dominator sets are equally useful and a dominator set can be selected arbitrarily from all possible sets. Consequently, only three of the ten mutants matter–if tests kill the mutants in a dominator mutant set, they are guaranteed to kill all non-equivalent mutants. All other mutants are redundant. This leads us to adopt the *dominator mutation score* or *dominator score* as a more precise metric. We define the dominator score as the number of killed mutants in a dominator set divided by the total number of mutants in the dominator set.

Mutant CCDL_2 is not killed by any of the tests in T, so it is shown in its own unconnected node with a dashed border. CCDL_2 might be equivalent or it might be killable, but not by any of the four tests in T. From the DMSG perspective, which is based on a finite test set, we can not distinguish between these two possiblities. In this study, we compute dominator scores with respect to mutants that are killed by our full test sets. For example, consider the test set $\{t_1\}$, which kills the shaded mutants in Figure 1. The mutation score for test set $\{t_1\}$ is $7 \div 9 = 0.78$, but its dominator score with respect to T is only $1 \div 3 = 0.33$.

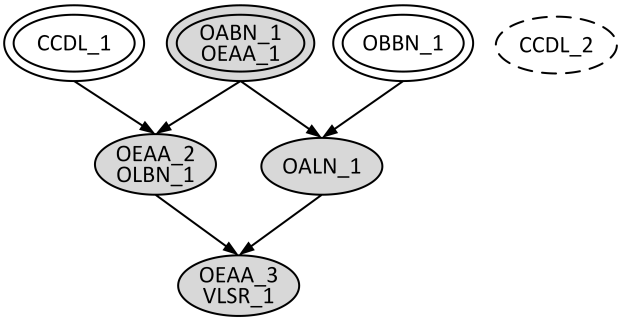

Figure 1: Example DMSG

The DMSG represents the subsumption relationship between all mutants with respect to the test set. If we kill any mutant in the DMSG, we are *guaranteed* to kill all the mutants that it subsumes [2], i.e. all connected mutants below it in the graph. A test that kills a mutant may *by chance*

also kill mutants above it or elsewhere in the graph, but this is not guaranteed by the subsumption relationship.

Dominator mutants tend to be hard to kill, in the sense that they are usually killed by relatively few tests. However the relationship between dominator mutants and hard-to-kill mutants as identified in other research is not strict; it is possible to have a dominator mutant that is killed by many tests, yet is not subsumed by any other mutants.

3.2 Measuring Redundancy and Equivalency

In our previous paper [22] we defined *redundancy* as the number of killable mutants that are not in the dominator mutant set divided by the number of mutants in the dominator set, as shown in Equation 1. In prior research, the number of equivalent mutants has typically been measured as a percentage of all mutants generated. However, this naturally couples the measurement of equivalent mutants with the measurement of redundant mutants. We seek a metric for equivalent mutants that is independent of redundancy, so we define *equivalency* as the number of equivalent mutants divided by the size of the dominator set, as shown in Equation 2.

$$redundancy = \frac{|killableMutants| - |dominatorSet|}{|dominatorSet|} \quad (1)$$

$$equivalency = \frac{|equivalentMutants|}{|dominatorSet|} \quad (2)$$

4. RESEARCH QUESTIONS

Researchers frequently use mutation score to evaluate the effectiveness of test suites designed by other means. Mutation score is defined as the number of mutants killed by a test suite, divided by the total number of non-equivalent (killable) mutants. Our previous paper [22] showed that mutation score has a non-linear relationship with test completeness due to redundancy among mutants, rendering it of limited usefulness for determining how much testing work has been completed. We now seek the impact of redundant and equivalent mutants in other ways, and we consider these research questions:

RQ1: How does redundancy and equivalency affect the amount of work required to develop mutation-adequate tests?

RQ2: Do the *E-selective* mutation operators identified by Offutt *et al.* [29, 30] reliably generate high dominator mutation scores across a range of programs?

RQ3: Is there a small set of mutation operators that improves upon *E-selective* and consistently generates higher dominator mutation scores with low work across a range of programs?

RQ4: With respect to dominator score and work, how do any improved selective operator sets compare with *E-selective* operators, random mutant selection, and statement deletion?

5. RQ1: HOW REDUNDANT AND EQUIVALENT MUTANTS AFFECT WORK

Consider an engineer testing the `print_tokens` program from Table 2 using Proteum. Proteum first generates a set of mutants. In a simplified model of software testing, the engineer selects a mutant for analysis and tries to write a test to kill the mutant; if successful, the tool removes that mutant from the set of unkilled mutants. Then the tool runs the tests on all other unkilled mutants and removes any that are also killed. If the engineer determines the mutant is equivalant, then it is removed. This process is repeated until the set of unkilled mutants is empty. Of course, this is not an efficient process, nor is it the process that an engineer would use; an engineer would likely generate some tests by other means and then use mutation testing to evaluate and enhance that test suite. Nevertheless it is a convenient idealization for our purpose, and it helps us to understand what happens across the whole range of testing.

We simulate this process using a process described in detail in our previous paper on redundancy and equivalency [22]. We generate the mutants using Proteum, then create a score function that indicates which mutants are killed by which tests. We select a mutant at random, then randomly select a test that kills it. If we can find such a test, we remove the selected mutant and all other mutants that are killed by the selected test. Note that some tests may kill the selected mutant and many additional mutants, while others may kill few or no additional mutants. If no tests kill the mutant, we assume it is equivalent and discard it. We repeat this process until all mutants are killed or identified as equivalent. Because the process involves random selection of mutants and tests, we repeat the analysis 1,000 times and derive an average score.

To estimate the work required to develop a test set, we must first define work. An obvious metric for work is time expended to develop tests and identify equivalent mutants. However, some mutants are easily killed, while some are much more difficult. Grün *et al.* [14] and Schuler and Zeller [32] found that it took nearly 15 minutes on average to determine that a single mutant was equivalent; on the other hand some equivalent mutants (e.g. those of the form "`return a++`") are trivially easy to identify. Because time is not easily quantified, we define work as *the number of mutants that are examined by the engineer*, or, in other words, the sum of the number of tests written to kill all non-equivalent mutants and the number of equivalent mutants identified, as shown in Equation 3.[2] While this definition of work is easily understood, it makes it difficult to directly compare the multiple programs in the Siemens suite, because the test sets and number of equivalent mutants vary between programs. To compare effectively between programs, we introduce *normalized work*, which we define as work divided by the number of dominator mutants, as shown in Equation 4. In other words, normalized work of 1.0 is the amount of work required to kill a dominator set of mutants by picking the least-advantageous set of tests, such that each test kills only one of the dominators. With zero redundancy and zero equivalency, we expect normalized work to be somewhat below 1.0, since it is likely that some of the selected tests will kill not only the intended dominator mutant but other dominators as well and somewhat fewer than one test per dominator will be required.

$$work = |testSet| + |equivalentMutants| \quad (3)$$

$$normalizedWork = \frac{|testSet| + |equivalentMutants|}{|dominatorSet|} \quad (4)$$

[2]It is possible that identifying equivalent mutants is easier or harder than developing a test case. A more sophisticated model might weight these tasks differently.

Note that we measure work on a per-trial basis. If during a particular trial we have written five tests and identified two equivalent mutants, our work at that point is $5+2=7$. If the mutant set contains ten dominator mutants, then our normalized work at that point is $(5+2) \div 10 = 0.70$.

Imagine a version of Proteum that generates only a dominator mutant set with no redundant or equivalent mutants. Such a tool might work by intelligent selection of mutation operators, by static analysis and filtering of mutants, or by some other mechanism. The mechanism is irrelevant so long as the results are achieved. Mutation score and dominator score are identical, because all mutants are dominator mutants.

Next imagine a version of Proteum that generates redundant mutants in addition to a dominator set, but that does not generate equivalent mutants. If it generates a number of redundant mutants equal to the size of the dominator set (that is, twice as many mutants as before), then the redundancy is 1.0 as per Equation 1. We simulate this by selecting a dominator set and then selecting other killable mutants at random until we have the proper number.

To determine the effect of redundancy on work required, we repeat the engineer-focused approach described above, selecting increasing redundancy from 0.0 to 50.0 at increments of 0.5. Equivalent mutants are eliminated from consideration. The results are shown in Figure 2, where the columns show the mean normalized work and the error bars show the 2σ variation in normalized work. With no redundancy, the normalized work is 0.59; this means that just over half as many tests as dominator mutants were needed to kill all of the dominators. There is significant variation in normalized work caused by random selection of tests. In some cases we find a remarkably small test that set kills the dominators with very little work, while in other cases we need nearly one test per dominator.

As redundancy is increased, the mean work increases only slightly. Even with redundancy of 50.0, normalized work is 0.71. With 50 times as many mutants, the total effort to produce a mutation-adequate test set increases by only 20%!

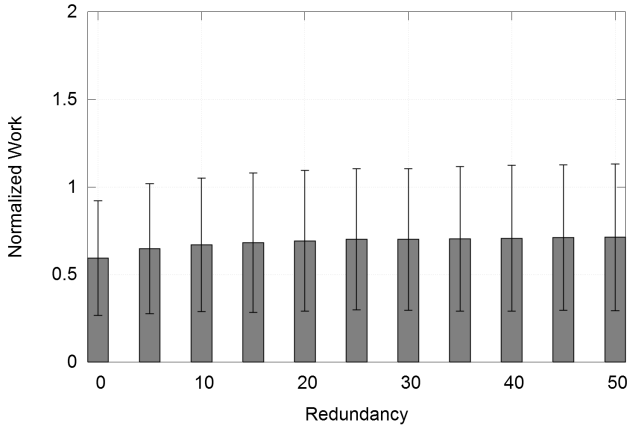

Figure 2: Work to develop a mutation-adequate test set for the Siemens suite with equivalency=0.0

To determine the effect of equivalency on work required, we repeat the approach described above, this time holding redundancy constant and varying equivalency. Because redundancy does not significantly impact work, and because we measure equivalency in terms of dominator mutants, there is no need to consider the effect of equivalency at different redundancy levels. Redundancy is simply held at 0.0.

As equivalency is increased, the mean work increases linearly,[3] as shown in Figure 3.

From the engineer's perspective, all of that extra work is wasted because it does not directly contribute to a mutation-adequate test set.

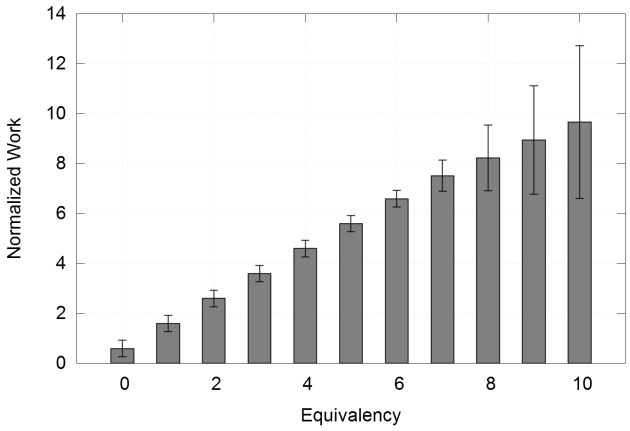

Figure 3: Work to develop a mutation-adequate test set for the Siemens suite with redundancy=0.0

6. RQ2: ANALYZING E-SELECTIVE MUTATION

To answer our second research question, we analyzed the Siemens suite of programs using the Proteum mutation tool to create mutants for the seven programs in Table 2, and executed each program against a subset of 512 tests. We created a score function for each analyzed program that shows which tests kill which mutants. In a previous paper, we analyzed all un-killed mutants and determined that only a small number of these mutants are killed by the full Siemens test set but not killed by the 512 selected tests [2]. In terms of mutation score, the 512 tests kill more than 99% of the non-equivalent mutants; those mutants that were not killed by any of the 512 tests were considered equivalent for our purposes.

We identified all of the mutants created by the *E-selective* operators. These operators map approximately to Proteum operators[4] OAAN, OLLN, OLNG, and ORRN [2]. We then determined a minimal set of tests that kill the non-equivalent mutants using a Monte Carlo approach to determine minimal test sets [2] as shown in Algorithm 1. This algorithm begins by randomly removing one test from the entire test set. If the remaining tests kill all of the mutants of interest, then the selected test is discarded; otherwise the test is restored and another test is randomly selected (without

[3]The growth rate of work is sublinear and the variation increases above equivalency of 7 because Siemens programs `schedule` and `tcas` have a maximum equivalency of just under 7 (see Table 2), and thus cannot contribute the required number of equivalent mutants.

[4]Delamaro *et al.* [8] describes the Proteum mutation operators. Missing from that list are CCDL (Constant Deletion), OODL (Operator Deletion), and VVDL (Variable Deletion).

Table 2: Siemens Suite Programs

Program	LOC	# Mutants	# Dominators	# Equivalents	Redundancy	Equivalency
print_tokens	472	4,322	29	611	153.4	21.8
print_tokens2	399	4,734	31	692	151.7	22.3
replace	512	11,080	59	2,297	186.8	38.9
schedule	292	2,108	43	270	48.0	6.3
schedule2	301	2,626	47	495	54.9	10.5
tcas	141	2,384	62	427	37.5	6.9
totinfo	440	6,693	20	872	333.7	43.6
average	*365.3*	*4,849.6*	*41.4*	*809.1*	*116.1*	*19.5*

replacement). The process is repeated until no tests can be removed from the test set while still killing the desired mutants. The result is a minimal test set.

Algorithm 1: Test set minimization

```
// Input:  Mutant set M and test set T
// Output: A minimal test set

minSet = T
for each t in minSet {
  // Note: t selected arbitrarily
  if (minSet-{t} maintains mutation score wrt M and T) {
    minSet = minSet - {t}
  }
}
return minSet
```

Each minimal test set is guaranteed to kill the mutants of interest, that is, the mutants generated by the selected mutation operators. However, there may be many possible minimal test sets and each one may have a different effect on the remaining mutants not generated by the selected mutation operators, including the dominator mutants. Consequently, we executed 10 runs for each mutation operator combination to determine the average performance of the operator combination.

For each run, we measured the mutation score (based on the number of mutants killed compared with the number of non-equivalent mutants) and the dominator mutation score (based on the total number of all dominator mutants killed as compared with the total number of dominator mutants).

Our results for the programs in the Siemens suite using the *E-selective* mutation operators are shown in Figure 4. This graph shows that while the *E-selective* operators consistently yield high mutation scores (above 0.90), the dominator mutation scores for is considerably lower, ranging from 0.63 to 0.79, with considerably more variation than the mutation score. With respect to question RQ2, we conclude that the *E-selective* mutation operators do not produce consistently high dominator mutation scores across a range of programs.

7. RQ3: IMPROVING UPON E-SELECTIVE MUTATION

Since the *E-selective* operator set does not reliably produce high dominator mutation scores, we look for other operator combinations that have better performance than *E-selective*.

7.1 Analyzing sets of four mutation operators

We repeated the test-based process for all combinations of up to four mutation operators for each program in the Siemens suite. Proteum has 78 operators, and taken one, two, three, and four at a time totals over 1.5 million combinations. The programs actually used only 59 of the Proteum operators, which is still almost 500,000 combinations. The combinations of four operators required about 24 hours of processing on a quad-core Intel I7 platform. Because computation increases by approximately a factor of 10 for each additional operator, this brute-force approach was impractical for larger sets of operators.[5]

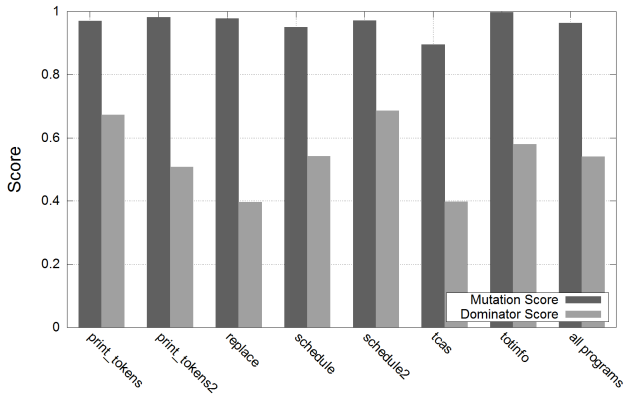

Figure 4: *E-selective* mutation scores for the Siemens suite

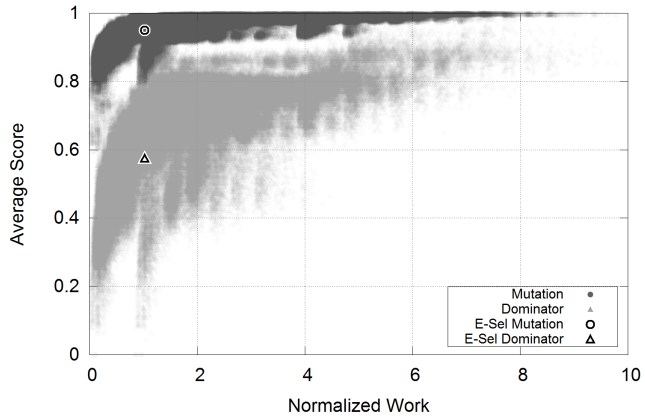

Figure 5: Selective mutation scores for `print_tokens` using 1-4 operators

The results for Siemens program `print_tokens` are shown in Figure 5. This program used 49 different Proteum oper-

[5]See Section 7.2 for an analysis of more than four operators.

ators, and the graph plots 231,525 different combinations of these operators, with two points plotted for each operator combination. Mutation scores are shown in dark gray and dominator mutation scores are shown in light gray. Score values are shown on the vertical axis, and the normalized work required to kill the mutants generated by the operators is shown on the horizontal axis. Bold points show the mutation and dominator mutation scores for the *E-selective* mutation operators.

We see that mutation scores for various mutation operators are fairly closely grouped; they quickly approach 1.0 as mutants are added (and more work is needed to kill them). The *E-selective* set of operators generate a mutation score of 0.97, which is high but not among the best of all operator combinations. Normalized work to kill these mutants is a relatively small 1.02.

The dominator mutation scores show much more variation. While in general sets of operators that require more work score higher, there is considerable variation in the dominator score for similar amounts of work. For example, sets of operators that require work of about 1.0 generate dominator scores between about 0.08 and 0.80. It is possible to achieve dominator scores of 0.90 with normalized work less than 2.0, which seems a very attractive proposition. The *E-selective* operators generate a dominator mutation score of 0.67. It is clear that many selective operator combinations significantly outperform *E-selective* for the `print_tokens` program. The other six Siemens programs show generally similar patterns, with the *E-selective* operators performing roughly in the middle of all sets of selective operators with one to four operators. Many other operator sets have higher dominator scores.

This suggests identifying new selective mutation operators that improve upon *E-selective*. However, when operator combinations are averaged across all seven programs of the Siemens suite, as shown in Figure 6 (489,405 data points), it becomes clear that there is no combination of one to four mutation operators that reliably produces dramatically higher dominator mutation scores across *all* programs. The *E-selective* operators on average produce among the best mutation scores, and while there are better candidates for dominator mutation score, *E-selective* remains a reasonable choice.

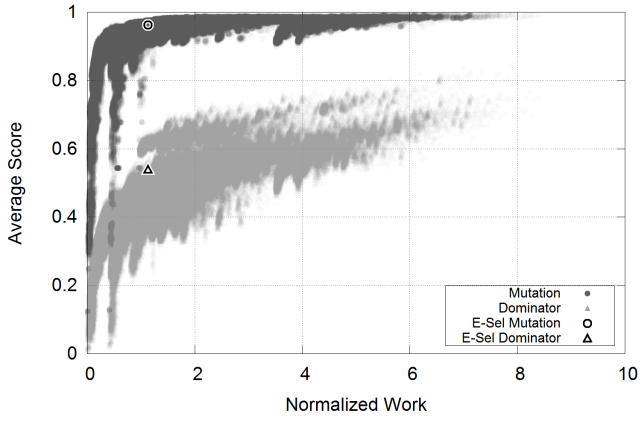

Figure 6: Selective mutatation scores for the Siemens suite

With respect to RQ3, we conclude that while each individual program may have many operator combinations that outperform *E-selective*, on average across multiple programs, it is among the better of a collection of poor performers.

7.2 Analyzing more than four mutation operators

If we cannot find a set of up to four operators that reliably generates a very high score, what is the best that we can find, and how does it compare to *E-selective*?

It may be possible to find sets of more than four mutation operators that produce high dominator mutation scores across our sample programs. Unfortunately it is not practical to evaluate arbitrarily large sets of operators using the brute-force techniques from Section 7.1, since with 59 operators represented in the Siemens programs there are $(2^{59})-1$, or almost 6 x 10^{17}, operator combinations. Even checking combinations of five operators results in more than five million combinations, and combinations of six operators results in more than 50 million. If we wish to explore larger sets of operators, we must take a different approach.

To expand the number of operators, we replaced the average-case Monte Carlo approach to determining mutation scores with a deterministic approach that would be computationally faster and would eliminate the need to perform repeated runs to determine an average. We created such a deterministic approach based on the dynamic subsumption relationship between mutants.

Computationally, using the DMSG allows us to reduce the amount of data processed for each operator set and, because it is deterministic, it allows us to eliminate the averaging of multiple executions as was done with the average-case analysis. This substantially reduces the analysis effort.

To compute the DMSG-based scores, we first identify all mutants in the DMSG that are generated by the selected mutation operators. We know that there is some set of tests that kill these mutants, though we no longer need to be concerned with exactly which tests they may be. Because any other mutants in the same nodes as the generated mutants have the exact same behavior as the generated mutants, these are also killed by the same tests. Finally, all mutants in all nodes subsumed by the generated mutants are also killed by the same tests.

As an example, consider Figure 1. We choose mutation operators OBBN and OLBN as our selective operator set. Mutants OBBN_1 and OLBN_1 (shown in black nodes and marked with asterisks) are generated by these operators and are killed by some set of tests. Mutant OEAA_2 is in the same DMSG node as OLBN_1, so this mutant is also killed. Mutant OALN_1 is subsumed by OBBN_1 and mutants OEAA_3 and VLSR_1 are subsumed by both OBBN_1 and OLBN_1, so these mutants (shown in gray nodes) are killed as well. This results in a mutation score of 0.67 (six of the nine non-equivalent mutants killed), but a dominator mutation score of 0.33 (kills one of three mutants from a dominator mutant set).

We would like to use this easily-computed DMSG-based score as a proxy for the more computationally-intensive average-case score. However, the DMSG-based approach does not compute the same scores as the test-based approach. The DMSG-based approach determines a worst-case score; scores evaluated using minimal test sets may actually be higher. Consider the same example using a test-based ap-

proach to determining mutation and dominator mutation scores. To kill selected mutant OBBN_1, we must choose test t_4. To kill selected mutant OLBN_1 we may choose either test t_1 or t_3; assume for this example we choose t_1. This test also kills OEAA_2. Test t_4 kills subsumed mutant OALN_1 and both tests kill subsumed mutants OEAA_3 and VLSR_1. However, t_1 also kills dominator mutants OABN_1 and OEAA_1, which were not killed in the worst-case approach based on subsumption–we are simply lucky to have killed these additional mutants. The resulting mutation score is 0.89 (eight of nine non-equivalent mutants killed), and the dominator mutation score is 0.67 (kills two of three mutants from a dominator mutant set), both of which are significantly higher than the worst-case approach.

To reliably use the worst-case score as a proxy for the average-case score, we must have confidence that there is a correlation between the two scoring approaches, not only with respect to dominator mutation score, but also with respect to work as defined in Section 5. Specifically, we need to know that sets of selective operators that generate higher worst-case dominator scores tend to generate higher average-case dominator scores. This allows us to find the best operator combinations using worst-case analysis, with the assurance that they are also among the best using average-case scoring.

We generated worst-case scores for all 489,405 combinations of one to four mutation operators for the seven Siemens programs, with the results shown in Figure 7. Compared to the average-case scores in Figure 6, these worst-case mutation scores are slightly lower, but the worst-case dominator mutation scores are significantly lower.

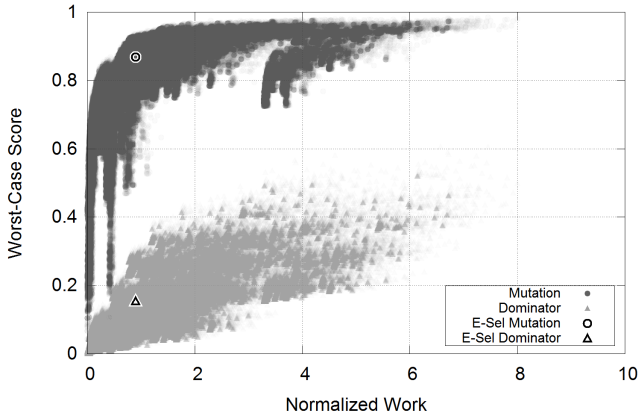

Figure 7: Selective mutation scores for the Siemens suite (worst-case)

We then compared the worst-case dominator scores to the earlier average-case dominator scores. Figure 8 shows all combinations of one to four operators graphed with worst-case dominator scores on the X-axis and average-case dominator scores on the Y-axis. Using Spearman's rank correlation, we found a strong positive monotonic correlation between worst-case and average case scores ($\rho = 0.966$, $p = 0.000$, $n = 489405$, $\alpha = 0.01$). Normalized work correlates even more strongly ($\rho = 0.999$, $p = 0.000$, $n = 489405$, $\alpha = 0.01$). This confirms the validity of using worst-case dominator score as a proxy for average-case dominator score.

While this approach is more computationally efficient than

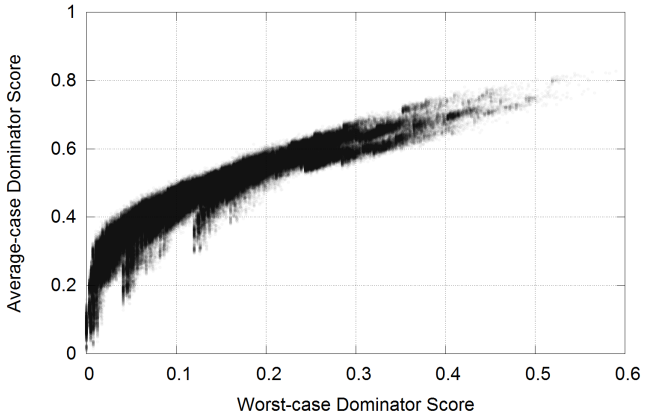

Figure 8: Dominator score correlation

averaging the results of multiple tests, we can further reduce analysis effort by eliminating computation of mutation operator sets that are unlikely to produce good results. Starting with $N = 1$, we performed a breadth-first search for all combinations of operators of length N for each of the seven Siemens suite programs individually, selecting up to a total working set of 100,000 best combinations. Each combination of length N that was within the 100,000 top scores served as the basis for combinations of length $N + 1$; combinations that were not among the top scores were not explored at additional lengths.

We measure best in this context by searching for operator combinations that have higher dominator scores and lower work than other combinations. Consider points E, H, I, and J, shown in the shaded area in Figure 9. These points are all dominated by point B, which has a higher dominator score and lower work. The points within the shaded area are sub-optimal solutions as compared with B. If we remove all sub-optimal solutions dominated by any other point, we are left with points A, B, C, and D. These points form the *Pareto front* of optimal points.

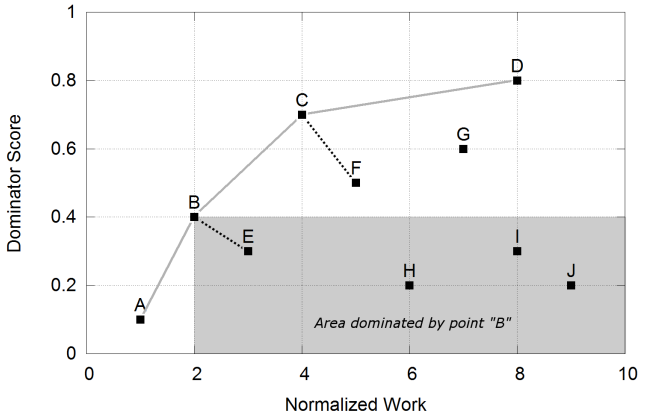

Figure 9: Example Pareto front

We compute the goodness of points that are dominated by the Pareto front by using their *Hausdorff distance* (d_H), the distance between the dominated point and the nearest

member of the Pareto front.[6] Smaller values of d_H indicate more optimal solutions. In Figure 9, the d_H of point E is 1.005, while point F has d_H of 1.020. Thus, point E is closer to the Pareto front and is a superior solution to F.

Because worst-case scores correlate to average-case scores, we assume that optimal worst-case points will be near-optimal when evaluated by the average-case technique, and conversely *near*-optimal worst-case points may prove to be optimal in the average-case analysis. Thus we select more than just the optimal worst-case points for further evaluation. After calculating all combinations of length N, we identify successive Pareto fronts until we have accumulated at least 100,000 best solutions for the next round of evaluation.

This processing resulted in a total of 100,026 operator combinations. To find the very best-performing selective operators in terms of average-case scores, we reprocessed these 100,026 best operator combinations using the test-based average-case approach described in Section 6, and then reduced the operator combinations to the 251 best combinations that represent the Pareto front of optimal solutions. The results are shown in Figure 10, with optimal dominator scores shown in medium gray. For comparison, dominator scores for all combinations of one to four operators are shown in light gray.

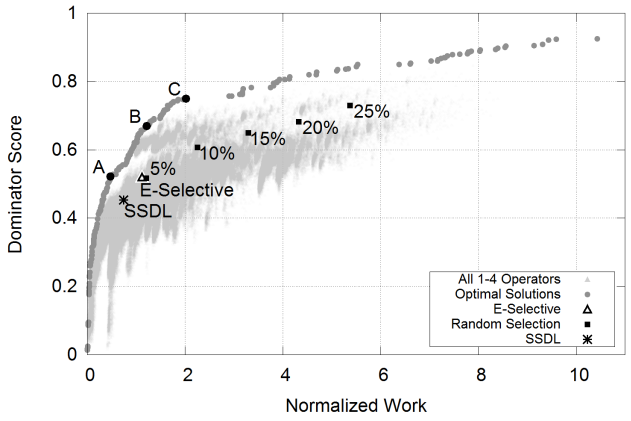

Figure 10: Best 251 operator combinations for all Siemens programs (average-case)

While it is tempting to identify a best set of operators based on this data, the best choice actually depends on the risk tolerance of the program. Engineers developing non-critical programs might select a set of operators that provide a modest dominator mutation score with a small amount of work, while those developing mission-critical programs might be willing to perform far more work to achieve a higher dominator score. Thus, every point in the optimal solutions plot is a potential best choice for some situations.

Nevertheless, a few points stand out for additional discussion. If we are satisfied with the dominator score that *E-selective* achieves, we can accomplish the same score with less than half the work (from 1.12 to 0.47) by using operator set {OABN, OBSN, OLSN, SMTC, SSWM, STRI, VGPR, VLPR, VSCR}, which produces the point labeled A. Alternatively, if we are willing to perform the same amount of work as for *E-selective*, we can significantly increase our dominator score (from 0.52 to 0.67) by using the operator set {OABN, OALN, OLSN, SMTC, SMTT, SSWM, VGPR, VTWD}, which produces the point labeled B. Finally, we can see a knee in the curve at $work \approx 2.0$ using operator set {OABN, OLSN, Oido, SMTC, SMTT, SSWM, STRI, SWDD, VGAR, VGSR, VLPR, VSCR, VTWD, VVDL} which produces point C; beyond this point substantially more work is required to achieve even modestly higher dominator scores.

There are several potential selective operator sets that are an improvement over *E-selective*, but such improvements are clearly incremental. No matter how many operators we use in our combinations, our optimum solutions are only moderately better than the best combinations of one to four operators. There is no combination of operators that reliably produces high dominator scores for small amounts of work across our selection of programs. Our desire for a high-performing set of operators that are program-independent remains unfulfilled, and a customized program-dependent approach to mutant selection is still needed.

With respect to question RQ3, we conclude that no sets of selective mutation operators of any size consistently produce among the very best dominator mutation scores with a small amount of required work across a range of programs.

8. RQ4: COMPARISON TO OTHER SELECTIVE TECHNIQUES

Several empirical studies have indicated that there is no significant difference in performance between selective mutation approaches and random selection of mutants. In this section we compare random with the best operator sets that we have discovered with respect to dominator score and normalized work. To develop the random scores, we select mutants at random until we reach the desired percentage of total mutants. As before, this is repeated ten times for each of the seven Siemens suite programs, and the results are averaged.

We evaluated random selection at five different percentages of mutants generated: 5% (similar to the number of mutants generated by *E-selective*), 10% (suggested by numerous authors [1, 4, 37] as a threshold for very high mutation scores), 15%, 20%, and 25%. We also evaluated the performance of the statement deletion operator (SSDL) on its own. All of these were compared to the Pareto front of most optimal selective operator sets, as shown in Figure 10.

Random selection of 5% of mutants performs very similarly to *E-selective*, but all random selection approaches are sub-optimal compared to the best selective operators found in our evaluation. Similarly, the SSDL operator is also sub-optimal, requiring less work but delivering a lower dominator score than *E-selective* or any of the random selection approaches. Table 3 shows the Hausdorff distances from the Pareto front for each of these alternate approaches.

With respect to question RQ4, we conclude that random selection and statement deletion are about as effective as current selective mutation operators. This is because traditional mutation score is inflated due to the redundancy between mutants and the *E-selective* operators are not optimized for dominator score. When redundancy is removed with dominator score, alternate sets of selective mutation

[6]Hausdorff distance has a well-known limitation that points in the gaps of a sparse Pareto front score poorly [33] With the relatively dense Pareto fronts generated by our analysis, this is not a significant problem.

Table 3: Hausdorff distance d_H of alternate points

Alternate approach	d_H
SSDL	0.104
E-Selective	0.134
5% Random	0.150
10% Random	0.274
15% Random	0.147
20% Random	0.229
25% Random	0.112

operators can be found that outperform these approaches. All of these current techniques are sub-optimal compared to selective operators specifically selected for high dominator scores and low work.

9. THREATS TO VALIDITY

As in any experimental paper, we must acknowledge threats to validity. The use of the Siemens suite has both advantages and disadvantages. The Siemens suite is a well-known set of programs that has been used many times. However, the individual programs themselves are fairly small and may not be representative of more typical real-world programs. For our particular research, however, we do not consider the disadvantages of the Siemens suite to be disqualifying– if selective mutation has shortcomings with respect to any particular set of programs, then it must be considered suspect with respect to software in general. Future work should expand this analysis to larger real-world programs to verify that the results still hold for those programs.

Our experimental data is from Proteum-specific mutation operators and unkilled mutants were not verified to be equivalent. The Proteum limitation of 512 tests leaves a small percentage of unkilled mutants that we treat as equivalent. As a result, the specific operator sets identified in Section 7.2 may not be the very best operator sets possible. Nevertheless, the broader point remains that common selective mutation approaches based on any fixed set of operators or on random selection of mutants cannot provide high dominator scores with low amounts of work over a range of programs.

Of course, our modeling decisions also limit our results. Our model of how the test engineer proceeds, namely one test or one equivalent mutant at a time, is a simplification of how testing occurs in practice. Our model further assumes that the effort to find a test to kill a mutant is comparable to the effort to declare a mutant equivalent. More precise models would more closely match the actual test process as well as weight tests and equivalent mutants differently. For example, a model that placed more weight on the work of equivalent mutant detection would tend to rate approaches such as SDL, which are explicitly designed to limit equivalent mutant generation, more favorably than *E-Selective* or random, which do not.

10. CONCLUSIONS

Scientists and practitioners alike have long known that mutation systems create large numbers of redundant mutants. What is new is a way to measure that redundancy [22], and moreover, to find a *non-redundant* set of dominator mutants [2, 20, 21]. The concept of dominator mutants allows us to redefine the mutation score to be significantly more precise.

The first and most important result is that previous results on selective mutation [25, 29, 30, 35, 36, 3, 26, 27, 28] were at best imprecise. The original study, using the Mothra mutation system, concluded that five of the 22 mutation operators could serve as a selective proxy for the rest, because on average, tests that killed all or most of the selective mutants also killed all or most of the complete set. As the results in Section 6 show, these results were biased by two factors. First, they are largely an artifact of the large redundancy inherent in the mutants. When this redundancy is eliminated by finding a dominator set of mutants, the full mutation scores of the selective tests dropped off dramatically.

Second, in the original study the five selective operators performed best on average, but each specific program in our study was best serviced by a different set of operators. Just as medical researchers sometimes argue for personalized medicine where treatment is optimized for individual patients, we suggest that mutation operators should be specialized for individual programs.

Prior results found that selective approaches were comparable with statement deletion [34, 11, 7] and even random selection [37, 39, 13]. This paper finds that these results hold even when modeled in terms of work and the more sensitive metric of dominator score. In particular, *E-selective* and 5% random selection score remarkably similarly in terms of dominator score and work when applied to the Siemens suite. None of these approaches, however, score particularly well compared to selective operator sets that are optimized to maximize dominator score and minimize work, which in turn are worse than operator sets optimized for a particular program. One interpretation is that all one-size-fits-all selective operator approaches are equally far from optimal, and that better approaches are required.

One such approach might involve determining a correlation between program features and mutation operators that tend to produce dominator or near-dominator mutants. This may allow a more sophisticated mutation engine to produce mutants that are specifically tailored to the program under test. We hope to explore this approach in future work.

All of these results are based on finding dominator sets of mutants, which of course is not easy. In our prior research, we leveraged manual analysis [20] and symbolic execution [21] to find approximate dominator set of mutants. The general problem is, unfortunately, undecidable [2]. If we could find dominator sets of mutants, mutation analysis would become dramatically cheaper and more practical. But even without a practical way to efficiently find dominator mutant sets before testing concludes, this paper demonstrates that minimal mutation can be used as an effective research tool to refine and expand our knowledge of mutation.

Acknowledgments

Bob Kurtz' research is supported by Raytheon. Offutt is partly supported by KKS (The Knowledge Foundation), by project 20130085, Testing of Critical System Characteristics (TOCSYC). Mariet Kurtz' affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions, or viewpoints expressed by the author. Gökçe's research is supported by The Scientific and Technological Research Council of Turkey (TUBITAK).

11. REFERENCES

[1] A. T. Acree. *On Mutation*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 1980.

[2] Paul Ammann, Marcio E. Delamaro, and Jeff Offutt. Establishing theoretical minimal sets of mutants. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, pages 21–31, Cleveland, Ohio, USA, March 2014.

[3] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification, and Reliability, Wiley*, 11(2):113–136, June 2001.

[4] T. A. Budd. *Mutation Analysis of Program Test Data*. PhD thesis, Yale University, New Haven, Connecticut, USA, 1980.

[5] T. Chusho. Test data selection and quality estimation based on the concept of essential branches for path testing. *IEEE Transactions on Software Engineering*, 13(5), May 1987.

[6] M. E. Delamaro, J. C. Maldonado, M. Jino, and M. Chaim. Proteum: Uma ferramenta de teste baseada na análise de mutantes (Proteum: A testing tool based on mutation analysis). In *7th Brazilian Symposium on Software Engineering*, pages 31–33, Rio de Janeiro, Brazil, October 1993. In Portuguese.

[7] Marcio E. Delamaro, Lin Deng, Serapilha Dureli, Nan Li, and Jeff Offutt. Experimental evaluation of SDL and one-op mutation for C. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, Cleveland, Ohio, March 2014.

[8] Márcio E. Delamaro, Lin Deng, Nan Li, and Vinicius H. S. Durelli. Growing a reduced set of mutation operators. In *Proceedings of the 2014 Brazilian Symposium on Software Engineering (SBES)*, pages 81–90, Maceió, Alagoas, Brazil, September-October 2014.

[9] Richard A. DeMillo, Richard J. Lipton, and Fred G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, April 1978.

[10] Richard A. DeMillo and Jeff Offutt. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, September 1991.

[11] Lin Deng, Jeff Offutt, and Nan Li. Empirical evaluation of the statement deletion mutation operator. In *6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)*, pages 65–74, Luxembourg, March 2013.

[12] R. Gopinath, Alipour A., Ahmed I., C. Jensen, and A. Groce. How hard does mutation analysis have to be anyway? In *IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2015.

[13] Rahul Gopinath, Amin Alipour, Iftekhar Ahmed, Carlos Jensen, and Alex Groce. Do mutation reduction strategies matter? Technical report, School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon, USA, August 2015.

[14] Bernard Grün, David Schuler, and Andreas Zeller. The impact of equivalent mutants. In *Fifth IEEE Workshop on Mutation Analysis (Mutation 2009)*, pages 192–199, Denver, Colorado, USA, April 2009.

[15] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering (ICSE '94)*, pages 191–200, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[16] René Just, Gregory M. Kapfhammer, and Franz Schweiggert. Do redundant mutants affect the effectiveness and efficiency of mutation analysis? In *Eighth IEEE Workshop on Mutation Analysis (Mutation 2012)*, Montreal, Canada, April 2012.

[17] René Just and Franz Schweiggert. Higher accuracy and lower run time: Efficient mutation analysis using non-redundant mutation operators. *Journal of Software Testing, Verification and Reliability*, 25(5-7):490–507, August-November 2015.

[18] Garrett Kaminski, Paul Ammann, and Jeff Offutt. Better predicate testing. In *Sixth Workshop on Automation of Software Test (AST 2011)*, pages 57–63, Honolulu HI, USA, May 2011.

[19] Garrett Kaminski, Paul Ammann, and Jeff Offutt. Improving logic-based testing. *Journal of Systems and Software, Elsevier*, 86:2002–2012, August 2013.

[20] Bob Kurtz, Paul Ammann, Marcio E. Delamaro, Jeff Offutt, and Lin Deng. Mutation subsumption graphs. In *Tenth IEEE Workshop on Mutation Analysis (Mutation 2014)*, pages 176–185, Cleveland, Ohio, USA, March 2014.

[21] Bob Kurtz, Paul Ammann, and Jeff Offutt. Static analysis of mutant subsumption. In *Eleventh IEEE Workshop on Mutation Analysis (Mutation 2015)*, Graz, Austria, April 2015.

[22] Bob Kurtz, Paul Ammann, Jeff Offutt, and Mariet Kurtz. Are we there yet? How redundant and equivalent mutants affect determination of test completeness. In *Twelfth IEEE Workshop on Mutation Analysis (Mutation 2016)*, Chicago, Illinois, USA, April 2016.

[23] Birgitta Lindström and András Márki. On strong mutation and subsuming mutants. In *Twelfth IEEE Workshop on Mutation Analysis (Mutation 2016)*, Chicago, Illinois, USA, April 2016.

[24] Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon. MuJava: A mutation system for Java. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06), tool demo*, pages 827–830, Shanghai, China, May 2006. IEEE Computer Society Press.

[25] Aditya Mathur. Performance, effectiveness, and reliability issues in software testing. In *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pages 604–605, Tokyo, Japan, September 1991.

[26] Akbar Siami Namin and James H. Andrews. Finding sufficient mutation operators via variable reduction. In *Second IEEE Workshop on Mutation Analysis (Mutation 2006)*, Raleigh, NC, November 2006.

[27] Akbar Siami Namin and James H. Andrews. On sufficiency of mutants. In *Proceedings of the 29th International Conference on Software Engineering, Doctoral Symposium*, pages 73–74, Minneapolis, MN,

USA, 2007. ACM.

[28] Akbar Siami Namin, James H. Andrews, and Duncan J. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th International Conference on Software Engineering*, pages 351–360, Chicago, Illinois, USA, 2008. ACM.

[29] Jeff Offutt, Ammei Lee, Gregg Rothermel, Roland Untch, and Christian Zapf. An experimental determination of sufficient mutation operators. *ACM Transactions on Software Engineering Methodology*, 5(2):99–118, April 1996.

[30] Jeff Offutt, Gregg Rothermel, and Christian Zapf. An experimental evaluation of selective mutation. In *Proceedings of the Fifteenth International Conference on Software Engineering*, pages 100–107, Baltimore, MD, May 1993. IEEE Computer Society Press.

[31] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. Threats to the validity of mutation-based test assessment. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'16)*, Saarbrücken, Germany, July 2016, to appear.

[32] David Schuler and Andreas Zeller. (un-)covering equivalent mutants. In *3rd IEEE International Conference on Software Testing, Verification and Validation (ICST 2010)*, pages 45–54, Paris, France, April 2010.

[33] O. Schutze, X. Esquivel, A. Lara, and C. A. C. Coello. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 16(4):504–522, August 2012.

[34] Roland Untch. On reduced neighborhood mutation analysis using a single mutagenic operator. In *ACM Southeast Regional Conference*, pages 19–21, Clemson SC, March 2009.

[35] W. Eric Wong, M. E. Delamaro, J. C. Maldonado, and Aditya P. Mathur. Constrained mutation in C programs. In *Proceedings of the 8th Brazilian Symposium on Software Engineering*, pages 439–452, Curitiba, Brazil, October 1994.

[36] W. Eric Wong and Aditya P. Mathur. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software, Elsevier*, 31(3):185–196, December 1995.

[37] W. Eric Wong and Aditya P. Mathur. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software*, 31(3):185–196, December 1995.

[38] Xiangjuan Yao, Mark Harman, and Yue Jia. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *Proceedings of the 36th International Conference on Software Engineering*, pages 919–930, Hyderabad, India, May 2014.

[39] Lu Zhang, Shan-San Hou, Jun-Jue Hu, Tao Xie, and Hong Mei. Is operator-based mutant selection superior to random mutant selection? In *32nd ACM/IEEE International Conference on Software Engineering*, pages 435–444, May 2010.