Evaluation of The Cost of Alternate Mutation Strategies *

Aditya P. Mathur and Weichen E. Wong
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

Amongst the various testing strategies, mutation testing has been empirically found to be the most effective in detecting faults. However, mutation often imposes unacceptable demands on computing and human resources due to the large number of mutants that need to be compiled and executed on one or more test cases. In addition, the tester needs to examine many mutants and analyze these for possible equivalence with the program under test. For these reasons, mutation is generally regarded by the practicing test engineer as too expensive to use. As one significant component of the cost of mutation is the execution of mutants against test cases, we believe that the cost can be reduced dramatically by reducing the number of mutants that need to be examined. We report the results from a case study designed to investigate two alternatives to reduce the cost of mutation. The alternatives considered are: (1) *randomly selected x% mutation* and (2) *constrained mutation*. We provide experimental data indicating that both alternatives lead to test sets that distinguish a significant number of *all* mutants and provide high all-uses coverage.

Resumo em português

Entre as várias estratégias de teste existentes, "mutation testing" (teste por mutação) tem sido empiricamente demonstrado ser o mais efetivo em detectar falhas. Entretanto, "mutation testing" frequentemente impõe uma demanda inaceitável nos recursos computacionais e humanos, devido ao grande número de mutantes que necessitam ser compilados e executados em um ou mais casos de teste. Além disso, a pessoa responsável pelo teste precisa examinar vários mutantes e analisá-los em relação a uma possível equivalência com o programa sendo testado. Por estas razões, "mutation testing" é em geral considerado por engenheiros de teste como dispendioso demais para ser usado. Como um componente significativo do custo do "mutation testing" é a execução dos mutantes nos diversos casos de teste, nós acreditamos que o custo pode ser dramaticamente reduzido diminuindo-se o número de casos de teste que precisam ser examinados. Nós reportamos os resultados de um caso de teste projetado para investigar duas alternativas para a redução do custo de "mutation testing". As alternativas consideradas são: (1) *x% das mutações selecionadas aleatoriamente* e (2) *mutação restrita*. Apresentamos dados experimentais indicando que ambas alternativas resultam em conjuntos de teste que distinguem um número significativo de *todos* mutantes e fornecem alta cobertura para "todos os usos" ("all-uses").

*This research was supported in part by NSF award CCR-9102331 and Purdue Research Foundation Research Grant No. 6901999.

1 Introduction

Testing of program P involves, among other activities, the construction of test cases, execution of P against these test cases, and the observation of program behavior to determine its correctness. Once any faults found in the program have been discovered and removed, one has at hand a test set T consisting of one or more test cases on which P behaves as desired. One may evaluate the adequacy of T using one or more of several test adequacy criteria. If T is inadequate with respect to some criterion C, then one may add test cases to T in an effort to satisfy C. Thus, an adequacy criterion serves a dual purpose. One, it enables a tester to evaluate how good is a test set. Two, it aids the tester in constructing new test cases. Note that "improvement" of a test set by the addition of test cases in an effort to satisfy some adequacy criterion may or may not reveal any hidden fault.

When a test set is enhanced to satisfy a criterion, one can merely hope that the new test cases so generated reveal any faults. We also note that there may exist more than one, possibly infinite, test sets that satisfy a given adequacy criterion. Not all of these test sets may be good at revealing faults in the program. However, cost considerations may require the generation of only one adequate test set with respect to a given criterion.

Of the many criteria available for evaluating the adequacy of test sets, we are concerned with mutation based criteria [8]. However, the cost of mutation testing has been a major bottleneck in its use by practitioners. In an effort to reduce its cost, we examined two alternatives. These alternatives are termed (1) *randomly selected x% mutation criterion* and (2) *constrained mutation criterion* [17]. We often refer to any one of these two criteria as *alternate mutation* and to the traditional mutation as *complete mutation* or simply as *mutation*. The alternate criteria reduce the number of mutants to be executed by a machine and the number of mutants to be examined by a tester. For these criteria, we ask the question: what are costs and relative strengths of the two criteria? The cost is measured in terms of the size of a test set required to satisfy a criterion and the number of mutants that need to be examined. The strength is measured in terms of the ability of an adequate test set to distinguish non-equivalent mutants and cover feasible def-uses examined in data flow testing. These criteria are defined later in this paper.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of mutation based adequacy criteria used in the remainder of this paper. In this section we also argue why a comparison of the kind we have reported here is important and useful to a tester. In Section 3, we explain how we compare the cost and strength of various mutation based test adequacy criteria. Our experimental methodology is described in detail in Section 4. Data collected from experiments and its analysis appear in Section 5. Section 6 explains how a practicing tester can benefit from our study. Our conclusions appear in Section 7.

2 Mutation based test adequacy criteria

2.1 Mutation criterion

Let P denote the program under test with D as its input domain. Test cases are selected from the input domain. $T \subset D$ is a test set consisting of one or more test cases. Let m be a syntactically correct program obtained by making a syntactic change in P. m is known as a *mutant* of P. Let r be a rule according to which P is changed. r is also known as a *mutant operator*. There could potentially be an infinite number of mutant operators. However, to keep mutation testing [illegible] operators [2, 4, 7, 20]. Consider, for example, a mutant operator that generates two mutants of P by replacing a use of x by $x+1$ and $x-1$. When applied to a program containing the assignment statement $z := x+y$, this mutant operator will generate two mutants of P, one obtained by replacing this assignment by $z := x+1+y$ and the other by replacing this assignment by $z := x-1+y$.

The application of a set of mutant operators R to P results in a set of mutants $m_1, m_2, \ldots, m_n, n \geq 0$. Mutant m_i is considered *equivalent* to P if for all $t \in D$, $P(t) = m_i(t)$. When executed against a test case t, mutant m_i is considered *distinguished* from P if $P(t) \neq m_i(t)$. Unless distinguished, a non-equivalent mutant is considered *live*. One may obtain a variety of mutation based criteria by varying R. Hereafter, we refer to the mutation based criterion wrt[1] a given R simply as *mutation* criterion.

A test set T may be evaluated against the mutation criterion by executing each mutant against elements of T. The ratio of the number of mutants distinguished to the number of non-equivalent mutants is the *mutation score* of T for P. T is considered adequate wrt mutation criterion if the mutation score is 1.

Although mutation testing has been empirically found [1, 4, 22] to be the most effective of the software testing methods proposed in detecting faults, it suffers from the problem of performance. The complexity of mutation is $O(n^2)$, where n is the number of variable references in the program. The number of mutants generated can be prohibitively large even for programs that are less than 100K lines of code. Thus, it is generally believed in the testing community that mutation testing is too expensive to use in practice. Since the major computational cost of mutation testing is incurred when executing mutants against test cases, we believe that this cost can be reduced by reducing the number of mutants to be examined. Based on this premise, we propose *the randomly selected x% mutation* and *the constrained mutation* [17] criteria described below. These criteria are examined for their cost effectiveness and strength as compared to mutation testing.

2.2 Randomly selected x% mutation criterion

One mechanism to reduce the cost of mutation testing is to examine a small percentage, say x%, of randomly selected mutants of each mutant type and ignore the remaining mutants [2]. Earlier empirical investigations by Acree [1], Sayward, and Budd [4] have shown that even with test cases generated using only 10% of the mutants, no more than 1% of the non-equivalent mutants were overlooked. The encouraging results from these studies have led us to investigate the effects of varying the percentage of mutants selected between 10% and 100%. In this paper we present results for experiments conducted by varying the selected percentage from 10 to 40.

2.3 Constrained mutation criterion

Another mechanism to reduce the cost of mutation testing is constrained mutation [17]. Using constrained mutation one examines only a few specified types of mutants and ignores the others. For example, the constrained *abs/ror* mutation, studied in this paper, examines only the *abs* and *ror* mutants. The *abs* mutant operator generates mutants by replacing a use of x by $abs(x)$, $-abs(x)$, and $zpush(x)$[2] wherever possible. When applied to a program containing an assignment statement $z := x+y$, the *abs* operator generates three mutants obtained by replacing this assignment by, respectively, $z := abs(x)+y$, $z := -abs(x)+y$, and $z := zpush(x)+y$.

[1] wrt: with respect to

[2] A mutant containing $zpush(x)$ is considered distinguished if a test causes the value of x to be 0 when execution reaches the mutated expression.

The *ror* mutant operator generates mutants by replacing one relational operator by other relational operators. In addition, it also replaces each condition consisting of at least one relational operator by boolean constants *true* and *false*. For example, when applied to a program containing a predicate $if\ (x = 0)\ then$, this mutant operator generates the following seven conditions: $(x < 0)$, $(x \leq 0)$, $(x \neq 0)$, $(x > 0)$, $(x \geq 0)$, $(true)$, and $(false)$. More about these mutant operators may be found in Mothra[8].

One significant issue that arises while using constrained mutation is that of selecting a small set of mutant operators. A good selection of mutant operators can reduce the execution cost dramatically without significantly sacrificing the error detecting capability of mutation testing. We conducted our experiments using the *abs* and *ror* operators for the following reasons.

1. Significance of *abs* mutants:

   When the *abs* operator is applied to a program, mutants are generated at all points where an absolute value sign can be inserted. To distinguish such mutants, a *positive*, a *negative*, and a *zero* value is necessary for the mutated expression when execution reaches that point. This requirement forces a tester to select test cases from different parts of the input domain which may possibly make the program fail if errors exist. We notice that Acree [1] also investigated the *abs* operator.

2. Significance of *ror* mutants:

   When the *ror* operator is applied to a program, mutants are generated at all points where a predicate can be replaced by its alternatives. For example, a mutant can be generated by changing a $\leq$ operator in a predicate to a $<$ operator. To distinguish this mutant, it's necessary to have a test case satisfying the original predicate but not the mutated predicate, i.e., a test case satisfying the *equal* sign of the predicate is required. Thus, distinguishing *ror* mutants forces a tester to construct test cases which examine points on or near a predicate border. Such process has been shown [3, 4, 6] to be effective in revealing certain types of errors such as the domain errors defined by Howden [15].

3 Comparison methodology

Weyuker et al [23] critically examined different relationships among test adequacy criteria. They argue that *effectiveness*, *cost*, and *difficulty of satisfaction* are several meaningful bases against which test adequacy criteria should be compared. Effectiveness refers to the fault detection capability of a criterion. Cost refers to the *work* necessary to satisfy it. Difficulty of satisfaction, also known as the *subsumption* relationship [6] and the *strength* [16, 19], refers to the computation of scores on one adequacy criterion using adequate test sets of another criterion. In this work, we compare the cost of satisfying the mutation, $x\%$ mutation, and *abs/ror* mutation adequacy criteria. We also compare the relative *strengths* of these criteria.

3.1 The cost criteria

The cost of mutation testing can be measured in several ways. We selected two cost metrics. One is the number of test cases required to satisfy a criterion. As construction of each test case requires effort from a test case developer, this appears to be a reasonable cost metric. The second metric is the number of mutants to be examined. As mutants are to be executed on one or more test cases, a reduction in the number of mutants leads to a reduction in the time to execute them. It is also likely to result in a reduction in the time spent by a tester in examining mutants for possible equivalence. We use Equation (1) to compute the reduction in terms of the number of test cases wrt the mutation adequacy criterion. Similarly, the reduction in terms of the number of mutants to be examined wrt the mutation adequacy criterion is computed using Equation (2).

$$\left(1 - \frac{\text{average size of test sets adequate wrt alternate mutation}}{\text{average size of mutation adequate test sets}}\right) * 100\% \tag{1}$$

$$\left(1 - \frac{\text{total number of mutants examined when using alternate mutation}}{\text{total number of mutants examined in mutation}}\right) * 100\% \tag{2}$$

3.2 Strength of an adequacy criterion

Strength of an adequacy criterion can be measured in several ways. We selected two measures. One, the strength is measured in terms of the mutation scores. Two, the strength is measured in terms of the all-uses scores [21]. Mutation scores defined in Equation (3) measure the loss in strength in terms of the power to distinguish non-equivalent mutants wrt the mutation adequacy criterion when using alternate mutation. Similarly, the all-uses scores defined in Equation (4) measure the loss in strength in terms of the coverage of feasible all-uses wrt to the mutation adequacy criterion for alternate mutation.

$$\frac{\text{number of mutants distinguished by a test set}}{\text{total number of non-equivalent mutants}} * 100\% \tag{3}$$

$$\frac{\text{number of all-uses covered by a test set}}{\text{total number of feasible all-uses}} * 100\% \tag{4}$$

Supported by data, we argue in Section 5 that both *x*% and *abs/ror* mutation adequacy criteria have almost the same power in distinguishing non-equivalent mutants and covering feasible all-uses as the mutation adequacy criterion. On the other hand, *x*% and *abs/ror* mutation adequacy criteria provide significant reductions on both size and expense wrt the mutation adequacy criterion.

4 Experimental methodology

Our experiments were conducted using two testing tools, `Atac`[12, 13, 14] and `Mothra`[5]. `Atac` is a data flow coverage measurement tool for C programs. Given a program and a test set, `Atac` computes the block, decision, p-use, c-use, and all-uses coverage. `Mothra` is a mutation testing tool for Fortran 77 programs. Given a program and a test set, `Mothra` generates a set of mutants, executes them against test cases in the test set, and computes the mutation score. While using these tools it is the tester's responsibility to determine equivalent mutants and infeasible all-uses. As the two tools accept programs in different programming languages, we faced an additional task of preparing programs for input to these tools. Preparation of programs, generation of adequate test sets, and a few other components of the experiments conducted, are described below.

4.1 Program selection and preparation

We used a suite of four programs described below. Source listing of each of these programs can be found in [19].

- FIND: This program is based on the Pascal version of Hoare's find program [11] used by Frankl and Weiss [9] with all reported faults removed. FIND takes two inputs, an array a and an index f, and permutes the elements of a so that elements to the right of position f are greater than or equal to $a[f]$ and elements to the left of position f are less than or equal to $a[f]$.
- STRMAT1 and STRMAT2: Both programs [9], with all reported faults removed, take a text and a pattern of zero or more characters. If the pattern appears in the text then the position of the first occurrence of the pattern in the text is returned, otherwise a 0 is returned. Although both programs share the same specification, their structures are different. STRMAT1 has a *do while* loop whereas STRMAT2 accomplishes the same task with a *while* loop. This syntactic difference makes STRMAT1 and STRMAT2 behave as two distinct programs in terms of the number of feasible all-uses and non-equivalent mutants.
- TEXTFMT: The correct version of the text formatting program in Goodenough and Gerhart [10] was used with all seven reported faults fixed. Other routines in this program are based on the Pascal routines used by Frankl and Weiss [9]. TEXTFMT takes a text as input and formats it. The text can be viewed as a sequence of words separated by spaces, tabs, and new lines. The formatted output sequence of words is identical to the input sequence and satisfies the properties given by Gerhart et al [10].

The source of each of the above programs was available in Pascal. We translated these programs to Fortran 77 and C. Since the tools we used support different programming languages, substantial effort and care went into the program translation from one language to another. This translation involved rewriting the programs with as little modification as possible and testing them to make sure that both the Fortran 77 and C versions provided identical results as the original Pascal version. Only black box testing was used in this step.

4.2 Experiments

We generated mutants in our experiments using all mutant operators in Mothra except the *goto* mutation operator[3]. Depending on the number of mutants examined, experiments were labeled for reference as indicated in Table 1. Note that experiments M, A-G, and H were designed to investigate mutation, randomly selected $x\%$ mutation, and constrained *abs/ror* mutation criteria, respectively.

4.3 Adequate test set generation

For experiments labeled EXPT-M, 30 mutation adequate test sets were generated randomly. To be able to complete the experiments in reasonable time, we restricted the input domain of each program as shown in Table 2.

When random test case generation failed to generate a mutation adequate test set in spite of repeated trials, additional test cases were generated manually. There were two parameters that controlled the number of trials before the random test case generation process for one test set was abandoned. First, we wanted to limit the time to execute mutants on test cases so as to be able

[3]The *goto* mutants were excluded for a historical reason. In our another study [19] we conducted a similar comparison on the relative strength between the all-uses and mutation adequacy criteria. Since the C-version of our programs did not contain any *goto* statement, we decided that for a fair comparison, *goto* statements should be excluded from mutation.

Table 1: Experiment sets

Experiment	Mutants examined
EXPT-M	all generated by Mothra except *goto*
EXPT-A	randomly selected 10% of each mutant type
EXPT-B	randomly selected 15% of each mutant type
EXPT-C	randomly selected 20% of each mutant type
EXPT-D	randomly selected 25% of each mutant type
EXPT-E	randomly selected 30% of each mutant type
EXPT-F	randomly selected 35% of each mutant type
EXPT-G	randomly selected 40% of each mutant type
EXPT-H	*abs* and *ror* mutants

Table 2: Input domain for random test case generation

Program	Constraints
FIND	• $1 \leq array\ size \leq 10$ • $1 \leq index \leq array\ size$ • $array\ element \in \{x \mid 0 \leq x \leq 100, x\ is\ an\ integer\}$
STRMAT1	• $0 \leq text\ length \leq 4$ • $0 \leq pattern\ length \leq 4$ • $text\ element \in \{a, b\}$ • $pattern\ element \in \{a, b\}$
STRMAT2	• $0 \leq text\ length \leq 4$ • $0 \leq pattern\ length \leq 4$ • $text\ element \in \{a, b\}$ • $pattern\ element \in \{a, b\}$
TEXTFMT	• $0 \leq text\ length \leq 15$ • $text\ element \in$ {*all uppercase and lowercase letters, tab, space, newline*}

to complete the experiments in reasonable time. As the number of trials increased, the mutation score increased slowly. Second, there was a natural desire to decrease the number of mutants to be examined manually to save on human effort. The number of trials used in each experiment was determined as a reasonable balance between the need to reduce the execution time and human effort. In our experiments, 150, 250, 100, and 250 trials were used for the generation of each mutation adequate test set of `FIND`, `STRMAT1`, `STRMAT2`, and `TEXTFMT`, respectively. Details of these procedures can be found in [19]. We note that Weyuker [23] has also proposed random selection of test sets in such experiments for different data flow adequacy criteria.

Similar procedures were used to generate $x\%$ and *abs/ror* mutation adequate test sets. Note that (1) for *abs/ror* mutation, 30 adequate test sets were generated for each program, and (2) for randomly selected $x\%$ mutation, 5 adequate test sets were generated for each program and x where $x \in \{10, 15, 20, 25, 30, 35, 40\}$.

Test sets generated by human testers could have introduced bias in the results. This may happen, for example, when the testers are familiar with the programs used in the experiments and therefore generate test sets that favor one testing method over another. Second, there could be a large number of test sets that satisfy an adequacy criterion. Selecting only one of these may possibly lead to false conclusions. As indicated above, our study overcomes each of these weaknesses by generating multiple adequate test sets automatically for each criterion.

Each adequate test set generated contained no redundant test case. This implies that each test case in any adequate test set distinguishes at least one non-equivalent mutant. This requirement is intended to make a fair comparison among different criteria. In the absence of this requirement one may always generate a $x\%$ or *abs/ror* mutation adequate test set which is also mutation adequate by merely adding the test cases required to distinguish any live mutant.

Equivalent mutants were identified manually. However, the failure of random test case generation to generate a test case that distinguishes a mutant resulted in a fairly small set of mutants examined manually. The number of equivalent mutants for each program are listed in Table 3.

5 Experimental results and analysis

Table 3 lists the number of mutants and decisions for the programs used. Larger programs tend to generate larger number of mutants. We note that `FIND` has the largest number of non-equivalent mutants (854) and `TEXTFMT` contains the largest number of decisions (18). The smallest number of non-equivalent mutants (446) and decisions (12) are in `STRMAT2`. These metrics serve as indicators of the relative complexity of programs considered in our experiments.

Tables 4 and 5 contain the all-uses and mutation scores using mutation, $x\%$ mutation and *abs/ror* mutation adequate test sets, respectively. Scores in the above tables are computed as percentages. Table 6 lists the size of mutation, $x\%$ mutation, and *abs/ror* mutation adequate tests, respectively. Table 7 lists the number of mutants examined in mutation, $x\%$ mutation, and *abs/ror* mutation.

5.1 Comparison based on all-uses scores

From our experimental data and the summary in Table 4 we make the following observations:

(1) All-uses scores using mutation adequate test sets:

- For `FIND`, `STRMAT1`, and `STRMAT2`, each mutation adequate test set is all-uses adequate.

Table 3: Program size metrics

		FIND		STRMAT1		STRMAT2		TEXTFMT	
		number	percentage	number	percentage	number	percentage	number	percentage
mutants	equivalent	62	6.77%	126	20.10%	64	12.55%	126	13.43%
	non-equivalent	854	93.23%	501	79.90%	446	87.45%	812	86.57%
blocks	feasible	20	100%	17	89.47%	15	100%	24	100%
	infeasible	0	0%	2	10.53%	0	0%	0	0%
decisions	feasible	16	94.12%	14	87.50%	12	100%	18	100%
	infeasible	1	5.88%	2	12.50%	0	0%	0	0%
all-uses	feasible	98	82.35%	58	92.06%	58	87.88%	33	62.26%
	infeasible	21	17.65%	5	7.94%	8	12.12%	20	37.74%

Table 4: All-uses scores using mutation, $x\%$ mutation, and *abs/ror* mutation adequate test sets

Experiment[†]		FIND	STRMAT1	STRMAT2	TEXTFMT
EXPT-M	average	100.00	100.00	100.00	99.39
	std. dev.	0.00	0.00	0.00	1.23
EXPT-A	average	99.59	97.93	98.62	96.97
	std. dev.	0.56	1.89	1.89	2.14
EXPT-B	average	99.39	99.31	99.31	96.97
	std. dev.	0.56	1.52	1.54	3.03
EXPT-C	average	99.39	98.62	97.93	98.79
	std. dev.	0.56	1.89	3.08	1.66
EXPT-D	average	99.80	100.00	99.31	95.76
	std. dev.	0.46	0.00	1.54	1.66
EXPT-E	average	99.80	100.00	99.31	95.15
	std. dev.	0.46	0.00	1.54	1.66
EXPT-F	average	100.00	100.00	100.00	98.18
	std. dev.	0.00	0.00	0.00	1.66
EXPT-G	average	99.80	100.00	99.31	98.79
	std. dev.	0.46	0.00	1.54	1.66
EXPT-H	average	99.56	97.13	96.27	95.45
	std. dev.	0.51	3.85	3.54	1.91

[†]See Table 1 for experiment names.

Table 5: Mutation scores using mutation, $x\%$ mutation, and *abs/ror* mutation adequate test sets

Experiment[†]		FIND	STRMAT1	STRMAT2	TEXTFMT
EXPT-M	average	100.00	100.00	100.00	100.00
	std. dev.	0.00	0.00	0.00	0.00
EXPT-A	average	97.59	97.25	96.37	99.01
	std. dev.	0.97	1.20	1.75	0.66
EXPT-B	average	97.52	97.96	96.86	98.74
	std. dev.	0.15	0.79	1.81	0.53
EXPT-C	average	97.59	98.28	97.04	98.89
	std. dev.	0.29	0.41	1.83	0.41
EXPT-D	average	98.27	98.52	99.01	98.92
	std. dev.	0.27	0.23	0.38	0.53
EXPT-E	average	98.95	98.96	98.97	99.51
	std. dev.	0.22	0.09	0.38	0.23
EXPT-F	average	99.11	98.88	99.42	99.63
	std. dev.	0.10	0.18	0.20	0.23
EXPT-G	average	99.23	98.88	99.60	99.85
	std. dev.	0.21	0.11	0.19	0.16
EXPT-H	average	98.29	98.84	95.70	95.90
	std. dev.	0.45	0.57	2.20	1.33

[†]See Table 1 for experiment names.

Table 6: Size of mutation, $x\%$ mutation, and *abs/ror* mutation adequate test sets

Experiment[†]		FIND	STRMAT1	STRMAT2	TEXTFMT
EXPT-M	average	24.57	20.23	17.57	16.67
	std. dev.	3.30	1.79	2.11	2.14
EXPT-A	average	9.20	10.00	8.60	9.60
	std. dev.	0.45	2.35	1.52	2.07
EXPT-B	average	10.40	11.60	9.80	10.40
	std. dev.	0.89	1.52	2.05	0.55
EXPT-C	average	13.00	12.60	10.40	12.00
	std. dev.	1.22	1.52	2.07	1.58
EXPT-D	average	13.80	13.60	11.00	9.00
	std. dev.	2.05	2.41	1.87	1.22
EXPT-E	average	14.60	14.80	11.20	12.60
	std. dev.	1.34	2.17	2.17	1.34
EXPT-F	average	16.00	14.00	12.00	12.20
	std. dev.	1.41	1.22	0.71	2.86
EXPT-G	average	15.80	15.20	12.60	12.60
	std. dev.	0.84	1.48	2.79	1.14
EXPT-H	average	14.73	12.00	8.27	6.97
	std. dev.	1.68	1.58	1.36	1.27

[†]See Table 1 for experiment names.

Table 7: Mutants[†] examined in mutation, $x\%$ mutation, and *abs/ror* mutation

Experiment[‡]	FIND		STRMAT1		STRMAT2		TEXTFMT	
	number	percentage	number	percentage	number	percentage	number	percentage
EXPT-M	916	100.00%	627	100.00%	510	100.00%	938	100.00%
EXPT-A	94	10.26%	65	10.37%	50	9.80%	103	10.98%
EXPT-B	135	14.74%	95	15.15%	78	15.29%	155	16.52%
EXPT-C	184	20.09%	124	19.78%	101	19.80%	210	22.39%
EXPT-D	231	25.22%	157	25.04%	128	25.10%	267	28.46%
EXPT-E	278	30.35%	190	30.30%	154	30.20%	318	33.90%
EXPT-F	321	35.04%	220	35.09%	179	35.10%	363	38.70%
EXPT-G	368	40.17%	252	40.19%	205	40.20%	414	44.14%
EXPT-H	166	18.12%	125	19.94%	99	19.41%	135	14.39%

[†]Including both equivalent and non-equivalent mutants.
[‡]See Table 1 for experiment names.

- For **TEXTFMT** of the 30 mutation adequate test sets obtained, a total of 24 are all-uses adequate.

(2) All-uses scores using $x\%$ mutation adequate test sets:

- For **FIND**, **STRMAT2**, and **TEXTFMT** of the 40 $x\%$ mutation adequate test sets obtained, a total of 29, 32, and 11 are all-uses adequate, respectively,
- For **STRMAT1**, each 25%, 30%, 35%, and 40% mutation adequate test set is all-uses adequate.
- All-uses scores of test sets adequate wrt to $x\%$ mutation for **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT** are at least 98.97%, 96.55%, 93.10%, and 93.94%, respectively.

(3) All-uses scores using *abs/ror* mutation adequate test sets:

- For **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT** of the 30 *abs/ror* mutation adequate test sets obtained, a total of 17, 18, 6, and 2 are all-uses adequate, respectively.
- All-uses scores of test sets adequate wrt to *abs/ror* mutation for **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT** are at least 98.98%, 89.66%, 87.93%, and 93.94%, respectively.

The above observations indicate that the average all-uses scores using mutation adequate test sets are less than 5% higher than those obtained using $x\%$ mutation and *abs/ror* mutation adequate test sets, respectively.

From Table 4 and 6, we also find that the average all-uses scores using higher $x\%$ mutation adequate test sets may not necessarily be higher than those using lower $x\%$ mutation adequate test sets. An example of this occurs in **TEXTFMT** for which the average all-uses score in **EXPT-E** (95.15%) is lower than that in **EXPT-B** (96.67%) even though the percentage of mutants examined in **EXPT-E** (33.90%) is higher than that in **EXPT-B** (16.52%). In addition, we find that *abs/ror* mutation adequate test sets result in almost the same all-uses coverage as $x\%$ mutation adequate test sets regardless of the value of $x\%$.

5.2 Comparison based on mutation scores

From our experimental data and the summary in Table 5 we make the following observations:

(1) Mutation scores using $x\%$ mutation adequate test sets:

- For `FIND`, `STRMAT1`, and `STRMAT2`, (i) none of the $x\%$ mutation adequate test set is mutation adequate, (ii) 10%, 15%, and 20% mutation adequate test sets give a mutation score of at least 96.14%, 95.21%, 94.84%, respectively, and (iii) 25%, 30%, 35%, and 40% mutation adequate test sets provide more than 98% mutation coverage.
- For `TEXTFMT`, (i) 2 out of 5 40% mutation adequate test sets are mutation adequate, and (ii) all 40 $x\%$ mutation adequate test sets provide more than 98.15% mutation coverage.

(2) Mutation scores using *abs/ror* mutation adequate test sets:

- For none of the four programs is an *abs/ror* mutation adequate test set mutation adequate.
- For `FIND`, `STRMAT1`, `STRMAT2`, and `TEXTFMT`, each of the 30 *abs/ror* mutation adequate test sets gives a mutation score of at least 97.67%, 97.60%, 91.48%, and 92.73%, respectively.

The above observations indicate that $x\%$ and *abs/ror* mutation adequate test sets suffer less than 5% loss on average mutation scores. The result on $x\%$ mutation is similar to those given in the earlier studies by Acree [1] and Budd [4].

From Table 5 and 6, we also find that the average mutation scores using higher $x\%$ mutation adequate test sets may not necessarily be higher than those using lower $x\%$ mutation adequate test sets. An example of this occurs in `TEXTFMT` for which the average mutation score in `EXPT-D` (98.92%) is lower than that in `EXPT-A` (99.01%) even though the percentage of mutants examined in `EXPT-D` (28.46%) is higher than that in `EXPT-A` (10.98%). In addition, we find that *abs/ror* mutation adequate test sets give almost the same mutation coverage as $x\%$ mutation adequate test sets regardless of the value of x.

5.3 Size and expense comparison

Figures 1 and 2 contain the size and expense reduction computed using Equations (1) and (2), respectively, for $x\%$ and *abs/ror* mutation adequacy criteria. From these two figures we make the following observations:

(1) The use of *abs/ror* mutation adequacy criterion leads to at least an 80% expense reduction and a 40% to 58% size reduction.

(2) The use of $x\%$ mutation adequacy criterion results in a 60% to 90% expense reduction and a 24% to 63% size reduction.

(3) A larger expense reduction, i.e., fewer mutants need to be examined, may not guarantee a correspondingly larger size reduction. An example of this occurs in `TEXTFMT` for which the size reduction in `EXPT-B` (37.61%) is less than that in `EXPT-D` (46.01%) even though the expense reduction in `EXPT-B` (83.48%) is larger than that in `EXPT-D` (71.54%).

In summary, we find that both $x\%$ and *abs/ror* mutation adequacy criteria provide significant size and expense reduction wrt to the mutation adequacy criterion. Such reduction is achieved without any significant loss in the ability of the adequate test sets to distinguish non-equivalent mutants and [illegible] feasible [illegible]

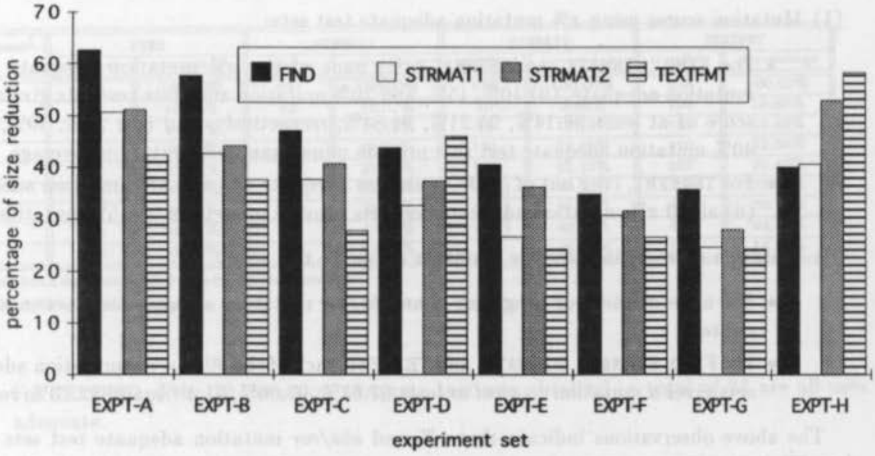

Figure 1: Size reduction for x% and *abs/ror* mutation adequacy criteria.

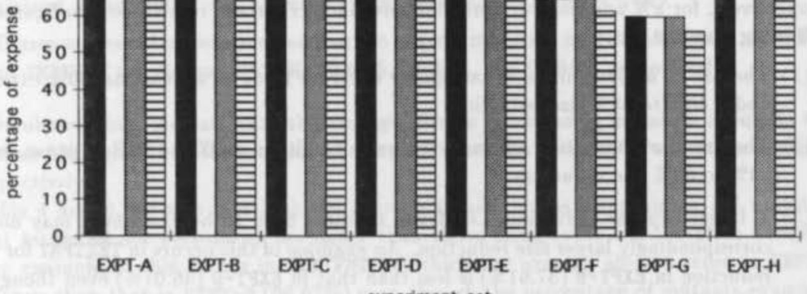

Figure 2: Expense reduction for x% and *abs/ror* mutation adequacy criteria.

6 Implications for a tester

It is important to examine the implications of our observations for a tester. First, we caution that our observations are made from a single case study. Thus, it may not be wise to generalize to all programs. Figure 3 presents a summary of the experimental results for STRMAT1[4]. We identify the following implications from this figure.

1. 10% random mutation leads to a small number of mutants to be examined by a tester and results in an average of 97.3% mutation score. This implies that a tester can do reasonably well in distinguishing all mutants by targeting a small number of randomly selected mutants.
2. The above implication also holds for *abs/ror* mutation.
3. A tester may hope to obtain a high all-uses score by targeting only a small percentage of mutants selected either randomly or selectively.

Each of the above observations imply that within tight budget and time constraints a tester may use random or constrained mutation testing and still hope to obtain a good test set wrt to complete mutation or the all-uses criteria. Another implication is that one need not examine all-uses to improve a test set. If a test set needs to be improved, one may examine additional live mutants and generate new test cases. If there are no additional live mutants then chances are that all all-uses have already been covered.

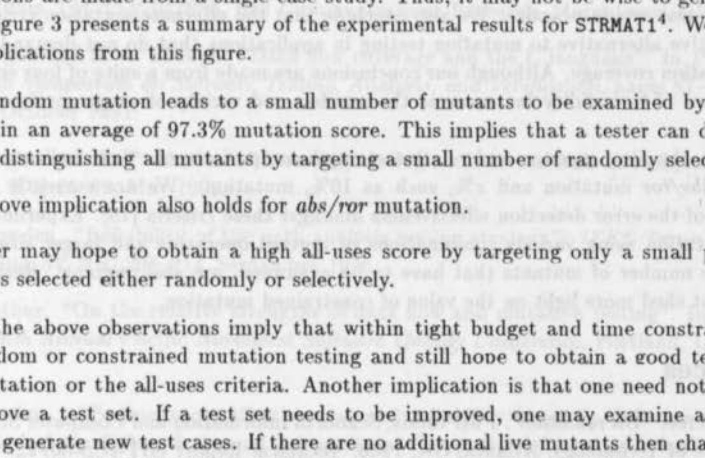

Figure 3: An overall comparison among mutation and its alternatives on STRMAT1

7 Conclusions

In this study we find that compared to mutation testing, both *randomly selected x% mutation* and *constrained mutation* provide significant reductions in terms of the number of test cases and

[4]For the remaining three programs in our suite of four programs, we obtained similar conclusions. However, due to space limitations, these data are not shown here.

the number of mutants to be examined. This gain is, however, accompanied by a small loss in the ability to distinguish non-equivalent mutants and cover feasible all-uses. These data suggest that examining only a small percentage of mutants may be a useful heuristic for evaluating and constructing test sets in practice. We also conclude that the alternate mutation criteria may serve as cost-effective alternative to mutation testing in applications that do not demand an ultra-high level of mutation coverage. Although our conclusions are made from a suite of four small programs, the confidence of its validity increases as the number and variety of programs for which it holds increases.

Based on the *size*, *expense*, and relative *strength* comparison, our data indicate no clear preference for *abs/ror* mutation and *x*%, such as 10%, mutation. We are currently working on a comparison of the error detection effectiveness amongst these criteria [18]. Experiments with constrained mutation using various combinations of mutant operators and larger size programs, in terms of the number of mutants that have to be examined, are also ongoing. Results from such studies might shed more light on the value of constrained mutation.

References

[1] A. T. Acree. *"On mutation"*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, 1980. Technical Report GIT-ICS-80/12.

[2] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. "Mutation analysis". Technical Report GIT-ICS-79/08, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, September 1979.

[3] 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.

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

[5] B. J. Choi, R. A. DeMillo E. W. Krauser, A. P. Mathur, R. J. Martin, A. J. Offutt, H. Pan, and E. H. Spafford. "The Mothra toolset". In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, HI, January 1989.

[6] L. A. Clarke. "A system to generate test data and symbolically execute programs". *IEEE Trans. on Software Engineering*, SE-2(3):215–222, September 1976.

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

[8] R. A. DeMillo and A. J. Offutt. "Constraint-based automatic test data generation". *IEEE Trans. on Software Engineering*, 17(9):900–910, September 1991.

[9] P. G. Frankl and S. N. Weiss. "Is data flow testing more effective than branch testing: An empirical study". In *Proceedings Quality Week 1991*, San Francisco, 1991.

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