Redundancy, The Mutants’ Elixir of Immortality

Anonymous Author(s)

1 MUTANT EQUIVALENCE

Mutation is the art of generating syntactic variations of a program \( P \), and is meaningful only to the extent that the syntactic modifications applied to \( P \) yield semantic differences; but in reality a mutant \( M \) may be syntactically distinct from the base program \( P \) yet still compute the exact same function as \( P \). The existence, and perversiveness, of equivalent mutants is a source of bias and uncertainty in mutation-based analysis:

- If we generate 100 mutants of program \( P \) and find that some test data set \( T \) kills 80 of them, what can we infer about \( T \) depends on the number of equivalent mutants.
- If test data set \( T \) kills 80 mutants of \( P \), it is important to know whether \( T \) has killed 80 distinct mutants, or has just killed 80 times the same mutant, or something in between?

The issue of mutant equivalence has been the focus of much research recently [1–3, 5–16, 18, 20, 21]. It is beyond the scope of this paper to do a survey of mutation equivalence research (see [17] for a recent survey). But if we may indulge in a broad generalization, we would say that most research on mutation equivalence falls into one of two broad categories:

- Research that infers equivalence from a local analysis of the mutation site; such methods are prone to loss of recall, as two programs may be distinctly local but still equivalent.
- Research that infers equivalence from comparing global behavioral envelopes; such methods are prone to loss of precision, because programs may have similar behavioral envelopes yet still be semantically distinct.

Determining whether two syntactically distinct programs are semantically equivalent is known to be undecidable [4]. Notwithstanding this theoretical result, attempting to determine whether two programs are semantically equivalent is a non-trivial, costly, and error-prone exercise; attempting to decide whether \( M \) mutants are equivalent to a base program is \( M \) times more difficult/impractical; attempting to decide whether \( M \) mutants are distinct from each other is \( M^2 \) times more difficult/impractical. Most importantly, we find that for most practical applications, it is not necessary to identify equivalent mutants individually; it suffices to estimate their number.

In this paper we argue that research on mutation equivalence ought to turn away from efforts to determine equivalence between programs, and focus instead on estimating the probability that two programs (e.g., a base program and a mutant, or two mutants) are semantically equivalent. We find that estimating this probability can be relatively easy and inexpensive, yet enables us to answer many important questions.

2 REM: THE RATIO OF EQUIVALENT MUTANTS

In [21] Yao et al. ask the question: what are the causes of mutant equivalence? Mutant equivalence is determined by two factors, namely the mutant operators and the program being mutated. For the sake of argument, we consider a fixed mutation policy (defined by a set of mutant operators) and we reformulate Yao’s question as: What attribute of a program makes it prone to generate equivalent mutants? A program is prone to generate equivalent mutants if it continues to deliver the same function despite the presence of mutations. Given that mutations can be seen as instances of faults [11], we can formulate this attribute as: a program is prone to generate equivalent mutants if it continues to deliver the same function despite the presence of faults. This attribute has a name: fault tolerance! We know what makes programs fault tolerant: redundancy.

Hence if only we could quantify the redundancy of a program, we can use it to estimate the number of equivalent mutants that a program is prone to generate. Specifically, we introduce the concept of Ratio of Equivalent Mutants (REM, for short) of a program \( P \) as the ratio of equivalent mutants over the total number of mutants generated for \( P \) by a given mutation policy. We introduce the following redundancy metrics, all defined by means of Shannon’s entropy [19]:

- **State Redundancy**, which reflects the gap between the declared state space of the program and its actual (used) state space; it is denoted by \( SR_I \) for the initial state and \( SR_F \) for the final state.
- **Functional Redundancy**, which reflects the gap between the entropy of the input space and that of the output space; it is denoted by \( FR \).
- **Non-Injectivity**, which reflects the amount of redundancy that the program creates between its state variables as it executes; it is denoted by \( NI \).
- **Non-Determinacy**, which reflects the amount of non-determinacy of the oracle that defines semantic equivalence between the
outcomes of two programs (a base program and a mutant, or two mutants); it is denoted by \( ND \).

We find (in work which, for the sake of anonymity, we do not cite) that \( REM \) is statistically correlated to \( SR_I, SR_F, FR, NI, \) and \( ND \); we have also derived regression formulas for \( REM \) as a function of the redundancy metrics. This statistical analysis is carried out using a fixed set of mutation operators, and is meaningful only for that set of operators.

Given a program \( P \) and a set of \( M \) mutants derived by the proposed mutation policy, we estimate the number of equivalent mutants as: \( REM \times M \). Then the performance of a test data set \( T \) must be judged against the estimated number of non-equivalent mutants, \( N = (1 - REM) \times M \) rather than against \( M \).

3 MUTATION REDUNDANCY

We consider a program \( P \) whose ratio of equivalent mutants is \( REM \) and we consider a test data set \( T \) and let \( N \) be the number of mutants that \( T \) kills. We cannot tell how good set \( T \) is unless we know how many distinct mutants the set of killed mutants contains. What really measures the effectiveness of \( T \) is not \( N \), but rather the number of equivalence classes of the set of mutants killed; the question that we must address then is, **how do we estimate the number of equivalence classes in a set of \( N \) mutants of \( P \)?**

Because the mutants differ only slightly from the base program (minor syntactic alterations) it is fair to assume that they have the same amount of redundancy as \( P \), hence they also have the same \( REM \). Hence we can interpret \( REM \) as the probability that any two mutants are equivalent to each other. Then the question that we must address is: **Given a set of \( N \) elements, where any two have a probability \( REM \) of being equivalent, what is the expected number of equivalence classes?** We denote this number by \( NEC(N, REM) \), and we write it as:

\[
NEC(N, REM) = \sum_{k=1}^{N} k \times p(N, REM, k),
\]

where \( p(N, REM, k) \) is the probability that the number of equivalence classes is \( k \). This probability can be estimated by the following inductive formulas:

\[
p(N, REM, 1) = REM^{N-1}, \quad p(N, REM, N) = (1 - REM)^{-1}.
\]

\[
p(N, REM, k) = (1 - (1 - REM)^k) \times p(N - 1, REM, k) + (1 - REM)^{k-1} \times p(N - 1, REM, k - 1),
\]

to account for, respectively, the case where there is a single equivalence class, the case where there are \( N \) equivalence classes, and the case where the number of equivalence classes is between 1 and \( N \) exclusive (the formula in this case reflects, respectively, the situation when the \( N^{th} \) element falls in an existing equivalence class, and the situation when the \( N^{th} \) element defines a new class).

Now that we have an explicit formula for the number of equivalence classes in a set of \( N \) mutants, we argue that it is this number \( NEC(N, REM) \), rather than \( N \), that truly measures the effectiveness of \( T \). With this in mind, it is worthwhile to consider two important properties of \( NEC(N, REM) \):

- This function depends a great deal more on \( REM \) than it depends on \( N \); hence we cannot get a sense for the value of \( NEC(N, REM) \) until we have determined \( REM \).
- For typical values of \( REM \) (in the neighborhood of 0.05 to 0.15), \( NEC(N, REM) \) is much smaller than \( N \); in particular, we could kill thousands of mutants, only to realize that actually we have killed only a few dozen distinct mutants.

As a consequence of these observations, it is clear that when a test set \( T \) kills a large number \( N \) of mutants of some program \( P \), we should not rush into celebrating prematurely, until we have computed \( NEC(N, REM) \); this, in turn, requires that we determine the \( REM \) of the program \( P \). For the sake of illustration, we show in the table below the value of \( NEC(N, REM) \) for some sample values of \( N \) and \( REM \).

<table>
<thead>
<tr>
<th>( REM )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>( p(3000, 0.05, k) )</td>
<td>( p(3000, 0.05, k) )</td>
<td>( p(3000, 0.05, k) )</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>99</td>
<td>0.126499</td>
</tr>
<tr>
<td>14</td>
<td>4.94066e-324</td>
<td>110</td>
<td>0.000200261</td>
</tr>
<tr>
<td>87</td>
<td>8.35248e-05</td>
<td>111</td>
<td>6.43533e-05</td>
</tr>
<tr>
<td>88</td>
<td>0.000296773</td>
<td>240</td>
<td>8.67925e-320</td>
</tr>
<tr>
<td>97</td>
<td>0.114522</td>
<td>241</td>
<td>0</td>
</tr>
<tr>
<td>98</td>
<td>0.126799</td>
<td>3000</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( REM = 0.05 \):

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p(3000, 0.05, k) )</th>
<th>( k )</th>
<th>( p(3000, 0.05, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>56</td>
<td>0.158642</td>
</tr>
<tr>
<td>7</td>
<td>4.94066e-324</td>
<td>63</td>
<td>0.000289045</td>
</tr>
<tr>
<td>47</td>
<td>0.00014182</td>
<td>64</td>
<td>5.79254e-05</td>
</tr>
<tr>
<td>54</td>
<td>0.169991</td>
<td>157</td>
<td>8.04833e-321</td>
</tr>
<tr>
<td>55</td>
<td>0.182252</td>
<td>158</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( REM = 0.10 \):

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p(3000, 0.05, k) )</th>
<th>( k )</th>
<th>( p(3000, 0.05, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>56</td>
<td>0.158642</td>
</tr>
<tr>
<td>7</td>
<td>4.94066e-324</td>
<td>63</td>
<td>0.000289045</td>
</tr>
<tr>
<td>47</td>
<td>0.00014182</td>
<td>64</td>
<td>5.79254e-05</td>
</tr>
<tr>
<td>54</td>
<td>0.169991</td>
<td>157</td>
<td>8.04833e-321</td>
</tr>
<tr>
<td>55</td>
<td>0.182252</td>
<td>158</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( REM = 0.15 \):
4 MUTATION SCORE

When we run \( M \) mutants of program \( P \) on some test data set \( T \) and we find that \( X \) mutants are killed, it is customary to view the ratio \( MS(T) = \frac{X}{M} \) as a measure of effectiveness of \( T \), called the mutation score of \( T \). We argue that this formula suffers from two major flaws:

- The denominator ought to reflect the fact that some of the \( M \) mutants are equivalent to \( P \), hence no test data can kill them.
- Both the numerator and the denominator ought to be quantified not in terms of the number of mutants, but instead in terms of the number of distinct mutants; a test data set cannot be credited for killing the same mutant repeatedly.

To address these shortcomings, we propose the following definition.

**Definition 4.1.** Given a program \( P \) and a set of \( M \) mutants thereof, of which \( N \) mutants are not equivalent to \( P \), and given a test data set \( T \). If execution of the \( M \) mutants on \( T \) causes \( X \) mutants to be killed, then the essential mutation score of \( T \) is denoted by \( EMS(N, X) \)

and defined as the ratio of distinct mutants in \( X \) over the total number of distinct mutants in \( N \).

The denominator of \( EMS \) is already known, viz \( NEC(N, REM) \).

To compute the numerator, we introduce the following function: \( COV(N, K, X) \), for a set of size \( N \) partitioned into \( K \) classes, and a subset thereof of size \( X \), is the (expected) number of equivalence classes that overlap with set \( X \) for the sake of simplicity, we may refer to a set and its cardinality by the same symbol; this function is called \( COV() \), for coverage.

To compute \( COV(N, K, X) \), we let \( C_1, C_2, \ldots C_K \) be the equivalence classes of set \( N \) and let \( f_i \) be the boolean function that takes value 1 if and only if \( X \) has a non-empty intersection with class \( C_i \). Then \( COV(N, K, X) \) can be written as the expected value of the following random variable: \( \sum_{i=1}^{K} f_i \). If the events "set \( X \) overlaps with class \( C_i \)" for \( 1 \leq i \leq K \), are independent (which can be assumed if \( N \) is very large), then we can write

\[
\sum_{i=1}^{K} f_i = K \times p(f_i = 1) = K \times (1 - p(f_i = 0)).
\]

As for \( p(f_i = 0) \), it can be computed as \( \left( \frac{K-1}{K} \right)^X \) since each element of \( X \) has probability \( \frac{K-1}{K} \) to be outside class \( C_1 \) (assuming all classes have the same size).

If we do not assume independence, and we still assume that all equivalence classes have the same size \( \left( \frac{N}{K} \right) \), then \( p(f_i = 0) \) can be evaluated as the product of the following probabilities, which represent the events: the first element of \( X \) is not in \( C_1 \); the second element of \( X \) is not in \( C_1 \); etc.: \( N - \frac{N}{K}, N - \frac{N}{K} - 1, N - \frac{N}{K} - 2, N - \frac{N}{K} - 3 \), etc. Combining these expressions, we find the following closed formula for \( COV(N, K, X) \):

\[
COV(N, K, X) = K \times \left( 1 - \frac{K-1}{K} \right)^X \times \prod_{i=2}^{X} \frac{N - i \times \frac{N}{K}}{N - i}.
\]

The following table shows sample values for \( COV(N, K, X) \) for \( N = 75, K = 15 \) and \( X \) varying between 0 and 75; this quantity represents the number of distinct mutants that are estimated to be in \( X \), given that \( N \) is divided into 15 equivalence classes.

<table>
<thead>
<tr>
<th>( X )</th>
<th>( COV(75, 15, X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
</tr>
<tr>
<td>5</td>
<td>4.4814</td>
</tr>
<tr>
<td>10</td>
<td>7.8214</td>
</tr>
<tr>
<td>15</td>
<td>10.2534</td>
</tr>
<tr>
<td>20</td>
<td>11.9766</td>
</tr>
<tr>
<td>25</td>
<td>13.1586</td>
</tr>
<tr>
<td>30</td>
<td>13.9382</td>
</tr>
<tr>
<td>35</td>
<td>14.4281</td>
</tr>
</tbody>
</table>

5 MINIMAL MUTANT SETS

If we learn anything from section 3, it is that the number of distinct mutants in a set of size \( N \) can be much smaller than \( N \), even for very small values of \( REM \). This raises the question: how can we identify a minimal set of distinct mutants that includes no equivalent mutants, yet misses no equivalence class? In other words, given a set of size \( N \) partitioned by an equivalence relation, we want to select one (and only one, for the sake of minimality) element from each class.

If we do not know how many equivalence classes the set \( N \) has, then we have to compare each element of \( N \) to each other element, an \( O(N^2) \) operation. But if we know how many equivalence classes the set has, which we do, thanks to \( NEC(N, REM) \), then we can pick elements of \( N \) in an arbitrary order and compare them to each other for equivalence until we find \( NEC(N, REM) \) non-equivalent elements. Whereas the brute-force algorithm is an \( O(N^2) \) algorithm, the proposed approach is an \( O(H^2) \) algorithm, where \( H \) is the estimated number of mutants we need to inspect to pick one element from each equivalence class.

To this effect, we introduce a new function, \( NOI(N, K) \) (NoI: Number Of Inspections), which represents the estimated number of elements of a set of size \( N \) partitioned into \( K \) equivalence classes, that we need to inspect to get at least one in each class. We let \( d_i \), for \( 1 \leq i \leq K \), be the number of additional draws needed to cover the \( i^{th} \) equivalence class, and we let \( D_K \) be defined as \( D_K = \sum i = 1K d_i \). Then our goal is to estimate the expected value of \( D_K \). If \( N \) is very large relative to \( K \), then the probability of covering a new equivalence class does not change with each draw; the estimate of \( D_K \) can be written as:

\[
E(D_K) = 1 + \frac{K}{K - 1} + \frac{K}{K - 2} + \frac{K}{K - 3} + \ldots + K,
\]

using expected values of geometric random variables. If we admit that probabilities do change with each draw, then we resort to a recursive formula where the probability of each \( d_i \) depends on the outcome of \( d_{i-1} \). Space limitations preclude us from presenting
details of this formula; we have a program written in R to compute this function.

As a simple illustration, consider the example discussed in section 3; for \(N = 3000\) and \(REM = 0.10\) we had found \(K = 55\). Whence we derive \(\text{NOI}(3000, 55) = 252.649\); in other words, it is estimated that if we draw one mutant at a time in a set of 3000 mutants, it takes about 253 draws before we encounter 55 distinct mutants.

So that by drawing mutants until we find 55 distinct elements we achieve a speedup of \(\frac{3000^2}{252^2} = 141.72\). Of course the speedup is very large because \(N\) is large but it is significant even for small values of \(N\), because \(\text{REM} = K / N\) is bigger than 1 by definition, hence its square is even bigger.

### 6 AUTOMATION

All the analysis presented in this paper depends critically on our ability to compute the \(REM\) of a program within a mutation generation experiment (i.e. for a given mutation policy); page limitation precludes us from a detailed discussion. We argue that for a given mutation policy, the \(REM\) of a program can be estimated by analyzing the amount of redundancy in the program, and we have developed empirical regression models that use the quantitative redundancy metrics to estimate the \(REM\). Also, we have used compiler generation technology to automate the estimation of the redundancy metrics of Java programs by analyzing their state space and the reduction in entropy that stems from the emergence of equivalent mutants, using techniques such as species discovery [3] and related methods [6]. A secondary research venue is to explore an alternative formula for the essential mutation score, where the numerator is \(\text{NEC}(X, REM)\) rather than \(\text{COV}(N, K, X)\); this is currently under investigation.

### REFERENCES


### Table

<table>
<thead>
<tr>
<th>Function</th>
<th>Interpretation/Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{REM}(K, S, F, N, D))</td>
<td>Ratio of equivalent mutants of a program, as a function of redundancy metrics</td>
</tr>
<tr>
<td>(\text{NEC}(N, REM))</td>
<td>Number of equivalence classes, set of size (N) where (REM) is the probability of equivalence of any two elements</td>
</tr>
<tr>
<td>(\text{COV}(N, K, X))</td>
<td>Number of equivalence classes covered by a set of size (X) in a set of size (N) partitioned into (K) classes</td>
</tr>
<tr>
<td>(\text{EMS}(N, REM, X))</td>
<td>Essential mutation score of a test data that has killed (X) mutants out of (N) for a program whose (REM) is (REM)</td>
</tr>
<tr>
<td>(\text{NOI}(N, K))</td>
<td>Number of draws in a set of size (N) partitioned into (K) classes to ensure that all classes are represented</td>
</tr>
</tbody>
</table>

**Figure 1: Functions and their Meaning**