Measuring Effectiveness of Mutant Sets

Rahul Gopinath∗, Amin Alipour‡, Iftekhar Ahmed†, Carlos Jensen§, and Alex Groce¶
Department of EECS, Oregon State University
Email: ∗gopinath@eecs.orst.edu, ‡alipour@eecs.orst.edu, †ahmed@eecs.orst.edu, §cjensen@eecs.orst.edu, ¶agroce@gmail.com

Abstract—Redundancy in mutants, where multiple mutants end up producing the same semantic variant of a program, is a major problem in mutation analysis; a measure of effectiveness that accounts for redundancy is an essential tool for evaluating mutation tools, new operators, and reduction techniques. Previous research suggests using the size of the disjoint mutant set as an effectiveness measure.

We start from a simple premise: test suites need to be judged on both the number of unique variations in specifications they detect (as a variation measure), and also on how good they are at detecting hard-to-find faults (as a measure of thoroughness). Hence, any set of mutants should to be judged on how it supports these measurements.

We show that the disjoint mutant set has two major inadequacies — the single variant assumption and the large test suite assumption — when used as a measure of effectiveness in variation, which stem from its reliance on minimal test suites. We show that when used to emulate hard to find bugs (as a measure of thoroughness), disjoint mutant set discards useful mutants.

We propose two alternative measures: one is oriented toward measuring variation and not vulnerable to either the single variant assumption, or the large test suite assumption; the other is oriented towards measuring thoroughness, and provides a benchmark of these measures using diverse tools.

I. INTRODUCTION

Software engineering relies on mutation analysis [1], [2] as the primary means of ascertaining quality of a test suite. Mutation analysis involves exhaustive generation of a syntactically defined set of faults (called mutations), and evaluation of the ability of a test suite to detect the resulting variants of the program. The ratio of faults detected to faults generated is taken as the mutation score of a test suite, and is taken to correlate with the ratio of detectable variants to total number of variants. Daran et al. [3], Andrews et al. [4], [5], Do et al. [6], and more recently Just et al. [7] suggest that mutation analysis is capable of generating faults that resemble real bugs, the ease of detection for mutations is similar to that for real faults, and the effectiveness of a test suite in detecting real faults is reflected in its mutation score. The terms used in this paper are given in Box 1.

A key concern in mutation analysis is whether the generated mutants are sufficient to generate all possible variants of the program. Given that not all mutants produce unique variants, another key concern [8] is to avoid generation of redundant mutants, using only a single representative mutant for each variant. Researchers have identified various techniques [9]–[24] for reducing redundancy in generated mutants: these include selective mutation, static subsumption, mutation clustering to identify similar mutants, static analysis of generated mutants, and other approaches. Recently [25] it was found that there is limited utility in mutation reduction strategies compared to random sampling, and it was more worthwhile to investigate additional effective mutation operators, rather than strategies for removal of mutants, however intelligent.

Whether it is for comparison between two mutation reduction strategies, or evaluation of effectiveness of a new mutation operator, one requires a measure of effectiveness of a set of mutation operators. Mutation analysis is a means of assessing the quality of a test suite, and any measure of effectiveness should consider how well a given set of mutants achieves this objective. A test suite should be judged on two main criteria: it should be able to detect and prevent as many unique variants as possible (measure of variation), and secondly, it may also be judged on how good it is in detecting subtle bugs (as a measure of thoroughness). Hence, we have corresponding requirements against which a set of mutants may be judged. A given set of mutants may be judged by the ratio of the unique variants it contains to the size of the full set of possible variants. It may also be judged by how hard to detect the variants induced by its mutants are.

Considering the first requirement, given a set of mutants, and a reduction strategy, one may judge the effectiveness of the strategy by the fraction of original unique variants that the reduction strategy was able to preserve in the reduced set. That is, for an ideal reduction strategy, each mutant in the reduced set should correspond to a unique variant, and each variant in the original set of mutants should have a unique mutant associated with it in the reduced set. Theoretically [26]–[28] this can be done by running all possible test sets — $T^O$ against each mutant, identifying its unique signature, and removing those mutants that are subsumed. This is however, undecidable as a consequence of Rice’s theorem [30]. Ammann et al. [27] suggests a compromise. Rely on the minimal test suite (denoted as $T_m$), a subset of the available test suite $T$, such that $T_m$ kills all mutants in the complete set $M$, and identify the minimal subset of mutants (we denote it by $M_d$) that requires at least all the tests in $T_m$ to completely kill (called the disjoint set by Kintis et al. [31], and minimal mutants by Ammann et al. [27]). We adopt the moniker disjoint mutant set because this was the original name, and also because we propose an alternative minimal set of mutants. There is however, a problem.

Mutation reduction techniques and newer mutation operators are not added with a particular test suite in mind. They take into account static characteristics to include or exclude certain mutants, and the set of mutants should not change with the

---

1Traditionally [26], [28], [29] subsumption is based on all execution paths. That is, $kill(m_a, T^U) \implies kill(m_d, T^U)$ when $m_a$ subsumes $m_d$. However, Ammann et al. [27] use dynamic subsumption, which relies only on the available test set. That is, $kill(m_a, T) \implies kill(m_d, T)$ when $m_a$ subsumes $m_d$. 

---
Expanding on the second requirement, it may be argued that identifying the actual hardest mutants to detect is important (not just the size of the set), and disjoint mutant set is a set of mutants that are hardest to detect. In other words, not all the unique variants should be considered with the same weight. Some of the variants are trivially detected by multiple test suites, and hence a measure of effectiveness ought to consider the value of trivial and non-trivial mutants too.

**Does the theoretical minimum set using disjoint mutant set capture all the hardest mutants to detect?** and Is the size of theoretical minimum set using disjoint mutant set the best measure of effectiveness in this respect? Our aim here is to show that disjoint mutant set does not select all the hardest mutants, and that subsumption can be used to define hardness – and thus can be used to provide a more informative measure.

### A. Problems with size of disjoint mutant set as a measure of effectiveness

The explicit assumptions made in theoretical disjoint mutant set are [27]: a comprehensive test suite, and a fixed set of mutants. However, using the theoretical disjoint mutant set as the true set of unique variants involves a few more implied assumptions. Let us imagine that we have a large set $M$ of non-redundant mutants (size $m > t$), each producing a different variant, and a minimum test suite $T$ containing a set of test cases (size $t > 1$), which is adequate to kill every mutant in the set of mutants. Say we create a super test case $t'$ by joining together all other test cases, and add it to $T$, creating $T'$. Plainly, both $T$ and $T'$ are adequate for $M$. Now, according to Ammann’s definition, the size of disjoint mutant set is the same as the size of the corresponding minimal test suite size, which is $t$ if we are using $T$, but $1$ if we are using $T'$. However, we do know that the actual number of variants is $m$. Even if we assume that $t = m$ initially, the actual number of variants is not $1$.

This suggests that if one is looking for the actual true set of unique variants, one has to assume that test cases are small, with no test case killing more than one variant. (If any test case kills more than one variant, the size of the minimal test suite will no longer correspond to the number of unique variants. This is the single variant assumption. The second assumption is that the number of test cases are at least equal to or greater than the number of unique variants — the large test suite assumption.

Even for large mutation adequate test suites, the size of the minimum test suite may be much smaller than the number of variants, because some test cases may detect more than a single variant. This makes size of corresponding disjoint mutant set a less than ideal measure for the number of variants.

---

2 Minimum set is the smallest sized set among minimal sets.

3 The word comprehensive is not defined by Ammann [27], but the rest of the paper suggests that it means mutation adequate test suite.
As a practical example, consider a set of tests for triangle (Box 2) \( \{t_1, t_2, t_3\} \), with corresponding mutant kills \( \{m_a, m_b\}, \{m_b, m_c\}, \{m_c, m_a\} \). Plainly, all the mutants produce different variants, and the three test cases are different from each other in terms of the variants they detect, with none subsumed by others. However, a minimal test suite based on mutation scores discards one of \( t_1, t_2, t_3 \), as not all three are required to maintain the mutation score. The size of the disjoint mutant set is same as the cardinality of the corresponding minimal test suite. This means that the theoretical minimum disjoint mutant set may discard mutants which represent actual unique variants.

There is a simple fix to this problem. Any two mutants can be considered to be representing two different variants if the test cases that kill them are different. We denote a set of mutants such that no pair have similar kill pattern as unique mutants \( M_d \). Such a set has a maximum size limit of \( 2^T \), the maximum resolving power of the test suite, and is not vulnerable to either of the large test suite assumption\(^4\) or the single variant assumption. The size of unique mutant set can be used as a measure to compare two mutant sets of the same size. To compare two tools, we use the ratio \( R_\delta = \frac{|M_d|}{|M_a|} \), which provides the odds that a given mutant is unique.

The second question is more involved. In Box 2, we know that disjoint mutant set discards at least one of the mutants, even though all of them are equivalent in terms of the number of tests they detect, and the pattern of detection. The same is true in the first example too, where adding a super test case results in ignoring most of the useful mutants. Hence the question is, do we have an alternative? What exactly is a trivial mutant? How do we measure effectiveness? Is the size of the non-trivial set the best one can do as a measure of thoroughness?

We develop a theory of hyper-geometric representation of variants such that variants encloes by similar volumes have similar effectiveness irrespective of the number of variants included. We also show that the variants at the surface of this volume (denoted by \( M_s \)) can be computed by a small modification to the computation of disjoint mutant set \( M_d \). We show that the alternative does not result in discarding important mutants in the given example.

This suggests a simple measure of effectiveness (with consideration for triviality). We compute the effectiveness as the ratio of volume of the sphere enclosed to the maximal volume.

\[ M_s \text{ is an alternative to } M_d, \text{ and like } M_d, \text{ selects the possible non-trivial variants in a set of mutants. However, unlike } M_d, \text{ we do not advocate its size as the effectiveness measure.} \]

For empirical analysis, we use large well tested real-world projects from Github, and diverse tools. As in previous work [27], we compare the measurements between multiple tools, and identify the fraction of unique mutant set expected from the mutants produced by each tool. For mutation reduction, we investigated the reduction in effectiveness due to sampling. We find that even though the size of unique mutant set decreases along with decrease in sample size, the volume ratio remains similar. This suggests that the random samples produce mutants that are as hard to detect as the full set of mutants. Our data as well as the subject programs are available for replication [34].

**Contributions:**

- We identify the need for two different kinds of effectiveness measures for mutant sets. The first to measure the fraction of unique variants present (variation effectiveness), and the second to measure how good the given mutants are in emulating subtle bugs (measure for thoroughness).
- We show that the size of disjoint mutant set can not be used as a variation effectiveness measure due to two unstated assumptions — the single variant assumption, and the large test suite assumption.
- We provide an alternative for size of disjoint mutant set \( (|M_d|) \) — the size of unique mutant set \( (|M_u|) \), which is not vulnerable to the single variant assumption or large test suite assumption.
- We also show that disjoint mutant set is not the best set of most hard to detect variants, as it may miss variants that are equally hard.
- We develop a theory of geometric representation of variants and use it to provide an alternative measure for disjoint mutant set — the surface mutant set, and also provide the semantic interpretation through volume ratio \( \psi \).
- We provide an empirical benchmark for the different measures using three different mutation tools, and a diverse set of real world projects.

The organization of this paper is as follows. Section II discusses our theoretical model. Section III details the methodology used for selecting tools and subjects for benchmark. Section IV analyzes the results, followed by a detailed discussion in Section V. Related works are given in Section VI. As in the previous publication by Ammann et al. [27], we do not provide a threats to validity section since the focus is on theory. The conclusion is given in Section VII.

**II. Geometric Model**

Our approach builds on the disjoint mutant set [31] formalized by Amman et al. [27] but extends it to provide fine-grained criteria for evaluating performance of different mutation techniques. We note that our formulation is very similar to the recently proposed theoretical model for mutation testing methods [35].

**A. Terminology**

Given a program \( P \), and its test suite \( T \), a mutation tool may generate a set of mutants \( M \) by injecting a set of mutations. The mutations are a subset of all faults that are possible for a given program, which is generated by a mutation

---

\(^4\) It is still vulnerable to a limit of \( 2^{|T|} \), where \( |T| \) is the size of all tests, is usually larger than the number of mutants. Given that \( 2^{|T|} \) grows much faster than \( |T| \), with about 10 test cases sufficient for uniquely identifying 1,024 variants, we do not consider this a practical problem.
tool. We use lowercase for elements, while sets are represented by uppercase — $m \in M$ is a single mutant, and $t \in T$ is a single test case. The set $T^U$ represents all possible test cases for $P$. That is, $T \subseteq T^U$. The powerset of $T$ is represented by $2^T$, and contains $2^{|T|}$ elements. The mutants in $M$ killed by a test suite $T$ are given by $\text{kill}(T, M)$. Similarly, the tests in $T$ that kill $M$ are given by $\text{cover}(T, M)$.

$$\text{kill} : T \times M \rightarrow M$$
$$\text{cover} : T \times M \rightarrow T$$

A test suite $T$ is mutation adequate for $M$ if $\text{kill}(T, M) = M$. Two tests $t_1$ and $t_2$ are indistinguishable or non unique if they kill exactly the same set of mutants. That is, $\text{kill}({t_1}, M) = \text{kill}({t_2}, M)$. They are called distinguished or unique otherwise.

The same applies to mutants. Two mutants $m_1$ and $m_2$ are distinguished or unique if $\text{cover}(T, \{m_1\}) \neq \text{cover}(T, \{m_2\})$, and indistinguishable or non unique otherwise. A set of mutants is called unique mutant set if no pairs within the set are indistinguishable.

A mutant $m_1$ dynamically subsumes another $m_2$ if the tests that kill $m_1$ are guaranteed to kill $m_2$ (and $m_1$ can be killed). That is, $\text{cover}(T, \{m_1\}) \subseteq \text{cover}(T, \{m_2\})$

A set of tests are minimal if removal of any test case causes the mutation score to drop, and a disjoint mutant set corresponds to a minimal test set such that no mutant in the set dynamically subsumes another.

A runtime variant (or simply variant) is a program or a mutant that shows a deviation in runtime behavior from the original program $P$. Not all mutants express a detectable deviation and not all deviations are unique — equivalent, and redundant mutants exist. A variant may be detected by multiple test cases. We call a variant $v_a$ that is detected only by a subset of test cases that detect another variant $v_b$, a harder variant to detect than $v_b$ (and conversely easier). In terms of subsumption, the variant $v_a$ can be said to subsume variant $v_b$. That is, $v_a \geq v_b$. We consider only dynamic subsumption by available test cases here. A variant $v_a$ is included in the volume of a set of variants $V$ if and only if at least one variant $v_b \in V$ subsumes $v_a$.

B. Approach

Imagine that we have an exhaustive set of variants $V^U$ for any program $P$. Some variants may be easier to detect, and hence detected by multiple test cases. We know that mutants and variants have a surjective relationship. As shown in Figure 1, while multiple mutants can produce the same variant, multiple variants can not be associated with a single mutant, and there is at least one unique mutant for each variant generated (and at least one fault for any possible variant).

Consider an $n$ dimensional volume with the number of dimensions given by $|T|$. A unit distance along any dimension represents the variant escaping the corresponding test case, while 0 in each dimension represents a detection of the variant by the test case corresponding to that dimension. With this formulation, the origin point is the variant that is detected by all test cases, and a volume of $2^n$ is the maximal volume possible. The inclusion of a variant now has a geometric representation. It is within the volume bounded by the variants that are already present. Effectively, this means that the inclusion needs to be checked only with the surface variants of the enclosed volume.

Thus suggests a way to evaluate the effectiveness of a set of mutants given a test suite, which is given by the ratio between the possible variants that could be included by the detected variant surface corresponding to the mutant set and the maximal volume of variants $2^{|T|}$. Let us denote the volume ratio by $\psi$, and the mutants in the variant surface by $M_s$.

Volume ratio $\psi$ is a measure of thoroughness of a group of mutants. The more harder to detect, the larger the $\psi$ of the mutant set. $\psi$ is independent of the size of the project, or the size of test suite — all a volume ratio of 0.5 means is that of the possible variants, only 50% of the variants were included by the mutant set.

For example, consider Figure 2 where mutant $m_1$ is detected by test cases $\{t_1, t_2, t_3, t_4\}$, $m_2$ by $\{t_3, t_4, t_5\}$, and $m_3$ by $\{t_5\}$. This is a two dimensional representation of the five dimensional matrix. The central polygon represents the origin point $(0, 0, 0, 0, 0)$, representing a mutant that can be killed by all test cases. We say that $m_3$ subsumes $m_2$ (harder) but not $m_1$ (different), and the ratio of detected variants to total is $\frac{1}{5}$. The number of surface mutants here is $M_s = \{m_1, m_3\}$.

Unfortunately, measuring the $\psi$ score becomes difficult as the number of test cases and mutants increases. That is, for computing $\psi$, the numerator is given by

$$\left| \bigcup_{i=1}^{M} A_i \right| = \sum_{J \subseteq \{1, 2, \ldots, |M|\} \setminus \emptyset} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|$$

where the number of terms is $2^{|M|} - 1$ — increasing exponentially. Computation of such a value is infeasible.

For example, if we have a set of test cases $\{a, b, c\}$, and a set of distinguishable mutants $\{A, B, C\}$ each killed by some combination of given tests, the total volume of mutants covered by our set of mutants $\{A, B, C\}$ would be

$$\psi = \frac{2^{|A|} + 2^{|B|} + 2^{|C|} - 2^{|A \cap B|} - 2^{|A \cap C|} - 2^{|B \cap C|} + 2^{|A \cap B \cap C|}}{2^{|\{a, b, c\}|}}$$

Note that not all points in the volume may have corresponding mutants, such as due to test cases that check an exact subset of specification from another test case. We ignore these in favor of a simpler model.
For almost all test cases, but none are subsumed by others. Hence, another useful measure is the surface correction, which indicates whether the selected mutants are trivial to detect.

### C. Analysis

In this section, we compare the runtime complexity of three measures: disjoint mutant set $M_d$, surface mutant set $M_s$ and unique mutant set $M_u$.

Finding the true minimum test suite for a set of mutants is NP-complete. The best possible approximation algorithm is Chvatal’s ([37]), using a greedy algorithm where each iteration tries to choose a set that covers the largest number of mutants. This is given in Algorithm 1, and achieves an approximation ratio of $H(|M|)^7$. The complexity of greedy set-cover approximation is $O(\sum_{m \in M} |m|)$ [38] where $M_t$ is the family of subsets of $T$. That is, $M_t$ is the set of tests killing each mutant $m \in M$. The bound can be simplified to $O(|T| \times |M|)$, where $|T|$ is the number of test cases and $|M|$ is the number of mutants.

**Algorithm 1 Finding the minimal test suite**

```c
function MinTest(Mutants, Tests)
    T ← Tests
    M ← kill(T, Mutants)
    T_{min} ← ∅
    while M ≠ ∅ do
        t ← random(max{|kill(t, M)|})
        M ← M \ kill(t, M)
        T_{min} ← T_{min} ∪ {t}
    end while
    return T_{min}
```

Adding a new mutant to an existing set of non-subsumed mutants is to simply iterate through the existing set of non-subsumed mutants to see if the new mutant is dynamically subsumed, and if not add it to the set. The algorithm to remove subsumed mutants is given in Algorithm 2.

The only difference between computing disjoint mutant set and surface mutant set is in which test suite gets passed to this algorithm. If the test suite that gets passed is the minimal test set (an approximation of the minimum test set), computed in Algorithm 1, then the minimal mutant set returned is the disjoint mutant set $M_d$. On the other hand, if what gets passed is the full test suite, then the surface mutant set $M_s$ are computed.

The complexity of Algorithm 2, ignoring the growth of tests, is $O(m^2)$ where $m = |M|$. If we assume that the size of minimal test set is proportional to the size of the test set as shown in Figure 3 (consider also that in the worst case, a minimal (or minimum) test set is same as the full set), computing $M_d$ has at least $O(m^2)$ complexity (ignoring the computation of minimal test suite). This is the same as

**Algorithm 2 Finding the disjoint mutant set**

```c
function MinTest(Mutants, Tests)
    T ← Tests
    M ← kill(T, Mutants)
    T_{min} ← ∅
    while M ≠ ∅ do
        t ← random(max{|kill(t, M)|})
        M ← M \ kill(t, M)
        T_{min} ← T_{min} ∪ {t}
    end while
    return T_{min}
```

Adding a new mutant to an existing set of non-subsumed mutants is to simply iterate through the existing set of non-subsumed mutants to see if the new mutant is dynamically subsumed, and if not add it to the set. The algorithm to remove subsumed mutants is given in Algorithm 2.

While the volume ratio can provide information about the ratio of included variants, due to the massive redundancy of mutations generated, the ratio may be very close to 1. Hence, another useful measure is the surface correction $s$. Geometrically, it represents how close the given surface is to an $n$-sphere. A perfect spherical surface will have $s = 1$. For our purposes, it represents the mean number of test cases killing each surface mutant. The smaller the $s$, the more number of test cases that kill the surface mutants. A small $s$ is an indication that the surface mutants are easy to detect. Hence, they may be improved further to fail in a smaller number of test cases, perhaps by engineering them using higher order mutants. The $s$ can vary between $\frac{|M|}{|T|} = 1$ (each mutant fails only for a single test case) and $\frac{|M|}{|M| \times \left(\frac{|T|}{|T|-1}\right)} = \frac{1}{|T|-1}$ (each mutant fails for almost all test cases, but none are subsumed by others).

Note that this is different from disjoint mutant set from $M_d$. With disjoint mutant set, mutants $m_3$ and $m_1$ is sufficient to cover the entire set of test suite. With our formulation, the above grouping will not cover a mutant killed by $t_1, t_2, t_3$.

That is, we can expect $2^{\{|A,B,C\}|} - 1$ terms on the numerator. We can avoid this problem by using statistical approximation of the ratio. That is, randomly generate $N$ possible combinations of $|T|$ test kills. Let $N'$ be the number of these combinations that were included in the mutant surface $M_s$. The value $\frac{N'}{N}$ provides a reasonable approximation to $\psi$, refined to the requisite accuracy.

Each measure we have listed evaluates a different aspect of a given mutation set. The surface mutant set provides an alternative to the disjoint mutant set—the set of mutants that are shown to be hardest to detect out of the given set of mutants, and its effectiveness is given by volume ratio. The surface correction indicates whether the selected mutants are trivial to detect.
computing \( M_{r} \) because it uses the same algorithm and has the same worst case test suite. Finally, the algorithm to compute unique mutant set is given in Algorithm 3.

Algorithm 3 Finding the unique mutant set

function UNIQUE_MUTANTS(Tests, Mutants)
    \( M \leftarrow \text{sort}(\text{Mutants}) \) \quad \triangleright \text{Compare by cover}(\text{T}, \{m\})
    \( T \leftarrow \text{Tests} \)
    \( M_{r} \leftarrow \emptyset \)
    while \( M \neq \emptyset \) do
        \( m \leftarrow \text{pop}(M) \)
        \( M_{r} \leftarrow M_{r} \cup \{m\} \)
        \quad \quad \triangleright \text{Compare by cover}(\text{T}, \{m\})
    end while
    return \( M_{r} \)
end function

Of the three reduced mutation sets, the unique mutant set \( M_{d} \) is the easiest to compute because the maximum computational complexity is for sorting \( \bigO(m \times \log(m)) \).8

III. Methodology

Our assertion that the unique mutant set is a better effectiveness measure than the disjoint mutant set can only be shown through reasoning. Hence we do not attempt to validate it here.

In previous work [27], Ammann et al. compared the mutation score for various selective mutation score strategies using the disjoint mutant set. However, we know [25] that mutation reduction strategies are severely limited in how much effectiveness they can gain, in theory and practice. In order to judge the quality of a reduction strategy one should compare the effectiveness of the strategy to the effectiveness expected from random sampling. Our previous research showed that popular mutation reduction strategies do not fare well in such a comparison [39].

Hence, for our empirical analysis, we have two main concerns. The first one is to benchmark how the different measures perform for mutants from different tools. The second is to evaluate the effectiveness of random sampling. Computing how many of the mutants in the original set that belongs to any of \( M_{d} \), \( M_{s} \), \( M_{b} \), end up in the reduced random sample is not very useful. Because we are using random sampling, we are assured that the average ratio of sizes of these different sets to the size of full set \( M_{h} \) will remain the same due to the central limit theorem.

What we investigate instead is the second measure. That is, provided \( M_{d} \) is a measure of the effectiveness accounting for the ease of detection of mutants, our proposed replacement is \( M_{s} \), and its actual impact is computed by the volume ratio \( \psi \). Hence our question is how the effectiveness will change when we sample. Is the surface mutant set from the sample easier or harder to detect than the surface mutant set from the full set?

To benchmark new measures, it is important to capture the variations present in the real world use of these measures. The major avenues of variation are: variation due to mutant distribution in individual projects, variation due to the language used, and variation due to the mutation generation tools used (especially the phase during which the mutants were produced).

Unfortunately, the language choice is not orthogonal to other sources of variation. That is, language choice determines the projects, and the tool being used, which makes it difficult to compare different tools, and reflect variation introduced due to projects. Hence, we avoided variation due to language, and standardized on Java projects. Keeping the goal of real world projects that best represent real world software, we chose 25 large Java projects from Github [40] and Apache Software Foundation [41], that had large test suites. These projects, the size of their test suite (number of tests), and mutation scores are given in Table I.

We performed our evaluation with three tools: PIT 1.0,
TABLE I: Subject Programs, the size of test suite, and mutation scores

<table>
<thead>
<tr>
<th>Project</th>
<th>TestSuite</th>
<th>July</th>
<th>Major</th>
<th>PIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation-ch</td>
<td>126</td>
<td>42.420</td>
<td>43.275</td>
<td>59.375</td>
</tr>
<tr>
<td>asterisk-java</td>
<td>214</td>
<td>13.540</td>
<td>21.541</td>
<td>20.644</td>
</tr>
<tr>
<td>beamutils</td>
<td>1185</td>
<td>50.712</td>
<td>42.689</td>
<td>56.778</td>
</tr>
<tr>
<td>beamutils2</td>
<td>680</td>
<td>59.472</td>
<td>52.488</td>
<td>61.854</td>
</tr>
<tr>
<td>clazz</td>
<td>205</td>
<td>24.458</td>
<td>39.452</td>
<td>30.198</td>
</tr>
<tr>
<td>cli</td>
<td>373</td>
<td>71.174</td>
<td>76.606</td>
<td>86.137</td>
</tr>
<tr>
<td>collections</td>
<td>4407</td>
<td>76.998</td>
<td>58.631</td>
<td>34.687</td>
</tr>
<tr>
<td>commons-codec</td>
<td>605</td>
<td>92.725</td>
<td>73.524</td>
<td>82.661</td>
</tr>
<tr>
<td>commons-io</td>
<td>964</td>
<td>88.385</td>
<td>70.650</td>
<td>77.342</td>
</tr>
<tr>
<td>config-magic</td>
<td>111</td>
<td>55.191</td>
<td>29.800</td>
<td>60.690</td>
</tr>
<tr>
<td>csv</td>
<td>173</td>
<td>53.010</td>
<td>68.077</td>
<td>79.683</td>
</tr>
<tr>
<td>dbutils</td>
<td>239</td>
<td>44.234</td>
<td>65.205</td>
<td>47.340</td>
</tr>
<tr>
<td>events</td>
<td>206</td>
<td>77.136</td>
<td>70.030</td>
<td>59.949</td>
</tr>
<tr>
<td>faunas</td>
<td>172</td>
<td>2.552</td>
<td>58.653</td>
<td>49.066</td>
</tr>
<tr>
<td>java-api-wrapper</td>
<td>125</td>
<td>14.992</td>
<td>84.912</td>
<td>76.035</td>
</tr>
<tr>
<td>java-classmate</td>
<td>219</td>
<td>66.167</td>
<td>77.230</td>
<td>90.257</td>
</tr>
<tr>
<td>jept-simple</td>
<td>566</td>
<td>84.500</td>
<td>79.315</td>
<td>44.949</td>
</tr>
<tr>
<td>mgwt</td>
<td>103</td>
<td>40.722</td>
<td>6.612</td>
<td>8.853</td>
</tr>
<tr>
<td>mirror</td>
<td>303</td>
<td>58.734</td>
<td>74.725</td>
<td>75.472</td>
</tr>
<tr>
<td>naptr</td>
<td>206</td>
<td>72.465</td>
<td>51.697</td>
<td>54.507</td>
</tr>
<tr>
<td>ognl</td>
<td>113</td>
<td>13.962</td>
<td>6.463</td>
<td>56.324</td>
</tr>
<tr>
<td>pipes</td>
<td>138</td>
<td>65.994</td>
<td>62.637</td>
<td>67.662</td>
</tr>
<tr>
<td>primitives</td>
<td>227</td>
<td>77.530</td>
<td>71.326</td>
<td>90.257</td>
</tr>
<tr>
<td>validator</td>
<td>382</td>
<td>50.266</td>
<td>39.061</td>
<td>68.208</td>
</tr>
<tr>
<td>webbit</td>
<td>146</td>
<td>73.949</td>
<td>67.172</td>
<td>52.407</td>
</tr>
</tbody>
</table>

Judy 2.1.x, and Major 1.1.5. For each tool, we used the settings for the maximum number of operators to mutate.

Unlike other structural coverage measures such as statement, branch or path coverage, there is very little agreement on what constitutes an acceptable set of mutants in mutation analysis. This means that we can expect a wide variation in the number of mutants produced. The mutants produced by each tool on each program is given in Table II. Unfortunately, this also means that the mutation scores do not necessarily agree as we see in Table I. One of the culprits is the presence of equivalent mutants — mutants that do not produce a measurable semantic variation to the original program. To avoid skewing the results due to the presence of equivalent mutants, we removed the mutants that were not killed by any of the test cases we had. Note that some of the projects produced very small number of mutants, ignoring classes that could not be mutated due to various reasons. For the sake of completeness, and to show that these does not impact our analysis. This means that we can expect a wide variation in what constitutes an acceptable set of mutants in mutation analysis.

A. Measures for the full set of mutants

The different measures for the full set of mutants are given in Table VI. As can be seen, except in a few cases, the volume being covered is indeed close to the full volume. However, we see that the surface correction $s$ given in Table VI suggests that there is still quite a bit of improvement possible. The number of mutants necessary by variant surfaces $M_s$, along with disjoint mutant set $M_d$ the size of unique mutant set $M_k$, and the total number of mutants detected $M_k$ are also given in Table VI. As expected, the surface mutant set $M_s$ is larger than the disjoint mutant set $M_d$, and the unique mutant set $M_k$ is the largest out of the total detected $M_k$.

B. Measures for 100 mutants sampled

The mean measures for 100 mutants sampled randomly 100 times from each tool, for each project is given in Table VII. The columns with labels $\psi$ and $s$ are measures from the full set, and are given for comparison. $\psi^d$, $s^d$, and $M^d_k$ are measures computed from the sample.

C. Comparing complete and sampled mutants

As we indicated previously, we know the behavior of $M_k$ under random sampling. Once we label some of mutants in $M_k$ as belonging to $M_k$, taking random samples from $M_k$ can be expected to preserve the ratio $\frac{M_k}{M_k}$ on average. Hence, if we are using the size of unique mutant set as an effectiveness measure, the effectiveness will decrease in inverse proportion to the size of the sample. That is, a sample with $\frac{1}{10}$ of the original unique variants.

One measure that is actually of interest is the volume ratio $\psi$. We evaluate the change in volume ratio using the linear regression:

$$\psi_{\text{original}} = \beta_\psi \times \psi_{\text{sample}}$$

Similarly, we evaluate the change in surface correction using the linear regression:

$$s_{\text{original}} = \beta_s \times s_{\text{sample}}$$


TABLE II: Subject programs, their size in LOC and mutants produced by each tool

<table>
<thead>
<tr>
<th>Project</th>
<th>LOC</th>
<th>July</th>
<th>Major</th>
<th>PIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation-ch</td>
<td>870</td>
<td>777</td>
<td>512</td>
<td>981</td>
</tr>
<tr>
<td>asterisk-java</td>
<td>29477</td>
<td>12685</td>
<td>5812</td>
<td>15476</td>
</tr>
<tr>
<td>beamutils</td>
<td>11640</td>
<td>6529</td>
<td>4382</td>
<td>9685</td>
</tr>
<tr>
<td>beamutils2</td>
<td>2551</td>
<td>990</td>
<td>615</td>
<td>2069</td>
</tr>
<tr>
<td>clazz</td>
<td>5681</td>
<td>2784</td>
<td>2022</td>
<td>5165</td>
</tr>
<tr>
<td>cli</td>
<td>2667</td>
<td>2308</td>
<td>1441</td>
<td>2677</td>
</tr>
<tr>
<td>collections</td>
<td>25400</td>
<td>1006</td>
<td>10301</td>
<td>24141</td>
</tr>
<tr>
<td>commons-codec</td>
<td>6603</td>
<td>44</td>
<td>7362</td>
<td>9093</td>
</tr>
<tr>
<td>commons-io</td>
<td>9472</td>
<td>164</td>
<td>6486</td>
<td>9799</td>
</tr>
<tr>
<td>config-magic</td>
<td>1251</td>
<td>527</td>
<td>650</td>
<td>1181</td>
</tr>
<tr>
<td>csv</td>
<td>1384</td>
<td>1154</td>
<td>991</td>
<td>1798</td>
</tr>
<tr>
<td>dbutils</td>
<td>2596</td>
<td>1159</td>
<td>677</td>
<td>1022</td>
</tr>
<tr>
<td>events</td>
<td>1256</td>
<td>2353</td>
<td>615</td>
<td>1155</td>
</tr>
<tr>
<td>faunas</td>
<td>9000</td>
<td>3723</td>
<td>3771</td>
<td>9688</td>
</tr>
<tr>
<td>java-api-wrapper</td>
<td>1760</td>
<td>929</td>
<td>611</td>
<td>1711</td>
</tr>
<tr>
<td>java-classmate</td>
<td>2402</td>
<td>1423</td>
<td>952</td>
<td>2543</td>
</tr>
<tr>
<td>jept-simple</td>
<td>1617</td>
<td>497</td>
<td>695</td>
<td>1790</td>
</tr>
<tr>
<td>mgwt</td>
<td>16250</td>
<td>1394</td>
<td>6564</td>
<td>12030</td>
</tr>
<tr>
<td>mirror</td>
<td>2590</td>
<td>1316</td>
<td>449</td>
<td>1876</td>
</tr>
<tr>
<td>mp3agic</td>
<td>1277</td>
<td>1277</td>
<td>4822</td>
<td>7182</td>
</tr>
<tr>
<td>ognl</td>
<td>13139</td>
<td>8243</td>
<td>5616</td>
<td>21227</td>
</tr>
<tr>
<td>pipes</td>
<td>3513</td>
<td>590</td>
<td>1171</td>
<td>3001</td>
</tr>
<tr>
<td>primitives</td>
<td>11965</td>
<td>14</td>
<td>695</td>
<td>1791</td>
</tr>
<tr>
<td>validator</td>
<td>5807</td>
<td>3220</td>
<td>3655</td>
<td>5846</td>
</tr>
<tr>
<td>webbit</td>
<td>5018</td>
<td>144</td>
<td>1327</td>
<td>3707</td>
</tr>
</tbody>
</table>
Comparing the size of $M_d$ produced by each tool (Figure 4), we find that PIT produced on average 0.224 unique variants per mutant produced (standard deviation = 0.084). In comparison, Major produced on average 0.334 unique variants per mutant produced (standard deviation = 0.072) and Judy produced 0.306 unique variants per mutant produced (standard deviation = 0.195). We note that comparing the size of disjoint mutant set, PIT produced on average 0.065 disjoint mutants per mutant produced (standard deviation = 0.03). In comparison, Major produced on average 0.129 disjoint mutants per mutant produced (standard deviation = 0.033) and Judy produced 0.116 disjoint mutants per mutant produced (standard deviation = 0.121).

These results suggest an advantage for mutants produced by Major, closely followed by Judy as suggested in Figure 4. Given that $|M_d|$ is the size of variants in any given set of mutants, and $R_d$ is an effectiveness measure, it is worthwhile to estimate this effectiveness quickly for any set of mutants corresponding to a program generated by any tool. However, computing $|M_d|$ requires the full test run for the entire set of mutants. Is there a way to reliably estimate the size of $|M_d|$? Our results suggest that the ratio $R_d$ is highly correlated to the ratio of unique mutant set from sampling. That is,

$$ R_d = \beta_1 \times M_d^{sample} + \beta_2 \times Tool $$

The regression $R^2 = 0.832$, and $p < 0.001$. Since the tool is significant, considering each tool separately:

$$ R_d = \beta_1 \times M_d^{sample} $$

$R^2 = 0.924$, and $\beta_1 = 0.307$, Major $R^2 = 0.901$, and $\beta_1 = 0.483$, Judy $R^2 = 0.666$, and $\beta_1 = 0.513 (p < 0.0001)$. That is, we have moderate to strong correlation between $R_d$ and $M_d^{sample}$, which may be used to predict $R_d$ from $M_d^{sample}$. However, further research is needed in this area to determine
the effects of sample size, project characteristics, and mutation tool used.

Table VI suggests that for most of our projects $\psi$ is very close to 1. That is, the volume enclosed by mutants is very close to the limit possible. This may be because most test cases are very targeted unit test cases that kill mutants within a given function. That is, for most mutants which are judged by a test suite, there is a small number of tests that detect each mutant. We also find that when we sample just 100 mutants (Table VII), the volume ratio remains close to the original volume ratio. That is, the coefficient $\beta_\psi$ for PIT is 0.997, for Major is 0.996, and for Judy is 0.99, with $R^2 > 0.99$ and $p < 0.0001$. Further, the standard deviation of $\psi$ suggests that there is very little deviation in any of the samples. A similar observation can be made for the surface correction $s$. The coefficient $\beta_s$ for $s_{sample}$ for PIT is 0.872, for Major is 0.875, and for Judy is 0.9, with $R^2 > 0.97$ and $p < 0.0001$. That is, the surface correction for samples is close to the surface correction detected for the full set.

Previous research [27] showed that using size of disjoint mutant set as a measure for effectiveness accounting for ease of detection, mutation reduction strategies fare poorly when it comes to maintaining effectiveness. Our results here show that even though the size of surface mutant set varies, the volume ratio $\psi$ of samples remains close to the original volume ratio $\psi$ of the full set of mutants, and the same case is true for the finer surface correction $s$. This result suggests that even if the size of surface mutant set or disjoint mutant set varies, the effectiveness accounting for ease of detection of the mutant set does not vary as much for random sampling. Does this effect extend to more intelligent mutation reduction strategies? Our previous research [25], [39] suggests that intelligent mutation strategies often fare poorly when compared against random sampling, and there is a need for further research in this area.

VI. Related Work

The idea of mutation analysis was first proposed by Lipton [1], and its main concepts were formalized by DeMillo et al. in the “Hints” [42] paper. The first implementation of mutation analysis was provided in the PhD thesis of Budd [43] in 1980. Previous research in mutation analysis [44]–[46] suggests that it subsumes different coverage measures, including statement, branch, and all-defs dataflow coverage [44]–[46]. There is also some evidence that the faults produced by mutation analysis are similar to real faults in terms of error trace produced [3] and the ease of detection [4], [5]. Recent research by Just et al. [7] using 357 real bugs suggests that the mutation score increases with test effectiveness for 75% of the cases, which was better than the 46% reported for structural coverage.

The validity of mutation analysis rests upon two fundamental assumptions: “The competent programmer hypothesis” – which states that programmers tend to make simple mistakes, and “The coupling effect” – which states that test cases capable of detecting faults in isolation continue to be effective even when faults appear in combination with other faults [42]. Evidence of the coupling effect comes from theoretical analysis by Wah [47], [48], empirical studies by Offutt [49], [50] and Langdon et al. [51]. The competent programmer hypothesis was quantified in our previous work [52].

One theoretical (and practical) difficulty in mutation analysis is identifying equivalent mutants — mutants that are syntactically different, but semantically indistinguishable from the original program, leading to incorrect mutation scores, because in general, identifying equivalent mutants is undecidable. The work on identifying equivalent mutants is generally divided into categories of prevention and detection [17], with prevention focusing on reducing the incidence of equivalent mutants [53] and detection focusing on identifying the equivalent mutants by examining their static and dynamic properties. These include efforts to identify them using compiler equivalence [17], [54], [55] dynamic analysis of constraint violations [56], [57], and coverage [58].

A similar problem is that of redundant mutants [8], resulting in a misleading mutation score. A number of studies measured the redundancy among mutants. Ammann et al. [27] compared the behavior of each mutant under all tests and found numerous redundant mutants. More recently, Papadakis et al. [17] used the compiled representation of programs to identify equivalent mutants. They found that on average 7% of mutants are equivalent while 20% are redundant.

Another fruitful area of research has been reducing the cost of mutation analysis, broadly categorized as do smarter, do faster, and do fewer by Offutt et al. [59]. The do smarter approaches include space-time trade-offs, weak mutation analysis, and parallelization of mutation analysis. The do faster approaches include mutant schema generation, code patching, and other methods to make the mutation analysis faster as
a whole. Finally, the *do fewer* approaches try to reduce the number of mutants examined, and include selective mutation and mutant sampling.

Various studies have tried to tackle the problem of approximating the full mutation score without running a full mutation analysis. The idea of using only a subset of mutants (*do fewer*) was conceived first by Budd [44] and Acree [60] who showed that using just 10% of the mutants was sufficient to achieve 99% accuracy of prediction for the final mutation score. This idea was further investigated by Mathur [9], Wong et al. [11], [61], and Offutt et al. [10] using the Mothra [62] mutant selection techniques for FORTRAN. Lu Zhang et al. [63] compared operator-based mutant selection techniques to random mutant sampling, and found that random sampling performs as well as the operator selection methods. Recently, it was found [64] that 9, 604 mutants were sufficient for obtaining 1% accuracy for 99% of the projects, irrespective of the independence of mutants, or their total population.

Researchers have evaluated different mutation tools in the past [65], based on fault model (operators used), order (syntactic complexity of mutations), and selectivity (eliminating most frequent operators), mutation strength (weak, firm, and strong), and the sophistication of the tool in evaluating mutants. Our evaluation differs from their research in focusing on the semantic impact of mutants produced by different tools. Another closely related publication is MuRanker [66] where the authors use various distance functions to rank different mutants. While that study is similar to ours, the focus in that paper is on differentiating mutants by measures other than test failures (such as coverage), and in ranking mutants, not in coming up with a measure for a set of mutants. We note that our formulation of variants as situated in a hypergeometric distribution provides a measure for a set of mutants. We propose a benchmark of the different measures. We provided a benchmark of the different measures [27] recently as an alternative that does not require either the single variant assumption, or the large test suite assumption. For the second criteria, we proposed unique mutant set as an alternative that does not require the single variant assumption, or the large test suite assumption. For the second criteria, we proposed the surface mutant set which preserves useful hard to detect mutants, and a semantics using volume ratio and surface correction that provides a more concrete explanation of what ease of detection entails.

It may be asked, why try to mitigate problems with mutant analysis by relying again on mutation detection by test suites? Why not use other criteria such as coverage information or program metrics instead?

Detection of mutants by test suites is still the closest, most effective, and most direct approach we have for evaluating the effectiveness of test suites. Other measures (such as most kinds of coverage) are subsumed by mutation analysis, and measures such as static information from program metrics are not suitable as a theoretical bound on the effectiveness of mutants, because they are dependent on particular languages. Furthermore, we are unsure about the information they provide about the variants, primarily because such information is highly indirect compared to mutant kills.

We provide a benchmark of the different measures $|M_d|$, $|M_s|$, and $|M_{\bar{s}}|$ using three different tools (PIT, Judy, and Major) on 25 real world projects. Our empirical analysis shows that random sampling works reasonably well for maintaining the effectiveness (accounting for ease of detection) of a given set of mutants.

### VII. Conclusion

Research on newer mutation operators, and on eliminating redundant mutants requires some measure of effectiveness for a given set of mutants. As the primary aim of mutation analysis is to evaluate the quality of test suites, the measure of effectiveness should be based on how best to achieve this.

For a given test suite, there are two main concerns. Does it detect a reasonable fraction of the deviations in the program specification? Secondly, does it detect and prevent subtle bugs?

To enable these measurements, a set of mutants should be able to provide as large a set of unique variants as possible, and also provide variants that are hard to detect.

Size of disjoint mutant set have been proposed [27] recently as a measure of effectiveness of mutation reduction techniques. However, a problem with this measure is that it is too tightly coupled with the test suite.

Most reduction strategies and mutation operators use static properties of the program in question to generate mutants. The aim of these strategies is to identify variants that are actually unique, not just to maintain the quality of a particular test suite. The test suite is only incidental to the measurement.

We showed that if we are to use disjoint mutant set as the set of all unique variants in a set of mutants, we need two more assumptions: the single variant assumption (that each test case kills exactly one variant), and the large test suite assumption (that the number of test cases is larger than the number of unique variants) beyond the stated assumptions of a comprehensive test suite, and a fixed set of mutants. Given that test cases routinely have more than a single assertion [67], and the number of mutants is usually much larger than the test suite size, these two assumptions may not be justified for real world test suites. On the other hand, if we were to use disjoint mutant set as the set of hardest to find mutants, we show that disjoint mutant set can discard some of important mutants that are as hard to detect as any other mutants. Hence disjoint mutant set is not an adequate solution as a measure of effectiveness of a mutant set.

Our contribution in this paper is to recognize the two different effectiveness criteria required. The first is the number of unique variants identified from a given set of mutants, and the second is a measure of ease of detection of a given set of mutants.

We proposed unique mutant set as an alternative that does not require either the single variant assumption, or the large test suite assumption. For the second criteria, we proposed the surface mutant set which preserves useful hard to detect mutants, and a semantics using volume ratio and surface correction that provides a more concrete explanation of what ease of detection entails.

**References**


