All-Uses versus Mutation Testing: An Experimental Comparison of Effectiveness*

Phyllis G. Frankl Stewart N. Weiss Cang Hu

June 7, 1996

Abstract

The effectiveness of a test data adequacy criterion for a given program and specification is the probability that a test set satisfying the criterion will expose a fault. Experiments were performed to compare the effectiveness of the mutation testing and all-uses test data adequacy criteria at various coverage levels, for randomly generated test sets. Large numbers of test sets were generated and executed, and for each, the proportion of mutants killed or def-use associations covered was measured. This data was used to estimate and compare the effectiveness of the criteria. The results were mixed: at the highest coverage levels considered, mutation was more effective than all-uses for five of the nine subjects, all-uses was more effective than mutation for two subjects, and there was no clear winner for two subjects. However, mutation testing was much more expensive than all-uses. The relationship between coverage and effectiveness for fixed-sized test sets was also explored and was found to be non-linear and, in many cases, non-monotonic.

1 Introduction

Although many systematic approaches to software testing have been developed over the last two decades, there is relatively little data available indicating how effective they are. Without such data, as well as data indicating how expensive the techniques are, it is difficult for software developers to decide which, if any, of them to adopt.

Many of the techniques of interest are white-box test data adequacy criteria. A set of test requirements is derived by analyzing the program under test, and the test data are considered

*P.G. Frankl is with the Computer and Information Sciences Dept., Polytechnic University, 6 Metrotech Center, Brooklyn, N.Y. 11201; E-mail: phyllis@morph.poly.edu. C. Hu was with Polytechnic University when this work was performed and is now at JYACC Corp. Their work was supported in part by NSF Grant CCR-9206910. P. Frankl was partially supported by a Visiting Fellowship grant, GR/L00445, to the Centre for Software Reliability from the UK Engineering and Physical Sciences Research Council. S.N. Weiss is with the Computer Science Dept., Hunter College, 695 Park Ave., New York, NY 10021. E-mail: weiss@roz.hunter.cuny.edu. His work was supported in part by NSF Grant CDA-9222720.

adequate only if they satisfy these requirements. For example, in statement testing, each requirement corresponds to a statement in the program; the test data is considered adequate if it causes the execution of each statement. The all-uses data flow testing criterion and the mutation testing criterion, summarized below, are more demanding white-box adequacy criteria, which many researchers consider to be promising.

Unfortunately, it is difficult to measure and compare the effectiveness of adequacy criteria in a meaningful way. Consider an erroneous program P, its specification S, and a test data adequacy criterion C. Even if we restrict the size of the test sets to be considered, there are a large number of different test sets that satisfy criterion C for P and S. These adequate test sets typically have different properties: some may detect one or many faults, while others detect no faults; some may be difficult to execute and check while others are easy. Thus, in defining and measuring effectiveness (or cost) of testing criterion C for P and S, it is not sufficient to consider a single representative test set. Rather, the space of test sets must be sampled and appropriate statistical techniques must be used to analyze the results.

This probabilistic notion of effectiveness can be studied either analytically or experimentally. For example, Frankl and Weyuker showed that if a certain relation called the "universally properly covers" relation holds between criteria C_1 and C_2, then assuming that test cases are selected according to an idealized randomized selection strategy, test sets selected to satisfy C_1 are at least as likely to detect a fault as those selected to satisfy C_2, regardless of the program or the nature of the fault [FW93b]. [1] This approach offers some insights into the strengths and weaknesses of the criteria, but also has some limitations: it is not clear to what extent the results can be generalized to other test selection strategies, and the resulting comparisons do not quantify how much more effective one criterion is than another. Furthermore, the criteria of interest in this paper, all-uses and mutation testing, are incomparable under the universally properly covers relation. Thus, experimental approaches are necessary.

We previously developed an experiment design that addresses this issue and used it to compare the effectiveness of all-uses to that of branch testing for a suite of nine small to moderate sized programs [FW93a]. The current paper compares all-uses to mutation testing for this suite of programs. The data gathering technique was refined to make it more amenable to studying mutation testing and to allow efficient comparison of larger numbers of test sets.

Consider a given program, test set selection strategy, and test data adequacy criterion. The test selection strategy gives rise to a collection of adequate test sets and a probability distribution on that collection. The *effectiveness* of a test data adequacy criterion (relative to the selection strategy, program, and specification) is defined to be the probability that an adequate test set (selected by the strategy) will expose the existence of a fault. In these experiments, we measured effectiveness by generating a statistically significant sample of the space of adequate test sets and determining which of the test sets exposed a fault.

One question of interest is whether test sets that satisfy (or almost satisfy) mutation testing are more effective than those that satisfy (or almost satisfy) all-uses. To address this

[1] Roughly speaking, criterion C_1 universally properly covers C_2 if for every program and specification every subdomain each subdomain of C_2 is a union of subdomains of C_1. No subdomain of C_1 can be used more often in covering the collection of subdomains of C_2 than its multiplicity in the multi-set of subdomains induced by C_1.

question we compared the two adequacy criteria using statistical tests of various hypotheses about their relative effectiveness, at significance levels of 1%.

A second group of questions deals with the relationship between coverage and effectiveness. How do test sets that satisfy $x\%$ of the requirements of criterion C compare to those that satisfy $y\%$ of the requirements? What are the numerical values of the effectiveness for test sets that achieve various coverage levels? How large would randomly generated test sets (not subjected to any adequacy criterion) have to be to achieve those same levels of effectiveness? We address these questions by measuring the effectiveness of the criteria at various coverage levels for fixed-size test sets for each program. These effectiveness levels are compared to those that would be achieved by random test sets of various sizes.

We consider the research presented here to be a small step toward answering the question of which testing techniques should be adopted by practitioners. It answers the above questions for a small group of relatively small programs and thus gives practitioners some insight into the strengths and weaknesses of the criteria examined. In particular, it shows that the testing techniques examined are sometimes extremely effective, but can also be rather ineffective. It casts some doubt on whether the effectiveness of the criteria justifies their cost. Unfortunately, it is impossible to draw conclusions about programs in general from empirical studies of this nature. However, we hope that as additional experiments are performed on more and larger programs, clear trends will begin to emerge, indicating which techniques tend to be the most cost-effective.

Section 2 of this paper reviews relevant background on data flow testing, mutation testing, and the definition of the effectiveness of an adequacy criterion. Section 3 describes the design of the experiment and the statistical techniques used; further discussion of this design and how it overcomes deficiencies in previous experiment designs, along with additional details on the statistical techniques, can be found in Reference [FW93a]. Section 4 briefly describes the programs that were used as subjects for the experiments. Section 5 presents and discusses the results. Section 6 compares our results to those of related studies. Section 7 points out directions for future work.

2 Background

2.1 Definitions of the adequacy criteria

This study compares the effectiveness of three adequacy criteria: the mutation testing criterion, the all-uses criterion, and the null criterion. Mutation testing and all-uses have been widely discussed in the literature. We briefly review their definitions here. The null criterion considers any test set to be adequate; thus application of the null criterion is the same as not using any adequacy criterion at all. We have included the null criterion in this study in order to investigate whether using a sophisticated criterion, such as mutation testing or all-uses, is significantly better than testing without an adequacy criterion.

An adequacy criterion is typically applied after the test set has been selected. The test selection strategy may be based on the adequacy criterion, may be based on entirely different factors, may be random (based on some well-defined distribution), or may be "haphazard". In the experiments reported here, we apply the adequacy criteria to randomly generated test sets. The combination of random test generation and the null adequacy criterion is what often referred to in the literature simply as "random testing".

The all-uses criterion [RW85] [2] is based on data flow analysis. A *definition-use association* (dua) is a triple $(d, u, \mathtt{v})$ such that d is a node in the program's flow graph in which variable $\mathtt{v}$ is defined, u is a node or edge in which $\mathtt{v}$ is used, and there is a definition-clear path with respect to $\mathtt{v}$ from d to u. A test case t *covers* dua $(d, u, \mathtt{v})$ if t causes the execution a path that goes from d to u without passing through any intermediate node in which $\mathtt{v}$ is redefined. The all-uses criterion demands that the test data cover every dua in the subject program. In these experiments, we used ASSET [Fra87, FWW85] to check the extent to which a given test set for a given Pascal program satisfies the all-uses criterion.

One problem with the all-uses criterion (and related criteria), as originally defined, is that for some programs, no adequate test set exists. This problem arises from infeasible paths through the program, i.e., paths that can never be executed. In practice, it is necessary to eliminate unexecutable definition-use associations from consideration [Fra87, FW88].[3]

Mutation testing [DLS78, Bud81] deems a test set to be adequate if it can distinguish the subject program from a collection of similar programs, called *mutants*, obtained by making small syntactic modifications to the subject program. Given a program P, a set of mutation operators, and a set of test data, a mutation testing system creates a set of mutant programs, each obtained by applying a single mutation operator to a single location in P. Typical mutation operators include replacing one variable by another and replacing one relational operator by another. In these experiments we used the Mothra tool [DGM+88], which performs mutation testing on Fortran programs, using a set of 22 mutation operators.

Those mutants that compute precisely the same function as P are called *equivalent mutants* and the others are called *inequivalent mutants*. The characterization of mutants as equivalent or inequivalent may depend on the environment (e.g., how variables are initialized) and on assumptions about P's input domain. If, on input t, the output of program P differs from that of mutant P', the test case t is said to *kill* mutant P'. The test set T is adequate if each inequivalent mutant is killed by some test case t in T.

Note that, like the question of whether a given def-use association is executable, the question of whether a given mutant is equivalent to the subject program is a semantic one. These questions are undecidable in general and addressing them is typically the most labor-intensive aspect of using these adequacy criteria. For the experiments reported here, we identified unexecutable definition-use associations and equivalent mutants by hand.

2.2 Effectiveness of an adequacy criterion

The goal of testing is to detect faults in programs. A test case t *exposes a fault* in program P, if on input t, the output of P is different than the specified output. A test set T is *exposing* if at least one test case t in T exposes a fault. We are not concerned here with the particular

[2] The original criteria were defined for programs written in a simple language. The definitions were subsequently extended by Frankl and Weyuker [FW88] to programs written in Pascal. We adopt their conventions and notation here.

[3] In Reference [FW88] the resulting criterion is called all-uses*. In this paper, we use the term all-uses to refer to all-uses*, as is common practice in the literature.

fault that caused the program to fail or the number of faults in the program, but only with whether the *existence of at least one fault* is demonstrated by the test set.

Consider a testing process in which a test set is generated by some given test generation algorithm, the program is executed on the test set, and, if no bugs are found, the program is released *provided that* some adequacy criterion (such as coverage of 100% of the executable duas or killing of 95% of the inequivalent mutants) has been satisfied. Although the released program is not guaranteed to be correct, one would like a good adequacy criterion to be less likely to allow release of a faulty program than a poor adequacy criterion.

The "effectiveness" of an adequacy criterion captures this intuition. Consider a given probability distribution on the space of all C-adequate test sets for program P and specification S. We define the *effectiveness* of C to be the probability that a test set selected randomly according to this distribution will expose a fault. In practice, the distribution on the space of test sets is often induced by a test generation or selection strategy G, such as random testing with a given input distribution, some other form of black-box testing, or a systematic white-box strategy. We define the *effectiveness of criterion C for P and S relative to test generation strategy* G to be the probability that a C-adequate test set generated by G will expose a fault in P. In this paper, we measure effectiveness of criteria relative to various random test set generation strategies.

Note that this notion of effectiveness captures the intuition of the "goodness" of an adequacy criterion: Suppose that P is incorrect and that we know *a priori* that C_1 is more effective than C_2 for P. Then the probability of our mistakenly believing P to be correct after using C_2 as an adequacy criterion is at least as great as if we had used C_1 as the adequacy criterion. (That is, the probability that a C_2-adequate test set will fail to expose a fault, even though one is present, is at least as great as the probability that a C_1-adequate test set will fail to expose a fault.) Thus after testing P without exposing a fault, we can have at least as much confidence that P is correct if we used criterion C_1 as if we used criterion C_2. [4]

3 Techniques

3.1 Sampling Technique

In order to measure the effectiveness of criterion C for program P and specification S, relative to distribution D, one can randomly sample the space of C-adequate test sets according to D and determine the proportion of these that expose a fault. When the distribution is induced by a random test generation strategy G, the most straight-forward way to do this is to repeatedly randomly generate a test set, check whether it is adequate according to C, and, if so, check whether it exposes a fault. This can be extremely computationally intensive.

To make test set generation more efficient, we first selected a *universe* of test cases for each program, then built a *coverage matrix* for each criterion. Each row in the matrix corresponds to a test case in the universe and each column corresponds to a testing *feature*

[4] Note that this is a statement about the probability that the program will be released with undiscovered faults. The fact that C_1 is more effective than C_2 *does not* imply that the the program will be more reliable, i.e., less likely to fail in operation, after testing with C_1 than with C_2.

(a def-use-association that must be covered or a mutant that must be killed.) Features are testing *requirements* that must be covered or killed by test cases. An entry of '1' in position (i, j) indicates that test case i kills feature j. To automate this, we added some pre- and post-processing programs to ASSET and Mothra. Note that any column that has at least one '1' corresponds to an executable dua (inequivalent mutant), while the columns that have only zeros correspond to *potentially* unexecutable duas (equivalent mutants), i.e., those that are not covered by any test case in the universe; these must be be examined manually. The second step was to build a column vector indicating which test cases in the universe expose a fault. Once the column vector and coverage matrix were built, it was possible to *simulate* the generation and execution of a large number of test sets, without further execution of the program or the mutants. For each test set, we simply randomly selected a set of rows (each row representing a test case), `OR`-ed them together to determine which requirements the test set covered, checked whether the test set exposed a fault, and recorded in a table the coverage and whether the test set exposed a fault.

Mothra allows the user to specify whether memory is uniformly initialized to zero or some other value, or is filled with random bits, prior to execution of the subject program and/or its mutants. We chose to initialize memory to zero values for subject program execution, and when possible, for mutant execution also, in part to make the experiment reproducible. However, for some of the subject programs, this left large numbers of mutants that were not killed by any tests in the subject programs' universes. Since it was impractical to manually determine which of these live mutants were equivalent, for these subjects we chose instead to initialize memory to random values for mutant execution. This made it much easier to kill mutants, and as a consequence made mutation analysis less demanding. For subjects `find1`, `strmatch1`, `strmatch2`, and `textformat` we initialized memory to zeros, and for the others we used random value initialization. For comparison, we repeated the `textformat` experiment using random value initialization, and observed a small difference in the results. We report results of both experiments, using the identifiers `textformat.0` and `textformat.r` to distinguish them. Only `textformat.0` is included in our summary of results.

We note that ideally, one would like to sample the *entire* space of adequate test sets (or at least those of some particular sizes) according to a distribution that reflects the way adequate test sets would be selected in practice. Unfortunately, such a *natural* distribution is difficult, if not impossible, to characterize. Our approach differs from this "natural" approach in that all of our test sets are made up of test cases from a particular universe and that the selection of distribution on the space of test sets is induced by uniform random selection of fixed-sized test sets from this universe. We tried to select "reasonable" universes, but it is possible that these universes may have unusual properties which make the results different than those that would be obtained using the natural distribution. For example, if all the test cases in the universe that kill a particular mutant M also expose a fault, then our results will indicate that 100% mutation coverage is extremely effective. If, however, there are many test cases in the input domain and outside the universe that kill M and do not expose a fault, our results may differ from the results of an experiment that uses the natural test set distribution.

Thus, it is important to remember that our experiments compare the effectiveness of the adequacy criteria for particular subject programs and for particular distributions on the space of adequate tests. We cannot guarantee that the results will be similar when other distributions are used. However, it is encouraging to see that when we used two different universes for one of the subject programs, the results were rather similar, qualitatively.

3.2 Statistical Analysis Techniques

For each program, for each of several fixed test set sizes, and for each of several coverage levels, we compared the effectiveness of test sets that killed at least $x\%$ of the inequivalent mutants to that of test sets that covered at least $y\%$ of the executable duas. Let p_m denote the effectiveness of test sets (of the given size) that kill at least $x\%$ of the inequivalent mutants and let p_u denote the effectiveness of test sets (of the given size) that cover at least $y\%$ of the executable duas. We would like to estimate these values by sampling the space of test sets. Consider the collection of sampled test sets of the given size that kill at least $x\%$ of the inequivalent mutants. Let $\hat{p}_m$ denote the proportion of these that expose a fault. Similarly, consider the collection of sampled test sets of the given size that kill at least $y\%$ of the executable duas. Let $\hat{p}_u$ denote the proportion of these that expose a fault. These measured quantities are estimates of the true effectiveness values.

Using standard statistical techniques, we can determine whether the difference $|\hat{p}_m - \hat{p}_u|$ is great enough that it is not due to chance. We use *hypothesis testing* to pit the *research hypothesis*, $H_1 : p_m \neq p_u$, against the *null hypothesis*, $H_0 : p_m = p_u$. Since the underlying data is categorical (each test set either does or does not expose a fault), the possible sample proportions follow binomial distributions. Assuming the samples are independent, under H_0, their difference follows a binomial distribution whose mean is 0 and whose variance can be estimated by a weighted average of $\hat{p}_m$ and $\hat{p}_u$. For sufficiently large samples we can use the normal approximation to the binomial distribution to determine whether the difference in sample means is sufficiently far from 0 that it is highly unlikely to be due to chance.[5] If so, we reject the null hypothesis and determine that there is a *significant difference* between the effectiveness of the criteria. We used a significance level of 0.01, meaning that if the null hypothesis is rejected, the probability that mutation testing and all-uses are *actually* equally effective is at most 0.01. For a fuller explanation of hypothesis testing, see Reference [BJ77] and for a discussion of why it is an appropriate tool for this type of experiment, see Reference [FW93a].

Note that "significant difference" is a technical term, relating to whether the observed difference in sample proportions can be due to choice of unrepresentative samples. When there is a large sample, it is possible to have a significant difference between values that are very close numerically. The decision as to whether this is an *important* difference in practice is a subjective one.

To gain insight into the numerical values of effectiveness of the criteria for these subjects we used standard statistical techniques to determine confidence intervals around the estimates of $\hat{p}_m$ and $\hat{p}_u$. This is a range such that it is highly unlikely that the true value is outside of that range. Again, a significance level of 0.01 was used.

In addition, for each program we examined the relationship between coverage and effectiveness for test sets of a given size by computing the effectiveness of those test sets that cover at least $x\%$ of the requirements, for various values of x. Again using the normal approximation to the binomial distribution (where applicable), confidence intervals around

[5] Note that in these experiments, unlike those reported in Reference [FW93a], we used a two-sided test.

these effectiveness values were computed. A significance level of 0.05 was used. The values of (coverage, effectiveness) were plotted, with error bars indicating the confidence intervals, and qualitative features of these graphs were examined.

Note that in comparing coverage to effectiveness, test set size is a confounding factor. For $x > y$, test sets that achieve $x\%$ coverage tend to be larger than those that achieve $y\%$ coverage. Since larger test sets are more likely to expose a fault than smaller ones, an increase in effectiveness as one goes from $y\%$ coverage to $x\%$ coverage may be due to increased test set size rather than to increased coverage, per se. If an increase in effectiveness is due to increased test set size, rather than to the inherent properties of increased coverage, it may be easier to achieve effective testing in practice simply by using larger test sets, rather than by using higher coverage test sets.

In the experiments reported in Reference [FW93a], we included test sets of various sizes in our sample, then tried to understand the effect of size by using logistic regression to relate effectiveness to the independent variables, size s and coverage c. In most cases it was difficult to find reasonably simple functions of s and c which fit the data, thus difficult to obtain intuitively clear explanations of how size and coverage contribute to effectiveness. We also grouped together test sets that were close in size and compared effectiveness of all-uses to all-edges for similar sized test sets.

The improved efficiency of this experiment design allowed us to address the issue of test set size more thoroughly. All of the results in this paper control for the confounding variable of test set size. To do this we selected several sizes of interest for each subject. For each size s, we simulated the random generation of a large number of test sets of size s (by selecting s rows of the matrix). We analyzed the results for each size separately and plotted coverage versus effectiveness for fixed size test sets. Thus, any plotted increase (or decrease) in effectiveness as coverage increases is due solely to coverage, not to size.

In our approach, the relationship between coverage and effectiveness is explored while size is held constant. Wong et al. [WHLM94] have explored the relationship between size and "fault-detection effectiveness" (as measured by proportion of seeded faults detected) when coverage is held constant. They compared effectiveness of randomly generated test sets to minimum sized subsets with the same coverage. Although it is beyond the scope of this paper, our data could be used to explore the relationship between effectiveness and size for fixed coverage levels.

4 Subject Programs

The design of the experiment imposes several constraints on the subject programs. The most important of these are that the program must have a fairly subtle fault, i.e., one that is exposed by relatively few test cases, and that there must be an efficient way to generate test cases and check the correctness of their outputs.

Our nine subjects were obtained from seven programs, all of which had naturally occurring faults. Due to the constraints imposed by available tool support, Pascal versions were used for the all-uses subjects and equivalent Fortran versions, as close as possible in structure, were used for the mutation testing subjects. In each case, one subprogram was instrumented for testing (i.e., we monitored the coverage of duas and killing of mutants from

subject	LOC	mutants	duas	inequiv mutants	exec duas	failure rate
`determinant`	60	4489	298	4123	103	0.0008
`find1`	33	932	114	836	93	0.066
`find2`	33	932	114	859	93	0.018
`matinv1`	60	4303	298	3971	106	0.012
`matinv2`	28	1267	81	1145	62	0.014
`strmatch1`	22	398	49	356	49	0.032
`strmatch2`	23	402	56	361	54	0.062
`textformat.0`	26	976	50	905	42	0.066
`textformat.r`	26	976	50	976	42	0.066
`transpose`	78	5358	97	4595	88	0.023

Table 1: Subject Programs

one subprogram.) We obtained more than one subject from two of the programs by using different input distributions and by instrumenting different procedures.

The subjects are described briefly below, and more thoroughly in Reference [WF91]. Table 1 gives the numbers of lines of code (in the instrumented procedure), mutants, duas, inequivalent mutants, executable duas, and proportions of failure causing test cases in each universe.

Where possible, we re-used the universes generated in our earlier experiment [FW93a]. However, for a few of the programs, there were inequivalent mutants that were not killed by any test case in the universe. For these subjects, we added additional test cases to the universe, by random generation, sometimes using a different random generator than for the original universe, and/or by selection of special test cases aimed at killing particular mutants. For a few subjects, a few mutants remained alive. We believed that these were equivalent, although we could not prove so. Nonetheless, we treated such mutants as equivalent. This issue is discussed further in Section 6.

Subjects `find1` and `find2` both have Buggyfind [BEL75] as the subject program. Buggyfind takes as input an array of integers and an array index. In `find1`, the test universe consists of 1000 randomly generated arrays, with sizes in the range [0..10], elements in the range [1..100], and indices in the range [1..array-size]. In `find2` the universe contains one test case for each array with elements selected from $\{0, 1, 2, 3, 4\}$ and range $[0 \ldots n]$, for each n from 0 to 5.

`textformat` is the text formating program analyzed by Goodenough and Gerhart [GG75], with all the faults corrected except N5 and N6, which caused leading blanks and adjacent blanks/line-breaks to be mishandled. The original universe consisted of 1,000 15-character long strings, each generated by repeated uniform random selection of a character from the set { x, blank, newline }, with a distribution that mimicked uniform selection from the set of letters, blank and newline characters. An additional 1,000 test cases, involving a larger set of characters, were added to the universe, in order to kill more mutants.

`transpose` is the transpose routine of a sparse-matrix package [Gus78, McN71], which fails when the first row of the matrix is all zeros. The universe has 1,000 randomly generated

r by c matrices ($0 \leq c \leq r \leq 50$) with densities between 0 and 66%.

`strmatch1` and `strmatch2` are string matching routines, each with a fault that we had previously inadvertently introduced. `strmatch1` fails whenever the null pattern is input, and `strmatch2` has the maximum pattern length shorter than specified. The universe consists of all (text, pattern) pairs on a two letter alphabet with text length and pattern length ranging from zero to four.

`matinv1`, `matinv2`, and `determinant` are matrix manipulation programs based on LU decomposition [PFTV86], written by graduate students. The error is a misunderstanding of the specification: the algorithm is intended for non-singular matrices; when a singular matrix is input, the algorithm is sometimes, but not always, able to determine this and give an error message. Thus, the programs fail on some, but not all, singular matrices [6]. In `matinv1` and `determinant` the lower-upper decomposition procedure was instrumented, while in `matinv2` the back-solving procedure was instrumented. For the matrix inversion programs the original universe consists of 1,000 square matrices with sizes uniformly selected between 0 and 5 and with integer entries uniformly selected between 0 and 24, while for `determinant` the size was between 1 and 5. Additional test cases with real entries were added to kill mutants not killed by the original universe.

5 Results

Tables comparing the criteria at or near 100% coverage are shown in the appendix. Examination of the tables shows that the results are mixed: mutation coverage is more effective than dua coverage for five subjects, dua coverage is more effective than mutation coverage for two others, and there is no significant difference for the remaining two. Comparisons of the criteria at 98% coverage and at 95% coverage are given elsewhere [FWH94].

For one subject, `strmatch1`, both mutation and dua coverage are extremely effective at coverage levels above 98%. The effectiveness drops substantially as coverage decreases to 95%, especially for dua coverage[7].

For `find1`, `matinv1`, and `strmatch2`, mutation coverage is more effective than dua coverage at all of the sizes and coverage levels compared. For `find1` and `strmatch2`, mutation coverage was quite effective (extremely so, for `find1`), while dua coverage appeared to be no better than the null criterion. For `matinv1` mutation coverage was extremely effective and dua coverage was quite a bit less effective but still substantially better than the null criterion. Very high levels of mutation coverage are extremely effective for `transpose`, and moderately effective for `matinv2`, but the effectiveness falls off rapidly as coverage decreases; dua coverage for these subjects is no more effective than the null criterion.

On the other hand, for two other programs, dua coverage does better than mutation coverage. For both `textformat.0` and `textformat.r`, dua coverage is more effective than mutation coverage for all the sizes/coverage-levels at which there is sufficient data. All levels of dua coverage examined are extremely effective, while mutation coverage is at most slightly

[6] Interestingly, the determinant and matrix inversion programs do not fail on the same set of inputs.

[7] No verdict is given in the table for this program because the fact that $\hat{p}_m = \hat{p}_u = 1.0$ makes the normal approximation inapplicable. It is clear by inspection that there is no significant difference between 100% mutation coverage and 100% dua coverage for this subject.

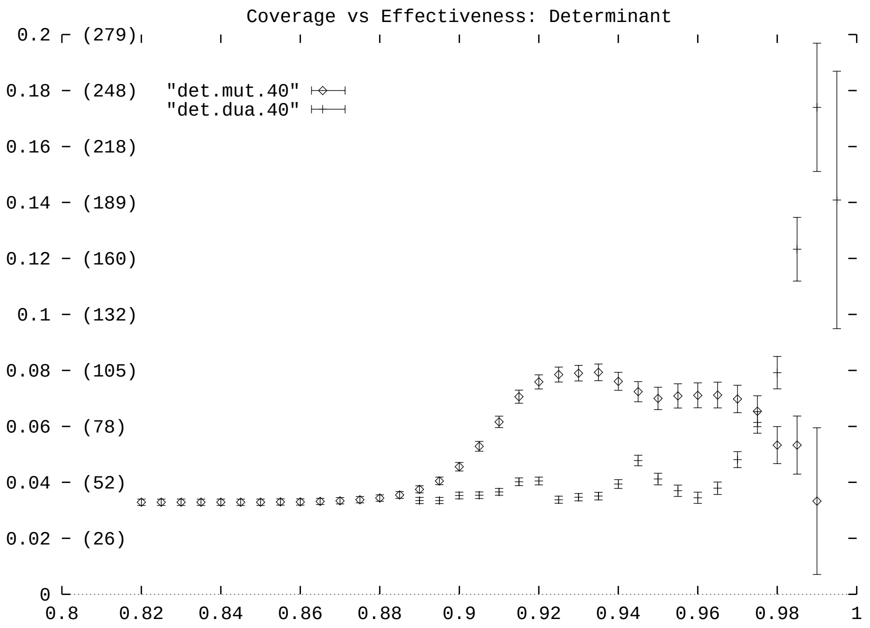

Figure 1: Coverage vs. effectiveness for `determinant`. Size 40.

more effective than the null criterion. For `determinant`, dua coverage is more effective for coverage levels of 98% or better and sufficiently small test sets, but both criteria are quite ineffective at coverage levels below 98%.

Finally, for `find2`, the data is not sufficient to reject the hypothesis that the effectiveness of the criteria is the same.

Graphs of coverage versus effectiveness for each program are shown in Figures 1 through 9. For an abscissa value of x the ordinate value represents the effectiveness of test sets (of the given size) achieving coverage of *at least* $100x$%. Mutation testing is indicated by diamonds and all-uses by either squares or hash marks, as indicated in the key. Error bars represent 95% confidence intervals. [8] For comparison, when appropriate data was available, we also plotted coverage versus effectiveness for branch testing. Note that the scales on both axes are different for each plot and in particular that the ordinate scale does not always reach 1.0. Given a failure rate of θ for the universe and a test set size of n, the null criterion would be represented by a horizontal line at $y = (1-(1-\theta)^n)$. In all of the graphs, except Figure 7, the reader can easily infer the position of this line by extending the horizontal line representing low coverage of the criteria.

Note that the scale on the abscissa axis refers to the proportion of *executable* duas or mutants covered. Readers interested in analogous information, when unexecutable duas and mutants are included can simply re-scale the axes, using the data in Table 1. For example, covering 100% of the executable duas in the `determinant` program is the same as covering $298/(298+103) = 74\%$ of the total duas.

Several things are noteworthy about these graphs. Almost all of the graphs fall into one

[8] When the measured effectiveness is 100%, the normal approximation does not apply, so we do not include error bars. Points of this nature at which the amount of data is too small are omitted from the plots.

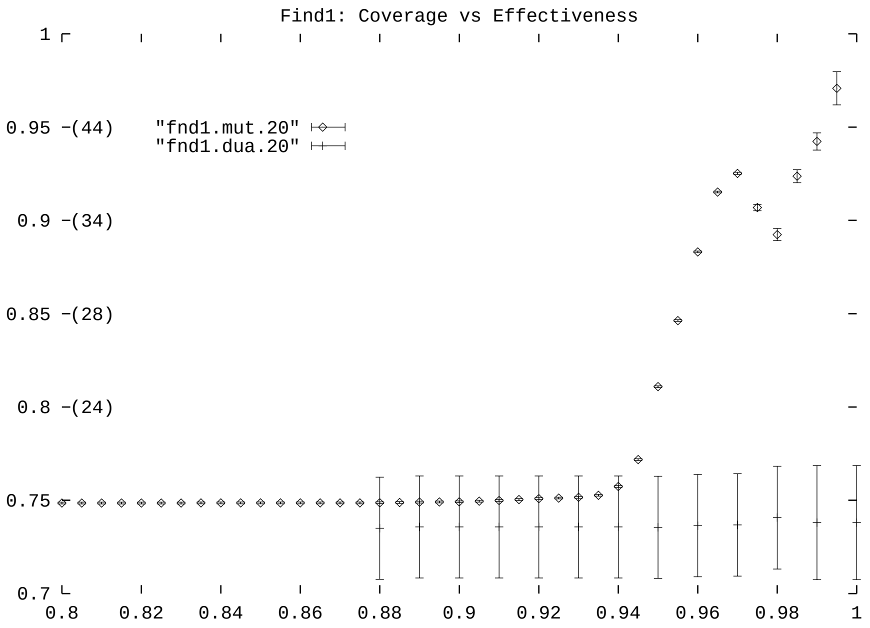

Figure 2: Coverage vs. effectiveness for `find1` program. Size 20.

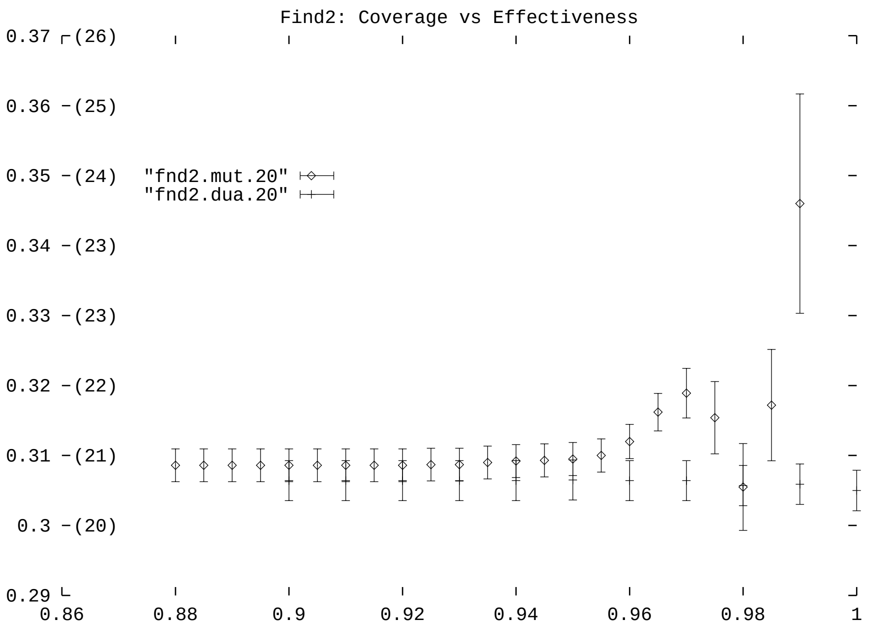

Figure 3: Coverage vs. effectiveness for `find2`. Size 20.

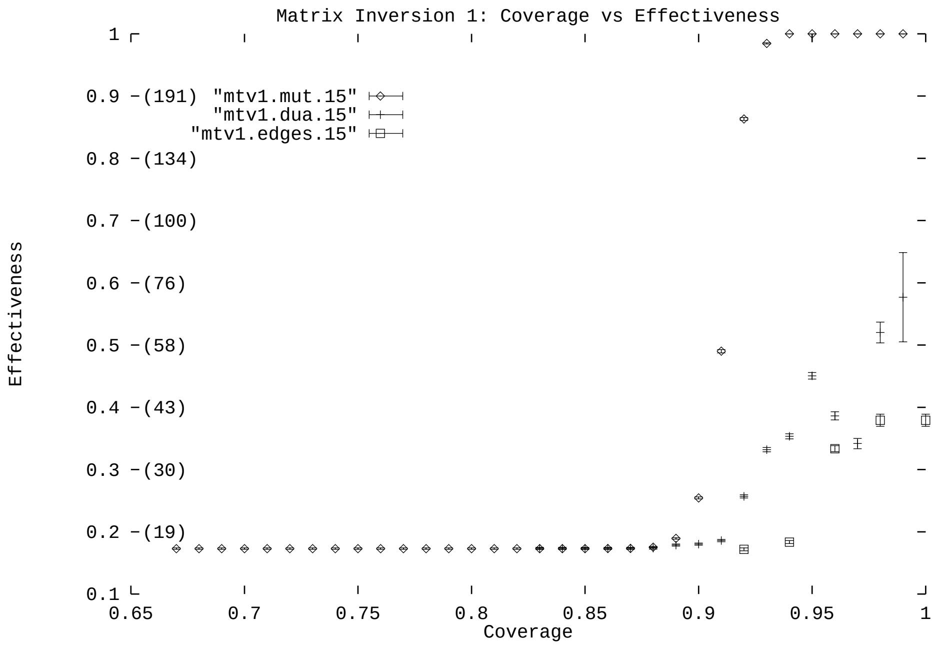

Figure 4: Coverage vs. effectiveness for `matinv1`. Size 15.

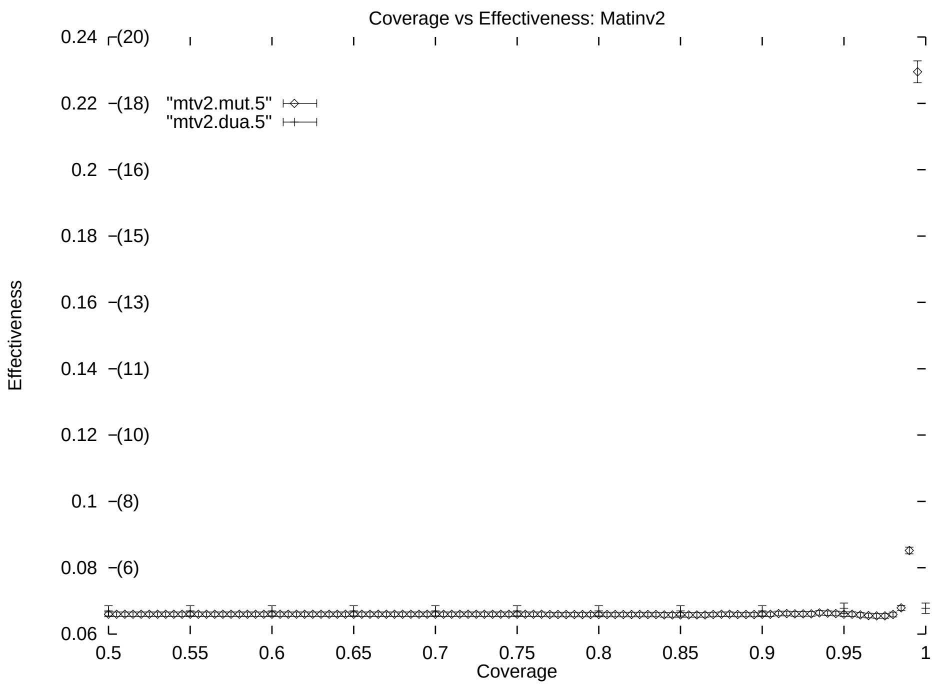

Figure 5: Coverage vs. effectiveness for `matinv2`. Size 5.

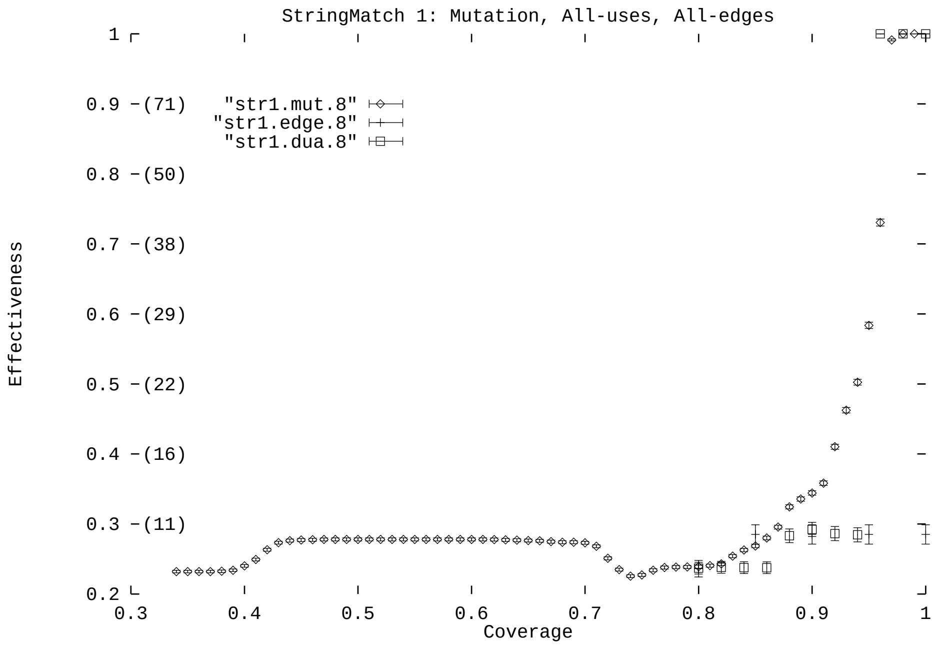

Figure 6: Coverage vs. effectiveness for `strmatch1`. Size 8.

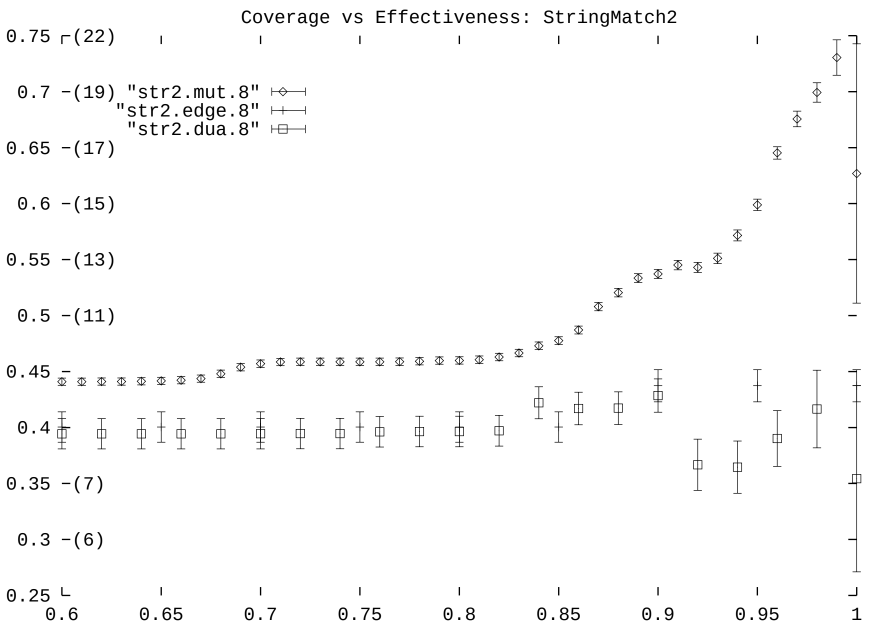

Figure 7: Coverage vs. effectiveness for `strmatch2`. Size 8.

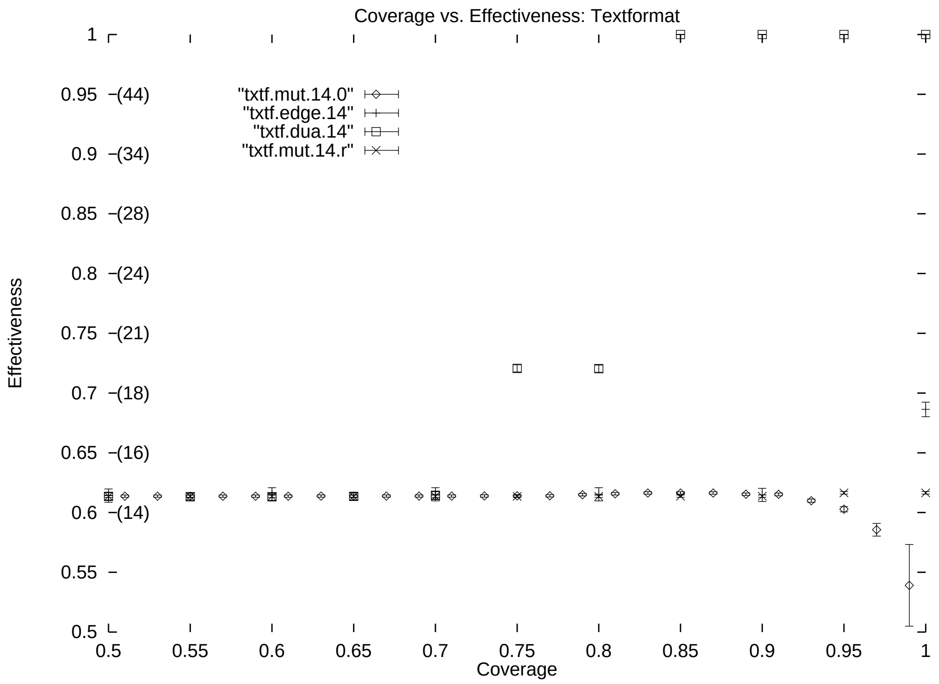

Figure 8: Coverage vs. effectiveness for `textformat`. Size 14

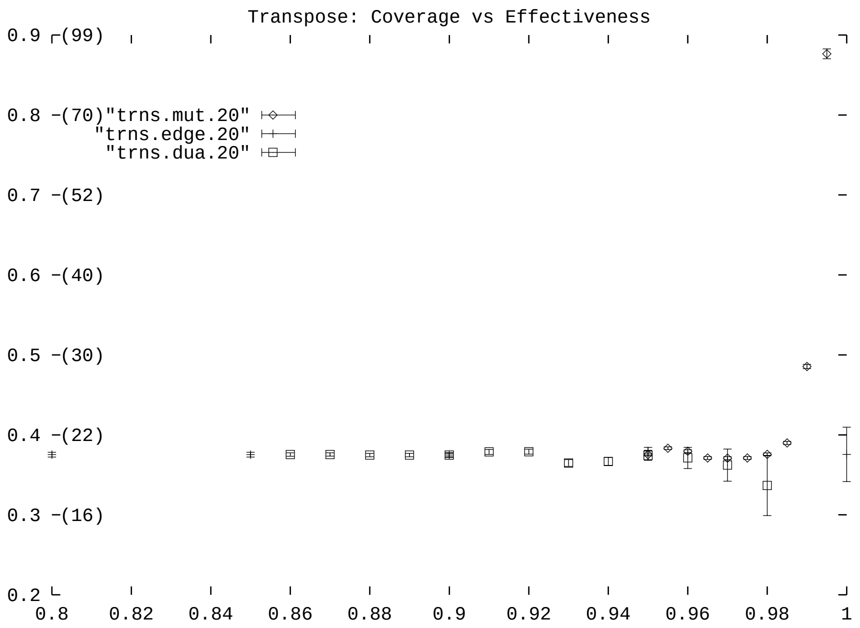

Figure 9: Coverage vs. effectiveness for `transpose`. Size 20.

of two categories:

1. those that are nearly horizontal, and
2. those in which the effectiveness at high coverage is noticibly greater than that at low coverage levels.

Roughly speaking, those in the second category indicate cases in which using the adequacy criterion increases effectiveness, and those in the first category indicate cases in which it does not. For all but one (mutation testing; `strmatch2`) of those in the second category, the graphs of coverage versus effectiveness remain flat (at or near the effectiveness expected for null-adequate test sets of that size) until a fairly high coverage level is reached, then increase steeply. This indicates that when using an adequacy criterion helps at all, it does not help until a high coverage level is reached. For example, there is only one graph (mutation testing for `strmatch2`) in which test sets achieving 80% coverage did better than null-adequate test sets. It is clearly *not* the case that if one achieves 80% coverage, one has done 80% as good a job of testing as if one had achieved 100% coverage.

A second noteworthy aspect of these graphs is their non-monotonicity. Several of the graphs exhibited intervals of high coverage in which the effectiveness was smaller than at lower coverages. Test sets with higher coverage may be less effective than those with lower coverage. Of course, adding test cases to the test set to increase coverage will *never* decrease effectiveness, but that is not the situation modelled here. Rather, our data show that among test sets of a given size, those that achieve at least x% coverage may be less likely to detect an error than those that achieve at least y% coverage, even if $x > y$.[9]

This phenomenon occurs because different test cases cover different numbers of requirements. Let $f(x)$ be the effectiveness of test sets with coverage *exactly* x. It may be difficult or impossible for a test set to achieve coverage within a particular range, say, $[x_1, x_2]$ unless it includes representatives of a certain group of test cases. If a high proportion of these particular test cases happen to expose a fault, we will have high values of $f(x)$ for $x_1 \leq x \leq x_2$. For another range, we may have low values of $f(x)$. The graphs shown represent the functions $g(t)$ obtained by taking weighted averages of the values of $f(x)$ from t to 1 (where the weights correspond to the numbers of test sets achieving each coverage value), and thus can also be non-monotonic. For example, suppose that for a certain program and adequacy criterion, all test cases fall into two disjoint categories, called group A and B. Suppose that almost all test cases in group A cover the same 80% of all required features (such as duas), but do not expose any faults. Suppose, too, that almost all test cases from group B cover the same 30% of required features and almost always expose a given fault. Furthermore, suppose that a test set that contains test cases from both A and B achieves 100% coverage of the required features. Then test sets with coverage around 80% will be less effective than those around 100% and those around 30%. This is a simplification of the kind of forces at work that make the graph non-monotonic.

In only one of the five subjects for which data on branch coverage is plotted (`textformat`), is any level of coverage more effective than the null criterion, and even in this one, the

[9] Note that, since when $x > y$ the "at least x% coverage" criterion subsumes the "at least y% coverage" criterion, this non-monotonicity is an empirical demonstration of the fact that the "subsumes" relation can be "misleading", as discussed in References [Ham89, WWH91].

subject	size (S)	S_m	S_u	S_m/S	S_u/S
`determinant`	20	118	248	6	12
`find1`	20	58	20	3	1
`find2`	10	27	10	3	1
`matinv1`	15	∞	65	∞	5
`matinv2`	5	34	5	7	1
`strmatch1`	8	∞	∞	∞	∞
`strmatch2`	8	16	7	2	1
`textformat.r`	14	14	∞	1	∞
`transpose`	12	129	12	10	1

Table 2: Comparison of test set sizes to size of null-adequate test sets that achieve the same effectiveness.

difference is very small. This corroborates the conclusions we drew (based on less data) in Reference [FW93a] that there is little correlation between branch coverage and effectiveness after controlling for the confounding factor of test set size.

The numbers in parentheses along the ordinates in Figures 1 through 9 indicate the size of a null-adequate test set that would be needed to achieve the given effectiveness level for that program. These values are derived from the failure rate of the universe. For example, the universe for the `determinant` program had a failure rate of 0.008; the size of a null-adequate test set whose expected effectiveness is 0.18 is the smallest number n such that $1-(1-0.0008)^n \geq 0.18$, which is 248. So the 0.18 on the ordinate of the graph for `determinant` is annoted with 248. These numbers aid in interpretation of the plotted data. For example the highest effectiveness level for dua coverage of size 40 test sets for the determinant program was 0.18 (occurring at coverage level slightly above 0.98). The same coverage level could be achieved by null-adequate test sets by increasing the test set size to 248. If it were less expensive to execute and check these additional 208 test cases than to check the extent to which all-uses is satisfied, then random testing (with no adequacy criterion) would be a more cost-effective choice.

This data is summarized in Table 2. The numbers in the column labeled "size" are the sizes of the test sets from which the data were gathered. S_m denotes the size of a null-adequate test that would achieve the same effectiveness as the most effective (almost) mutation-adequate test sets. S_u is the analogous number for all-uses. The last two columns give the ratios of S_m and S_u, respectively, to S (the size), thus give some indication of the benefit of using the adequacy criterion rather than random testing.

Given the expense of using all-uses and particularly the expense of using mutation testing, it is disappointing that in ten of the 18 cases examined, one could achieve the same effectiveness with null-adequate test sets by increasing the test set size by a factor of less than 3! The ratios of S_m to S were generally a little higher than those of S_u to S, but the added expense of using mutation testing (as opposed to random testing) is greater than that of using all-uses. These results indicate that random testing may be a more cost-effective alternative than either mutation testing or all-uses. Duran and Ntafos reached an analogous conclusion based on simulations of hypothetical "partition testing" methods [DN84].

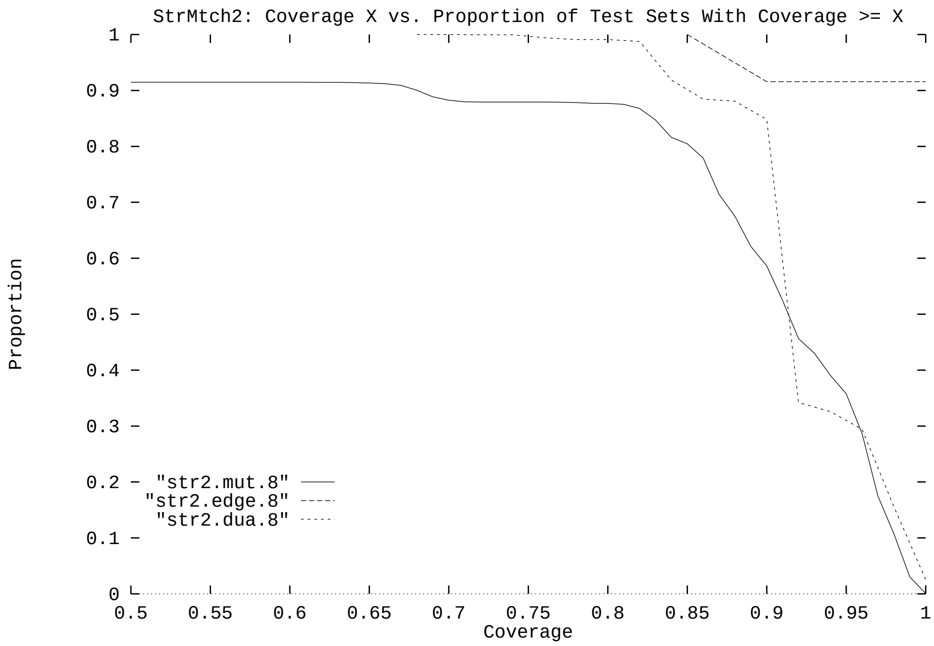

Figure 10: Coverage vs. proportion achieving that coverage for strmatch2. Size 20.

We also observed that it was often extremely difficult to achieve high coverage scores, particularly for mutation testing. To gain more insight into this, we graphed coverage against the proportion of test sets of a given size that achieved at least that coverage level, as illustrated in Figure 10. For the subject shown, it was very easy to achieve high levels of branch coverage with randomly generated test sets of size 8, very difficult to achieve high levels of dua coverage, and nearly impossible to achieve high levels of mutation coverage.

There is no simple characterization of the circumstances under which all-uses and/or mutation testing perform well. However, close examination of the data gives some insight into some of the pheonomena that can occur.

The mutation testing literature hypothesizes the existence of a "coupling effect", wherein test data that is good enough to attain a high coverage score (i.e., to detect the hypothetical faults represented by the mutants) is also good enough to detect faults that actually occur in programs. Our results can be interpreted as checking the validity of the coupling effect for these subjects. Three of the subjects, `matinv1`, `strmatch1`, and `transpose` each had one or more mutants for which every test case that killed the mutant failed. The coupling effect clearly holds for these three subjects. Two others, `find1` and `strmatch2` had no such mutant, but still showed fairly high effectiveness for high mutation coverage levels, thus showed moderately good "coupling". For the other four subjects, the coupling effect held only weakly, if at all.

Previous investigations of the coupling effect [Off92] measured the number of "compound mutants" killed by data that was adequate for killing all simple mutants, and showed a high degree of coupling. The difference between those results and the results from our (admittedly small) sample of natural faults suggests that "compound mutations" may not be an accurate model of natural faults.

Further insight into the relationship between coverage and effectiveness can be gleaned from examination of the extent to which failure-causing inputs are "concentrated" in subdomains. For feature i (the i^{th} def-use-association or i^{th} mutant in program P) let θ_i be the *failure rate* of the set of test cases that cover feature i: the fraction of test cases that cover the feature and that also cause the program to fail. θ_i measures the extent to which failure-causing inputs are concentrated in subdomain i. For mutation testing, θ_i can be considered to be a crude measure of the extent to which the i^{th} mutant fixes the actual fault. A closely related quantity has formed the basis of several analytical investigations of fault-detecting ability [DN84, HT90, WJ91, FW93b].[10]

If $\theta_i = 1$ for some i then any test case that covers feature i will detect a fault, so any test set that achieves 100% coverage will detect a fault. As noted above, this was the case for mutation testing for three of the subjects.

The overall relationship between coverage and effectiveness depends not only on the maximal value of θ_i, but on all the values of θ_i as well as on the complicated dependencies between different features. Several different phenomena are exhibited by the subject programs. We illustrate these with mutation testing subjects; similar phenomena occur for all-uses. Some subjects have very high values of θ_i for a few features, and relatively low values for most of the rest. For example for `transpose` mutation testing, three mutants have $\theta_i = 1.0$, while over 4000 of them have $\theta_i < 0.03$. Thus, test sets which achieve 100% coverage are guaranteed to detect a fault, while those with coverage only a few percent less are unlikely to have killed one of the "good" mutants, hence unlikely to expose the fault. This leads to the sharp rise in effectiveness near 100% coverage, shown in Figure 9.

In other subjects, such as `strmatch2` mutation, the values of θ_i are more evenly spread out, and consequently, effectiveness increases more gradually with increasing coverage. In still others, such as `textformat` mutation, all of the values of θ_i are low (in this case at most 0.1212) and effectiveness does not grow at all with coverage. Although the details of the model are different, this is similar to the situation analyzed by Weyuker and Jeng [WJ91] in which "partition testing" and random testing are equally effective.

6 Comparison to Related Work

Since their introduction, mutation analysis and data flow analysis have been touted as potentially effective methods of test data selection and evaluation. But there were very few actual experiments and little empirical evidence to corroborate any such claims until quite recently. In the last few years, several researchers have done empirical studies to evaluate mutation analysis and the all-uses criterion, separately as well as in comparison to each other, with respect to various measures, including effectiveness at exposing faults, difficulty of satisfaction, and cost of use.

Until the last seven years or so, there had been no experiments on the effectiveness of any of the data flow criteria, and just a few on mutation analysis. There had been some case studies and some theoretical analyses concerned with relative degrees of satisfaction. The most common analytical method of comparing criteria was the subsumption relation.

[10] Our θ_i is measured with respect to a particular universe, whereas the parameter used in the analytical studies is based on the entire input domain.

In fact, it has been known for many years that mutation analysis and the all-uses criterion are incomparable under the subsumption relation. Gourlay proved informally that mutation analysis was incomparable to any criterion that required covering paths of length greater than two,[11] under his power relation [Gou83]. In particular, it followed that mutation analysis was incomparable to all-uses under the power relation. Weiss showed that for any two criteria, C_1 and C_2, if C_1 subsumes C_2, then C_1 is more powerful than C_2 [Wei89]. One implication was that neither all-uses nor mutation analysis could subsume the other. Several papers have downplayed the value of comparison by subsumption [Wei90, WWH91, FW93b] and researchers have begun to evaluate the criteria by other means, including more direct methods of evaluating fault detection ability, including experimentation.

Several papers have reported on investigations of the fault detecting ability of a generalization of path testing called "partition testing". Duran and Ntafos [DN84] performed simulations comparing random testing to partition testing, in which, using hypothetical input domains, partitions, and distributions of faults, they compared the probabilities that each technique would detect a fault. Their conclusion was that, although random testing did not always compare favorably to path testing, it did well enough to be a cost effective alternative. Considering those results counterintuitive, Hamlet and Taylor [HT90] did more extensive simulations, and arrived at more precise statements about the relationship between partition probabilities, failure rates, and effectiveness, which corroborated the Duran-Ntafos results. Jeng and Weyuker [WJ91] attacked the same problem analytically and showed that the effectiveness of partition testing depends greatly on how failure causing inputs are distributed among the "subdomains" of the partition. Frankl and Weyuker [FW93b, FW93c] investigated the conditions under which one criterion is guaranteed to be more likely to detect a fault than another, and found that the fact that C_1 subsumes C_2 does not guarantee that C_1 is better at detecting faults than C_2. A stronger relation with more bearing on fault-detecting ability was also described. Since mutation testing and all-uses are incomparable under the subsumption relation, they are also incomparable under this stronger relation.

Recently there has been increasing recognition of the need to study fault-detecting ability of testing criteria empirically. Several comparisons of mutation and all-uses have been done, but, since the experiment designs and notions of effectiveness employed have differed, it is difficult to compare the results directly.

Mathur and Wong empirically compared various aspects of the all-uses and mutation analysis criteria, including their relative effectiveness. In one experiment [MW93], they defined effectiveness in the same way we do here (as the proportion of adequate test sets that expose at least one fault). They randomly generated 30 adequate test sets for each of three variants of mutation analysis and for all-uses for each of five programs, (two of which were very similar to two of the subjects used in our experiments `find` and `strmatch2`). Several different universes were used for some of the subjects. In all ten of their subjects, mutation testing was at least as effective as all-uses. In addition, less expensive variants of mutation-testing were reported to be superior to all-uses. They did not use any statistical inference procedure, such as hypothesis testing or computing confidence intervals around their estimates of effectiveness, so it is not clear whether the reported advantage is statistically

[11] He argued that mutation analysis was incomparable to $PATH_n$, for $n > 2$. $PATH_1$ is all-edges in this context, and $PATH_0$ is all-nodes.

significant.

In spite of the lack of statistical rigour, it seems striking that their results differ so dramatically from ours. The key difference seems to be that they generated test sets by randomly selecting test cases until the desired coverage was achieved, whereas we generated test sets of a fixed size, then selected all of those that achieved the given coverage level. They did not report on the sizes of the resulting test sets, but one would expect their test generation procedure to yield much larger test sets for mutation testing than for all-uses. Thus we conjecture that the high effectiveness they demonstrated for mutation testing may have been an artifact of large test set sizes.

Offutt et al. also compared the all-uses criterion with mutation analysis [OPTZ94]. They posed four different research hypotheses, of which two were about the relative difficulty of satisfying the criteria, and two were about fault detection properties. In particular, one hypothesis was that mutation adequate test sets will find more faults than all-uses adequate test sets. They used ten different subject programs, each of a very small size (between 11 and 29 statements) and only five test sets per subject for each criterion. Faults were seeded by hand, and test sets were generated by Godzilla [DO93] for mutation, and by a random method for all-uses. No attempt was made to correct for the large disparity in the size of mutation adequate versus all-uses adequate test sets in the interpretation of the results. The authors found that mutation adequate test sets exposed 16% more faults on average than all-uses adequate test sets. However, these test sets were much larger on average, so again, the advantage of mutation testing may have been an artifact of test set size.

If results like those of Mathur and Wong and of Offutt et al. could be shown to scale up to larger programs and to be statistically significant, they would provide a compelling argument in favor of choosing mutation testing over all-uses, *if cost were not a factor*. However, our results, in which we control for the confounding variable of test set size, are less squarely in favor of mutation testing. Since testing practitioners must generally operate within fixed budgets, controlling for test set size yields important information. If the benefit of using criteria like mutation testing and all-uses is primarily due to the size of the test sets, rather than to coverage, per se, then it may be more cost-effective to use less expensive means of generating large test sets.

7 Conclusions

Although the results of our experiments were quite mixed, over all, mutation testing did better than all-uses. At the highest coverage levels compared, for five of the subjects, mutation was clearly more effective; for two, dua coverage was more effective; for two others, there was little difference between the criteria. However, it was more difficult to achieve high levels of mutation coverage than high levels of dua coverage, and much more difficult to determine which mutants were equivalent than to determine which duas were unexecutable.

Although our experiments were designed to measure effectiveness, we also observed that using these criteria, particularly mutation testing, was costly. Even for these small subject programs, the human effort needed to check a large number of mutants for equivalence was almost prohibitive. Furthermore, as indicated by the ratios of N_m to T_m in Tables 3 and 4 in the appendix, the probability of achieving a coverage level near 100% for mutation testing was often extraordinarily small. This indicates that random testing is a very poor way to achieve mutation-adequate test sets, and thus, more expensive test generation methods are needed. For practitioners, this adds to the difficulty of using the criteria, while for experimenters, it may also introduce bias. Analogous problems for the all-uses criterion were present but less severe. These costs, along with the fact that the more expensive criteria were not always substantially more effective than less expensive ones, casts doubt on the cost-effectiveness of all-uses and (even moreso) of mutation testing.

A number of problems arose in the design and implementation of the experiment. One difficulty we encountered was generating universes that were large and varied enough to kill all of the inequivalent mutants and executable duas. In several cases we found that our first attempts at generating universes left some inequivalent mutants that were not killed by any elements of the universe. We considered two different solutions to this problem. One was to tailor additional test cases to particular mutants. This approach can bias the results, as including a test case that exposes the fault will make 100% mutation testing *appear* to be 100% effective, while including one that does not expose the fault may lower the measured effectiveness slightly. A second approach was to generate a large number of additional test sets in such a way that the remaining mutants were likely to be killed. Each of these approaches models a less natural distribution on the space of test cases than those universes generated by using a single strategy. There were several other difficulties:

- The enormous computational and human effort needed to run all mutants on all test cases, identify inequivalent mutants, and simulate execution of hundreds of thousands of test sets;
- The possible sensitivity of the results to the choice of the universe and to environmental factors, such as how memory is initialized by the mutation testing tool. The fact that mutation was more effective than all-uses for `find1` but not for `find2` demonstrates that our results are at least somewhat dependent on the choice of the universe. On the other hand, the graphs corresponding to these two programs have roughly the same shape, so it is not clear how severe this problem is.
- The difficulty of achieving statistically significant numbers of test sets with high enough coverage, when there are large numbers of features, several of which are covered by only a few test cases.

We consider these experiments on a small number of relatively small programs to be only a small step toward determining whether all-uses or mutation coverage is a good choice for the testing practitioner. We hope that after additional research, including further experimentation with larger programs, experiments designed to explicitly measure the cost of using the criteria, and experiments with universes generated by other (non-random) test generation heuristics, a clearer picture of the relative strengths and weaknesses of these testing criteria will emerge.

Acknowledgements

We would like to thank the Mothra Group at Purdue University for supplying us with a copy of the Mothra II testing package, Jeff Offutt for answering numerous questions about Mothra, and Eric Wong for supplying the Fortran translations for some of the subject programs. We thank the anonymous referees for their useful suggestions.

References

[BEL75] R.S. Boyer, B. Elspas, and K.N. Levitt. SELECT – a formal system for testing and debugging programs by symbolic execution. *SIGPLAN Notices*, 10(6):234–245, June 1975. International Conference on Reliable Software.

[BJ77] Gouri K. Bhattacharyya and Richard A. Johnson. *Statistical Concepts and Methods*. John Wiley & Sons, Inc., New York, 1977.

[Bud81] Timothy A. Budd. Mutation analysis: Ideas, examples, problems and prospects. In B. Chandrasekaran and S. Radicchi, editors, *Computer Program Testing*, pages 129–148. North Holland, July 1981.

[DGM+88] R. A. DeMillo, D. S. Guindi, W.M. McCracken, A.J. Offutt, and K.N. King. An extended overview of the Mothra software testing environment. In *Proceedings Second Workshop on Software Testing Verification and Analysis*, pages 142–151. IEEE Computer Society Press, July 1988.

[DLS78] Richard A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, April 1978.

[DN84] Joe W. Duran and Simeon C. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering*, SE-10(7):438–444, July 1984.

[DO93] Richard A. DeMillo and A. Jefferson Offutt. Experimental results from an automatic test case generator. *ACM Transactions on Software Engineering Methodology*, 2(2):109–127, April 1993.

[Fra87] Phyllis G. Frankl. *The Use of Data Flow Information for the Selection and Evaluation of Software Test Data*. PhD thesis, Courant Institute of Mathematical Sciences, New York University, October 1987.

[FW88] Phyllis G. Frankl and Elaine J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, SE-14(10):1483–1498, October 1988.

[FW93a] Phyllis G. Frankl and Stewart N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Transactions on Software Engineering*, 19(8):774–787, August 1993.

[FW93b] Phyllis G. Frankl and Elaine J. Weyuker. A formal analysis of the fault detecting ability of testing methods. *IEEE Transactions on Software Engineering*, pages 202–213, March 1993.

[FW93c] Phyllis G. Frankl and Elaine J. Weyuker. Provable improvements on branch testing. *IEEE Transactions on Software Engineering*, 19(10):962–975, October 1993.

[FWH94] Phyllis G. Frankl, Stewart N. Weiss, and Cang Hu. All-uses versus mutation: An experimental comparison of effectiveness. Technical Report PUCS-94-100, Polytechnic University Computer Science Dept., February 1994.

[FWW85] Phyllis G. Frankl, Stewart N. Weiss, and Elaine J. Weyuker. ASSET – a system to select and evaluate tests. In *Proceedings of the IEEE Conference on Software Tools*, pages 72–79, April 1985.

[GG75] J.B. Goodenough and S.L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, SE-1(2):156–173, June 1975.

[Gou83] John S. Gourlay. A mathematical framework for the investigation of testing. *IEEE Transactions on Software Engineering*, SE-9(6):686–709, November 1983.

[Gus78] F. Gustavson. Remark on algorithm 408. *ACM Transactions on Mathematical Software*, 4:295, 1978.

[Ham89] Dick Hamlet. Theoretical comparison of testing methods. In *Proceedings ACM SIGSOFT Third Symposium on Software Testing, Analysis, and Verification*, pages 28–37. ACM Press, December 1989.

[HT90] Dick Hamlet and Ross Taylor. Partition testing does not inspire confidence. *IEEE Transactions on Software Engineering*, 16(12):1402 – 1411, December 1990.

[McN71] John M. McNamee. Algorithm 408: A sparse matrix package (part I) [F4]. *Communications of the ACM*, 14(4), April 1971.

[MW93] Aditya P. Mathur and Weichen E. Wong. Comparing the fault detection effectiveness of mutation and data flow testing: An empirical study. Technical Report SERC-TR-146-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, March 1993.

[Off92] A. Jefferson Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering Methodology*, 1(1):5–20, January 1992.

[OPTZ94] A. J. Offutt, J. Pan, K. Tewary, and T. Zhang. Experiments with data flow and mutation testing. Technical Report ISSE-TR-94-105, Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia, 1994.

[PFTV86] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, 1986.

[RW85] Sandra Rapps and Elaine J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-14(4):367–375, April 1985.

[Wei89] Stewart N. Weiss. Comparing test data adequacy criteria. *Software Engineering Notes*, 14(6):42–49, October 1989.

[Wei90] Stewart N. Weiss. Methods of comparing test data adequacy criteria. In *COMPSAC 90*, pages 1–6, October 1990.

[WF91] Stewart N. Weiss and Phyllis G. Frankl. Comparison of all-uses and all-edges: Design, data, and analysis. Technical Report CS-91-03, Hunter College Computer Science Dept., March 1991.

[WHLM94] Weichen Eric Wong, Joseph R. Horgan, Saul London, and Aditya P. Mathur. Effect of test set minimization on the fault detection effectiveness of the all-uses criterion. Technical Report SERC-TR-152-P, Software Engineering Research Center, Purdue University, April 1994.

[WJ91] Elaine J. Weyuker and Bingchiang Jeng. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering*, 17(7):703–711, July 1991.

[WWH91] Elaine J. Weyuker, Stewart N. Weiss, and Richard Hamlet. Comparison of program testing strategies. In *Proceedings Fourth Symposium on Software Testing, Analysis, and Verification*, pages 1–10. ACM Press, October 1991.

A Appendix

Tables 3 and 4 show the estimated effectiveness and its confidence interval for various sized test sets and various coverage levels for each of the criteria. In addition they show the results of hypothesis testing, where the null hypotheses were of the form "x% mutation coverage has the same effectiveness as y% dua coverage for test sets of the given size". The first column shows test set size; the second and third columns show the fraction of coverage of inequivalent mutants and executable duas considered (Cov_m,Cov_u), respectively. The fourth and fifth columns give the measured effectiveness of test sets attaining at least these coverages($\hat{p}_m$,$\hat{p}_u$). The sixth column gives the effectiveness of the null criterion for test sets of this size ($\hat{p}_n$). The seventh and eighth columns give the number of test sets that achieved the given coverage levels, i.e., the sample sizes (N_m,N_u). The ninth and tenth columns give the total number of test sets of this size generated in attempting to obtain the sample (T_m,T_u). The eleventh column gives the verdict, also indicating which criterion was more effective in those cases where there was a significant difference. The last column shows the confidence intervals, first of mutation coverage, then of dua-coverage. For example, the first row compares test sets for `determinant` that killed at least 98.84% of the inequivalent mutants to those that covered at least 99.09% of the executable duas. None of 98.84%-mutation-adequate test sets exposed the fault, while 19% of the 99.09%-all-uses-adequate sets did. 1.62% of all the test sets of that size exposed the fault. 23 of 200,000

Size	Cov_m	Cov_u	$\hat{p}_m$	$\hat{p}_u$	$\hat{p}_n$	N_m	N_u	T_m	T_u	Decision	Conf. Intervals
`determinant`											
20	0.9884	0.9909	0.00	0.19	0.02	23	134	200K	100K	mut<dua	[0.00,0.18]vs.[0.10,0.27]
20	0.9884	0.9863	0.00	0.15	0.02	23	609	200K	100K	mut<dua	[0.00,0.18]vs.[0.11,0.18]
30	0.9913	0.9954	0.00	0.18	0.02	26	112	150K	150K	mut<dua	[0.00,0.16]vs.[0.09,0.27]
30	0.9913	0.9909	0.00	0.18	0.02	26	718	180K	150K	mut<dua	[0.00,0.16]vs.[0.14,0.21]
40	0.9930	0.9954	0.12	0.14	0.03	25	220	100K	100K	same	[0.00,0.29]vs.[0.08,0.20]
40	0.9930	0.9909	0.12	0.17	0.03	25	1052	100K	100K	same	[0.00,0.29]vs.[0.14,0.20]
`find1`											
20	0.9976	1.0	0.98	0.74	0.75	123	794	3M	1K	mut>dua	[0.95,1.00]vs.[0.70,0.78]
20	0.9976	0.9895	0.98	0.74	0.75	123	972	3M	1K	mut>dua	[0.95,1.00]vs.[0.70,0.78]
25	1.0	1.0	1.00	0.81	0.82	19	862	3.5M	1K	mut>dua	[0.78,1.00]vs.[0.78,0.85]
25	0.9988	0.9894	1.00	0.80	0.82	66	996	3.5M	1K	mut>dua	[0.93,1.00]vs.[0.77,0.83]
30	1.0	1.0	1.00	0.84	0.87	27	934	1.5M	1K	mut>dua	[0.84,1.00]vs.[0.81,0.87]
30	0.9988	0.9894	0.98	0.85	0.87	89	997	1.5M	1K	mut>dua	[0.94,1.00]vs.[0.82,0.87]
40	1.0	1.0	1.00	0.94	0.94	249	973	2.1M	1K	mut>dua	[0.99,1.00]vs.[0.93,0.96]
`find2`											
10	0.9919	1.0	0.38	0.16	0.17	32	64514	200K	100K	same	[0.15,0.60]vs.[0.16,0.17]
10	0.9919	0.9902	0.38	0.17	0.17	32	76144	200K	100K	same	[0.15,0.60]vs.[0.16,0.17]
15	0.9953	1.0	0.33	0.23	0.24	40	89636	200K	100K	same	[0.13,0.52]vs.[0.23,0.24]
15	0.9953	0.9902	0.33	0.24	0.24	40	93878	200K	100K	same	[0.13,0.52]vs.[0.23,0.24]
20	0.9953	1.0	0.34	0.31	0.31	154	96980	150K	100K	same	[0.25,0.44]vs.[0.30,0.31]
20	0.9953	0.9902	0.34	0.31	0.31	154	98285	150K	100K	same	[0.25,0.44]vs.[0.30,0.31]
25	0.9965	1.0	0.45	0.36	0.37	44	49510	200K	50K	same	[0.26,0.65]vs.[0.36,0.37]
25	0.9965	0.9902	0.45	0.36	0.37	44	49727	200K	50K	same	[0.26,0.65]vs.[0.36,0.37]
30	0.9977	1.0	0.27	0.42	0.42	26	19935	100K	20K	same	[0.05,0.49]vs.[0.42,0.43]
30	0.9977	0.9902	0.27	0.43	0.42	26	19967	100K	20K	same	[0.05,0.49]vs.[0.42,0.43]
`matinv1`											
15	0.9955	0.9955	1.00	0.54	0.17	37	48	400K	200K	mut>dua	[0.88,1.00]vs.[0.36,0.73]
20	0.9962	0.9955	1.00	0.60	0.22	31	117	250K	200K	mut>dua	[0.86,1.00]vs.[0.48,0.72]
20	0.9955	0.9955	1.00	0.60	0.22	58	117	250K	200K	mut>dua	[0.92,1.00]vs.[0.48,0.72]
25	0.9972	0.9955	1.00	0.78	0.27	33	127	250K	100K	mut>dua	[0.87,1.00]vs.[0.68,0.87]
25	0.9955	0.9955	1.00	0.78	0.27	153	127	250K	100K	mut>dua	[0.97,1.00]vs.[0.68,0.87]
30	0.9977	1.0	1.00	1.00	0.32	35	8	200K	100K		[0.88,1.00]vs.[0.56,1.00]
30	0.9977	0.9955	1.00	0.80	0.32	35	247	200K	100K	mut>dua	[0.88,1.00]vs.[0.74,0.87]
35	0.9980	1.0	1.00	1.00	0.36	34	14	200K	100K		[0.87,1.00]vs.[0.72,1.00]
35	0.9980	0.9955	1.00	0.79	0.36	34	367	200K	100K	mut>dua	[0.87,1.00]vs.[0.73,0.84]
40	0.9982	1.0	1.00	1.00	0.40	39	17	200K	100K		[0.89,1.00]vs.[0.76,1.00]
40	0.9982	0.9955	1.00	0.82	0.40	39	548	200K	100K	mut>dua	[0.89,1.00]vs.[0.78,0.86]
`matinv2`											
5	1.0	1.0	0.38	0.07	0.07	8	97665	500K	100K	same	[0.00,0.82]vs.[0.07,0.07]
5	0.9991	1.0	0.43	0.07	0.07	2365	97665	500K	100K	mut>dua	[0.41,0.46]vs.[0.07,0.07]
10	1.0	1.0	0.61	0.13	0.13	80	49980	300K	50K	mut>dua	[0.47,0.75]vs.[0.12,0.13]
15	1.0	1.0	0.41	0.19	0.19	227	49999	200K	50K	mut>dua	[0.33,0.50]vs.[0.18,0.19]
20	1.0	1.0	0.56	0.24	0.24	265	30K	100K	30K	mut>dua	[0.48,0.64]vs.[0.23,0.25]
`strmatch1`											
8	1.0	1.0	1.00	1.00	0.23	38	1443	200K	10K		[0.89,1.00]vs.[1.00,1.00]
8	0.9972	0.9796	1.00	1.00	0.23	547	1586	200K	10K		[0.99,1.00]vs.[1.00,1.00]
10	1.0	1.0	1.00	1.00	0.28	108	2104	100K	10K		[0.96,1.00]vs.[1.00,1.00]
12	1.0	1.0	1.00	1.00	0.33	348	2746	100K	10K		[0.99,1.00]vs.[1.00,1.00]

Table 3: Comparisons of effectiveness at various coverage levels.

generated test sets achieved a mutation score greater than or equal to 98.84%, while 134 of 100,000 generated test sets covered at least 99.09% of the duas. The result of hypothesis testing is that the two criteria differ in effectiveness; Column 11 also indicates the direction of this difference. The last column shows that the probability that the effectiveness value for mutation is outside the range $[0.00, 0.18]$ is less than 1%, as is the probability that the effectiveness value for all-uses is outside the range $[0.10, 0.27]$.

Values of coverage levels x and y were chosen as follows: For a given subject and test set size, the largest values x_1 and y_1 are the largest coverage levels such that the number of test sets achieving that level (or higher) was sufficiently large to allow the normal approximation to be used in analyzing the results. We had originally hoped to be able to use $x_1 = y_1 = 100\%$ in every case, but because of the difficulty of satisfying the criteria (particularly mutation testing) we had to settle on smaller values, despite the fact that hundreds of thousands of test sets of each size were generated. In those cases where $x_1 < y_1$ we performed an additional test, comparing $x_1\%$ mutation coverage to $y_2\%$ dua coverage, where $y_2 < x_1$ and y_2 is as close as possible to x_1, given the discrete nature of the data points. Situations where $y_1 < x_1$ were treated analogously.

Size	Cov_m	Cov_u	$\hat{p}_m$	$\hat{p}_u$	$\hat{p}_n$	N_m	N_u	T_m	T_u	Decision	Conf. Intervals
`strmatch2`											
8	1.0	1.0	0.63	0.35	0.40	67	127	100K	5K	mut>dua	[0.47,0.78]vs.[0.24,0.46]
10	1.0	1.0	0.81	0.45	0.47	31	62	10K	1K	mut>dua	[0.62,0.99]vs.[0.29,0.61]
12	1.0	1.0	0.81	0.52	0.54	85	908	10K	10K	mut>dua	[0.70,0.92]vs.[0.48,0.56]
`textformat.r`											
14	1.0	1.0	0.62	1.00	0.61	765771	854	1000K	100K	mut<dua	[0.62,0.62]vs.[0.99,1.00]
16	1.0	1.0	0.66	1.00	0.66	404207	1122	500K	100K	mut<dua	[0.66,0.67]vs.[1.00,1.00]
18	1.0	1.0	0.71	1.00	0.71	423174	1524	500K	100K	mut<dua	[0.71,0.71]vs.[1.00,1.00]
20	1.0	1.0	0.74	1.00	0.74	437230	1992	500K	100K	mut<dua	[0.74,0.75]vs.[1.00,1.00]
22	1.0	1.0	0.78	1.00	0.78	449073	2371	500K	100K	mut<dua	[0.78,0.78]vs.[1.00,1.00]
`textformat.0`											
14	0.9967	1.0	0.54	1.00	0.61	52	854	1000K	100K	mut<dua	[0.36,0.72]vs.[0.99,1.00]
14	0.9967	0.9756	0.54	1.00	0.61	52	12487	1000K	100K	mut<dua	[0.36,0.72]vs.[1.00,1.00]
16	0.9978	1.0	0.52	1.00	0.66	54	1122	1000K	100K	mut<dua	[0.34,0.69]vs.[1.00,1.00]
16	0.9978	0.9756	0.52	1.00	0.66	54	15557	1000K	100K	mut<dua	[0.34,0.69]vs.[1.00,1.00]
18	0.9978	1.0	0.75	1.00	0.71	40	1524	500K	100K	mut<dua	[0.57,0.93]vs.[1.00,1.00]
18	0.9978	0.9756	0.75	1.00	0.71	40	19056	500K	100K	mut<dua	[0.57,0.93]vs.[1.00,1.00]
20	0.9989	1.0	0.86	1.00	0.74	28	1992	500K	100K	same	[0.69,1.00]vs.[1.00,1.00]
20	0.9989	0.9756	0.86	1.00	0.74	28	22646	500K	100K	same	[0.69,1.00]vs.[1.00,1.00]
22	0.9989	1.0	0.73	1.00	0.78	48	2371	500K	100K	mut<dua	[0.56,0.89]vs.[1.00,1.00]
22	0.9989	0.9756	0.73	1.00	0.78	48	26233	500K	100K	mut<dua	[0.56,0.89]vs.[1.00,1.00]
`transpose`											
12	0.9970	0.9890	0.95	0.24	0.25	41	340	400K	600K	mut>dua	[0.86,1.00]vs.[0.18,0.30]
12	0.9889	0.9890	0.38	0.24	0.25	102169	340	400K	600K	mut>dua	[0.38,0.39]vs.[0.18,0.30]
14	0.9976	0.9890	0.96	0.28	0.28	24	389	450K	400K	mut>dua	[0.85,1.00]vs.[0.22,0.34]
14	0.9889	0.9890	0.39	0.28	0.28	149780	389	450K	400K	mut>dua	[0.39,0.40]vs.[0.22,0.34]
16	0.9976	0.9890	0.94	0.32	0.31	31	457	300K	300K	mut>dua	[0.82,1.00]vs.[0.26,0.38]
16	0.9889	0.9890	0.41	0.32	0.31	122075	457	300K	300K	mut>dua	[0.41,0.41]vs.[0.26,0.38]
18	0.9980	0.9890	0.94	0.33	0.34	31	432	400K	200K	mut>dua	[0.82,1.00]vs.[0.27,0.39]
18	0.9889	0.9890	0.43	0.33	0.34	192097	432	400K	200K	mut>dua	[0.42,0.43]vs.[0.27,0.39]
20	0.9980	0.9890	0.91	0.34	0.38	45	603	300K	200K	mut>dua	[0.80,1.00]vs.[0.29,0.39]
20	0.9889	0.9890	0.45	0.34	0.38	163518	603	300K	200K	mut>dua	[0.44,0.45]vs.[0.29,0.39]
24	0.9982	1.0	1.00	0.38	0.43	44	21	300K	200K	mut>dua	[0.90,1.00]vs.[0.11,0.65]
24	0.9982	0.9890	1.00	0.41	0.43	44	1049	300K	200K	mut>dua	[0.90,1.00]vs.[0.37,0.45]
28	0.9986	1.0	1.00	0.29	0.48	38	24	300K	150K	mut>dua	[0.89,1.00]vs.[0.05,0.53]
28	0.9986	0.9890	1.00	0.44	0.48	38	1127	300K	150K	mut>dua	[0.89,1.00]vs.[0.40,0.48]
32	0.9988	1.0	1.00	0.35	0.53	30	37	300K	150K	mut>dua	[0.86,1.00]vs.[0.15,0.55]
32	0.9988	0.9890	1.00	0.53	0.53	30	1669	300K	150K	mut>dua	[0.86,1.00]vs.[0.50,0.56]

Table 4: Comparisons of effectiveness at various coverage levels.