NORTH-HOLLAND

Reducing the Cost of Mutation Testing: An Empirical Study

W. Eric Wong
Bell Communications Research, Morristown, New Jersey

Aditya P. Mathur
Department of Computer Sciences, Purdue University, West Lafayette, Indiana

Of the various testing strategies, mutation testing has been empirically found to be effective in detecting faults. However, mutation often imposes unacceptable demands on computing and human resources because of 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 as too expensive to use. Because one significant component of the cost of mutation is the execution of mutants against test cases, we believe that this cost can be reduced dramatically by reducing the number of mutants that need to be examined. We report results from a case study designed to investigate two alternatives for reducing the cost of mutation. The alternatives considered are randomly selected x% mutation and constrained *abs/ror* mutation. We provide experimental data indicating that both alternatives lead to test sets that distinguish a significant number of nonequivalent mutants and provide high all-uses coverage.

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 removed, one has a test set T consisting of one or more test cases on which P behaves as desired. One can evaluate the adequacy of T by use of one or more of several test adequacy criteria. If T is inadequate with respect to some criterion C, then additional test cases can be added to T in an effort to satisfy C. Thus, an adequacy criterion serves a dual purpose. First, it enables a tester to evaluate how good is a test set. Second, 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 (DeMillo et al., 1978). 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: a randomly selected x% mutation criterion and a constrained *abs/ror* mutation criterion. We refer to either of these two criteria as *alternate mutation*. These 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

Address correspondence to W. Eric Wong, Bellcore, 445 South Street, Morristown NJ 07960.

J. SYSTEMS SOFTWARE 1995; 31:185–196

655 Avenue of the Americas, New York, NY 10010

0164-1212/95/$9.50
SSDI 0164-1212(94)00098-8

relative strengths of the two criteria? Cost is measured in terms of the number of test cases required to satisfy a criterion and the number of mutants that need to be examined. Strength is measured in terms of the ability to distinguish nonequivalent mutants and cover feasible all-uses.

The remainder of this article is organized as follows. Section 2 contains a brief overview of mutation-based test adequacy criteria. Section 3 explains how to compare the cost and strength of these criteria. Our experimental methodology is described in detail in Section 4. Data collected from experiments and resulting analyses appear in Section 5. The possible benefit of our study to a practicing tester is discussed in Section 6. Conclusions and ongoing work are presented in Section 7.

2. MUTATION-BASED TEST ADEQUACY CRITERIA

In this section, we provide an overview of mutation-based test adequacy criteria by presenting only the information required for an understanding of the rest of the article. More details can be found elsewhere (Acree, 1980; Budd, 1980; DeMillo et al., 1978; DeMillo and Offutt, 1991; Mathur, 1994). Let P denote the program under test with $\mathscr{D}$ as its input domain. Test cases are selected from the input domain. A test set $T \subseteq \mathscr{D}$ consists of one or more test cases.

2.1 Mutation Criterion

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 known as a *mutant operator*. There could potentially be an infinite number of mutant operators. To keep mutation testing practical, the set of mutant operators is kept small and consists of simple mutant operators (Agrawal et al., 1989; Budd et al., 1978; Choi et al., 1989; DeMillo et al., 1988). For example, consider a mutant operator that generates 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 generates two mutants of P, one obtained by replacing this assignment with $z := x + 1 + y$ and the other by replacing this assignment with $z := x - 1 + y$.

The application of a set of mutant operators MO 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 $t \in \mathscr{D}$, $P(t) = m_i(t)$. 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 nonequivalent mutant is considered *live*. One can obtain a variety of mutation-based criteria by varying MO. Hereafter, we refer to the mutation-based criterion with respect to the set of all 22 mutant operators in Mothra as the *mutation* criterion. The description of these mutant operators can be found in the Appendix.

A test set T can 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 nonequivalent mutants is the *mutation score* of T for P. T is considered adequate with respect to the mutation criterion if the mutation score is unity.

Although mutation testing has been empirically found to be effective in detecting faults (Acree, 1980; Budd, 1980, 1981; Budd et al., 1980; DeMillo et al., 1978; Walsh, 1985), it suffers from a 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 < 100 K lines of code. Thus, it is generally believed that mutation testing is too expensive to use in practice. Because a significant portion of the computational cost of mutation testing is incurred in executing mutants against test cases, this cost can be reduced by reducing the number of mutants to be examined. We therefore propose the randomly selected $x\%$ mutation, where x is a suitably chosen number, and the constrained *abs/ror* mutation criteria described below. These criteria are examined for their cost effectiveness and strength compared with mutation testing.

2.2 Randomly Selected $x\%$ Mutation Criterion

One mechanism for reducing the cost of mutation testing is to examine a small percentage, say $x\%$, of randomly selected mutants of each mutant type and ignore the others. Earlier empirical investigations by Acree (1980) and coworkers (1979), and Budd (1980) showed that even with test cases generated by use of only 10% of the mutants, no more than 1% of the nonequivalent 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 article, we present results for experiments conducted with $x \in \{10 + 5i | 0 \leq i \leq 6, i \in \text{Integer}\}$. We start with 10% to verify Acree's and Budd's results, then we increase x by 5 up to 40 to examine the cost

effectiveness and the relative strength of mutation testing using a different percentage of mutants. Hereafter, we refer to the randomly selected $x\%$ mutation as $x\%$ mutation.

2.3 Constrained *abs/ror* Mutation Criterion

Another mechanism for reducing the cost of mutation testing is constrained mutation (Mathur, 1991). Using constrained mutation, one examines only a few specified types of mutants and ignores the others. For example, the constrained *abs/ror* mutation studied here examines only the *abs* and *ror* mutants. The *abs* mutant operator generates mutants by replacing each use of x with $abs(x)$, $-abs(x)$, and $zpush(x)$ wherever possible. The terms $abs(x)$ and $-abs(x)$ represent the absolute value and the negative absolute value of x, respectively. A mutant containing $zpush(x)$ is considered distinguished if a test case causes the value of x to be zero when execution reaches the mutated expression. When applied to a program containing an assignment statement $z := x + 1$, the *abs* operator generates six mutants obtained by replacing this assignment with $z := abs(x) + 1$, $z := -abs(x) + 1$, $z := zpush(x) + 1$, $z := abs(x + 1)$, $z := -abs(x + 1)$, and $z := zpush(x + 1)$.

The *ror* mutant operator generates mutants by replacing each relational operator with other relational operators. In addition, it also replaces each condition consisting of at least one relational operator with the 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: "if $(x < 0)$ then," "if $(x \leq 0)$ then," "if $(x \neq 0)$ then," "if $(x > 0)$ then," "if $(x \geq 0)$ then," "if **(true)** then," and "if **(false)** then." Details of these mutant operators appear in the **Mothra** user's manual (DeMillo and Martin, 1987).

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 strength and fault-detecting capability of mutation testing. We conducted our experiments using the *abs* and *ror* operators for the following reasons.

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 are 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 of the expression that is mutated. Acree (1980) also investigated the *abs* operator.

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 is 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 that examine points on or near a predicate border. Such a process has been shown to be effective in revealing certain types of faults (Boyer, 1975; Budd, 1980; Clarke, 1976) such as the domain faults defined by Howden (1976).

Hereafter, we refer to the constrained mutation using *abs* and *ror* mutant operators as *abs/ror* mutation.

3. COMPARISON METHODOLOGY

Weyuker et al. (1991) critically examined different relationships among test adequacy criteria. They suggest 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 (Clarke et al., 1989) and the *strength* (Mathur, 1991a; Wong and Mathur, 1994), refer to the computation of scores in one adequacy criterion by use of adequate test sets of another criterion. In this article, 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 Cost Criterion

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. Construction of each test case requires effort from a test case developer, so this appears to be a reasonable cost metric. The second metric is the number of mutants to be examined. Because 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 required to execute them. It is also likely to reduce the time spent by a tester in examining mutants for possible equivalence. We use equations 1 and 2 to compute the cost reduction with respect to the mutation adequacy criterion in terms of the number of test cases required and the number of mutants examined, respectively:

$$\left(1 - \frac{\text{average size of test sets adequate with respect to alternate mutation}}{\text{average size of mutation-adequate test sets}}\right) * 100\% \quad (1)$$

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

3.2 Strength of Adequacy Criterion

The strength of an adequacy criterion can be measured in several ways. We selected two measures. In one, the strength is measured in terms of the mutation scores. In the other, the strength is measured in terms of the all-uses scores (Rapps and Weyuker, 1985). Mutation scores defined in equation 3 can be used to measure the loss in strength in terms of the power to distinguish nonequivalent mutants with respect to the mutation adequacy criterion when using alternate mutation. Similarly, the all-uses scores defined in equation 4 can be used to measure the loss in strength in terms of the coverage of feasible all-uses with respect to the mutation adequacy criterion for alternate mutation:

$$\frac{\text{number of mutants distinguished by a test set}}{\text{total number of nonequivalent mutants}} * 100\% \quad (3)$$

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

Supported by data, we argue in Section 5 that both *x%* and *abs/ror* mutation adequacy criteria have almost the same power to distinguish nonequivalent mutants and cover feasible all-uses as the mutation adequacy criterion. On the other hand, *x%* and *abs/ror* mutation adequacy criteria provide significant cost reductions in both the number of test cases required and the number of mutants examined with respect to the mutation adequacy criterion.

4. EXPERIMENTAL METHODOLOGY

We used two testing tools, ATAC (Horgan and London, 1991, 1992; Horgan and Mathur, 1992) and Mothra (Choi et al., 1989). 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.

Because 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 the four programs described below. A source listing of each of these programs can be found in Mathur and Wong (1993).

- **FIND.** This program is based on the Pascal version of Hoare's (1971) find program used by Frankl and Weiss (1991) 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 (Frankl and Weiss, 1991), 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 zero 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 nonequivalent mutants.
- **TEXTFMT.** The correct version of the text-formatting program in Goodenough and Gerhart (1975) was used with all seven reported faults fixed. Other routines in this program are based on the Pascal routines used by Frankl and Weiss (1991). **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 Goodenough and Gerhart (1975).

The source of each of these programs was available in Pascal. We translated these programs to Fortran 77 and C. Because 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 provide 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.[1] Depending on the number of mutants examined, experiments were labeled for reference as indicated in Table 1. Experiments M and 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 experiment M, 30 mutation-adequate test sets were generated randomly. Although it is possible for such a random test case generation process to generate a mutation-adequate test set if the entire input domain of the program under test is used, this process might be very time consuming. Test cases so generated might repeatedly distinguish the same set of mutants and not improve the mutation score. To be able to complete the experiments in a 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 by use of these input domains despite repeated trials, additional test cases were generated manually. Two parameters 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 required to execute mutants on test cases so as to be able 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 reduce 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 Wong and Mathur (1994).

Similar procedures were used to generate $x\%$ and *abs/ror* mutation-adequate test sets. For *abs/ror* mutation, 30 adequate test sets were generated for each program; for randomly selected $x\%$ mutation, 5 adequate test sets were generated for each program and x where $x \in \{10, 15, 20, 25, 30, 35, 40\}$. The same input domain restrictions listed in Table 2 were applied.

Test sets generated by human testers could have introduced bias in the results. This may happen, for

Table 1. Experiment Sets

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

Table 2. Input Domain for Random Test Case Generation

Program	Constraints
FIND	$1 \leq$ array size ≤ 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 ≤ 4
	$0 \leq$ pattern length ≤ 4
	test element $\in \{a, b\}$
	pattern element $\in \{z, b\}$
STRMAT2	$0 \leq$ text length ≤ 4
	$0 \leq$ pattern length ≤ 4
	text element $\in \{a, b\}$
	pattern element $\in \{a, b\}$
TEXTFMT	$0 \leq$ text length ≤ 15
	text element $\in$ {all uppercase and lowercase letters, tab, space, newline}

[1] The *goto* mutants were excluded for a historical reason. In another study we conducted a similar comparison on the relative strength between the all-uses and mutation adequacy criteria (Wong and Mathur, 1994). Because 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.

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.

During the generation of mutation, *x%* mutation, and *abs/ror* mutation-adequate test sets, a test case was discarded if it could not distinguish at least one nonequivalent mutant. This requirement is intended to make a fair comparison among different criteria. In the absence of this requirement, one can always generate an *x%* or *abs/ror* mutation-adequate test set that is also mutation adequate by merely adding the test cases required to distinguish any live mutant. Our generation method only guarantees that each consequent test case is not redundant in terms of distinguishing nonequivalent mutants. It does not reexamine whether the previous test cases are still necessary. Once a test case is included in a test set, it will not be excluded because of the inclusion of any new test cases.

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 is listed in Table 3.

5. EXPERIMENTAL RESULTS AND ANALYSIS

Table 3 lists the number of mutants and decisions for the programs used. **FIND** has the largest number of nonequivalent mutants (854), and **TEXTFMT** contains the largest number of decisions (18). The smallest number of nonequivalent mutants (446) and decisions (12) is 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 derived by use of mutation, *x%*, and *abs/ror* mutation-adequate test sets, respectively. Scores in these tables are computed as percentages. Table 6 lists the size of mutation, *x%*, 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 Strength Comparison Based on All-Uses Scores

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

All-uses scores derived by use of mutation-adequate test sets:

- For **FIND, STRMAT1**, and **STRMAT2**, each mutation-adequate test set is all-uses adequate.
- For **TEXTFMT**, of the 30 mutation-adequate test sets obtained, 24 are all-uses adequate.

All-uses scores derived by use of x% mutation-adequate test sets with $x \in \{10, 15, 20, 25, 30, 35, 40\}$:

- For **FIND, STRMAT2**, and **TEXTFMT**, of the 35 *x%* mutation-adequate test sets obtained, 24, 27, 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 with respect to *x%* mutation for **FIND, STRMAT1, STRMAT2**, and **TEXTFMT** are at least 98.97, 96.55, 93.10, and 93.94%, respectively.

All-uses scores derived by use of 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.

Table 3. Program Size Metrics

Program	Number of Mutants*		Number of Decisions†	
	Equivalent (%)	Nonequivalent (%)	Feasible (%)	Infeasible (%)
FIND	62 (6.77)	854 (93.23)	16 (94.12)	1 (5.88)
STRMAT1	126 (20.10)	501 (79.90)	14 (87.50)	2 (12.50)
STRMAT2	64 (12.55)	446 (87.45)	12 (100)	0 (0)
TEXTFMT	126 (13.43)	812 (86.57)	18 (100)	0 (0)

$$*\% = \frac{\text{number of equivalent (nonequivalent) mutants}}{\text{number of total mutants generated}} * 100$$

$$\dagger\% = \frac{\text{number of feasible (infeasible) decisions}}{\text{number of total decisions}} * 100$$

Table 4. Average All-Uses Scores Using Mutation, $x\%$ with $x \in \{10, 15, 20, 25, 30, 35, 40\}$, and *abs / ror* Mutation-Adequate Test Sets

Experiment	FIND	STRMAT1	STRMAT2	TEXTFMT
M	100.00	100.00	100.00	99.39
A	99.59	97.93	98.62	96.97
B	99.39	99.31	99.31	96.97
C	99.39	98.62	97.93	98.79
D	99.80	100.00	99.31	95.76
E	99.80	100.00	99.31	95.15
F	100.00	100.00	100.00	98.18
G	99.80	100.00	99.31	98.79
H	99.56	97.13	96.27	95.45

Table 5. Average Mutation Scores Using Mutation, $x\%$ with $x \in \{10, 15, 20, 25, 30, 35, 40\}$, and *abs / ror* Mutation-Adequate Test Sets

Experiment	FIND	STRMAT1	STRMAT2	TEXTFMT
M	100.00	100.00	100.00	100.00
A	97.59	97.25	96.37	99.01
B	97.52	97.96	96.86	98.74
C	97.59	98.28	97.04	98.89
D	98.27	98.52	99.01	98.92
E	98.95	98.96	98.97	99.51
F	99.11	98.88	99.42	99.63
G	99.23	98.88	99.60	99.85
H	98.29	98.84	95.70	95.90

Table 6. Average Size of Mutation, $x\%$ with $x \in \{10, 15, 20, 25, 30, 35, 40\}$, and *abs / ror* Mutation-Adequate Test Sets

Experiment	FIND	STRMAT1	STRMAT2	TEXTFMT
M	24.57	20.23	17.53	16.67
A	9.20	10.00	8.60	9.60
B	10.40	11.60	9.80	10.40
C	13.00	12.60	10.40	12.00
D	13.80	13.60	11.00	9.00
E	14.60	14.80	11.20	12.60
F	16.00	14.00	12.00	12.20
G	15.80	15.20	12.60	12.60
H	14.73	12.00	8.27	6.97

- All-uses scores of test sets adequate with respect to *abs/ror* mutation for **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT** are at least 98.98, 89.66, 87.93, and 93.94%, respectively.

These observations indicate that the average all-uses scores derived by use of mutation-adequate test sets are $< 5\%$ higher than those obtained by use of $x\%$ and *abs/ror* mutation-adequate test sets, respectively. The average all-uses scores derived by use of higher $x\%$ mutation-adequate test sets may not necessarily be higher than those derived by use of lower $x\%$ mutation-adequate test sets. An example of this occurs in **TEXTFMT**, for which the average all-uses score in experiment E (95.15%) is lower than that in experiment B (96.97%), even though the percentage of mutants examined in experiment E (33.90%) is higher than that in experiment B (16.52%). We also find that *abs/ror* mutation-adequate test sets result in almost the same all-uses coverage as $x\%$ mutation-adequate test sets for $x \in \{10, 15, 20, 25, 30, 35, 40\}$.

5.2 Strength Comparison Based on Mutation Scores

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

Mutation scores derived by use of $x\%$ mutation-adequate test sets with $x \in \{10, 15, 20, 25, 30, 35, 40\}$:

- For **FIND**, **STRMAT1**, and **STRMAT2**, none of the $x\%$ mutation-adequate test set is mutation adequate; 10, 15, and 20% mutation-adequate test sets give a mutation score of at least 96.14, 95.21, and 94.84%, respectively; 25, 30, 35, and 40% mutation-adequate test sets provide $> 98\%$ mutation coverage.
- For **TEXTFMT**, 2 of 5 40% mutation-adequate test sets are mutation adequate; all 35 $x\%$ muta-

Table 7. Number of Mutants (Equivalent and Nonequivalent) Examined in Mutation, $x\%$ with $x \in \{10, 15, 20, 25, 30, 35, 40\}$, and abs / ror Mutation

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

$$* \% = \frac{\text{number of mutants examined using alternate mutation}}{\text{number of mutants examined in mutation}} \times 100$$

tion-adequate test sets provide > 98.15% mutation coverage.

Mutation scores derived by use of 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 thirty *abs/ror* mutation-adequate test sets gives a mutation score of at least 97.67, 97.60, 91.48, and 92.73%, respectively.

These observations indicate that *x*% and *abs/ror* mutation-adequate test sets suffer < 5% loss on average mutation scores. The result of 10% mutation is similar to those given in the earlier studies by Acree (1980) and Budd (1980). The average mutation scores derived with higher *x*% mutation-adequate test sets may not necessarily be higher than those obtained with lower *x*% mutation-adequate test sets. An example of this occurs in **TEXTFMT,** for which the average mutation score in experiment D (98.92%) is lower than that in experiment A (99.01%), even though the percentage of mutants examined in experiment D (28.46%) is higher than that in experiment A (10.98%). We also find that *abs/ror* mutation-adequate test sets give almost the same mutation coverage as *x*% mutation-adequate test sets for $x \in \{10, 15, 20, 25, 30, 35, 40\}$.

5.3 Cost Comparison

Figures 1 and 2 contain the cost reductions in terms of the number of test cases required and the number of mutants examined, respectively, for *x*% with $x \in \{10, 15, 20, 25, 30, 35, 40\}$ and *abs/ror* mutation adequacy criteria. These reductions were computed by use of equations 1 and 2. From these two figures we make the following observations:

- The use of the *abs/ror* mutation adequacy criterion leads to at least an 80% reduction in the number of mutants examined and a 40–58% reduction in the number of test cases required.
- The use of the *x*% mutation adequacy criterion results in a 60–90% reduction in the number of mutants examined and a 24–63% reduction in the number of test cases required.
- A larger reduction in the number of mutants examined may not guarantee a corresponding larger reduction in the number of test cases required. An example of this occurs in **TEXTFMT,** for which the reduction in the number of test cases in experiment B (37.61%) is less than that in experiment D (46.01%), even though the reduction in the number of mutants in experiment B (83.48%) is larger than that in experiment D (71.54%).

In summary, we find that both the *x*% and *abs/ror* mutation adequacy criteria provide significant reduction with respect to the mutation adequacy criterion in terms of the number of test cases required and the number of mutants examined. Such reduction is achieved without any significant loss in the ability of the adequate test sets to distinguish nonequivalent mutants and cover feasible all-uses.

6. IMPLICATIONS FOR A TESTER

In this section we examine the implications of our observations. We caution, however, that such obser-

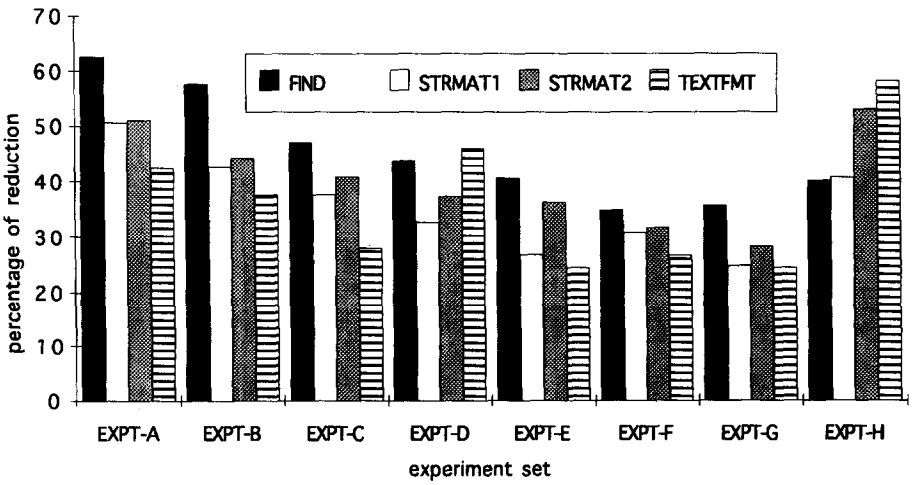

Figure 1. Cost reduction in terms of the number of test cases required for *x*% with $x \in \{10, 15, 20, 25, 30, 35, 40\}$ and *abs/ror* mutation-adequate test sets. EXPT, experiment.

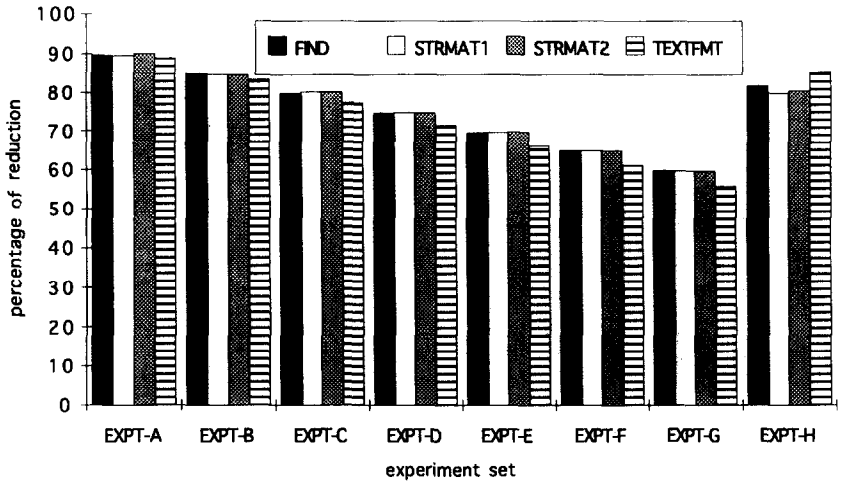

Figure 2. Cost reduction in terms of the number of mutants examined for $x\%$ with $x \in \{10, 15, 20, 25, 30, 35, 40\}$ and *abs/ror* mutation-adequate test sets. EXPT, experiment.

vations are made from a single case study; thus, they may not be used to arrive at general conclusions for programs. However, based on our results and those of others cited earlier, we considered any significant variation to be unlikely. Figures 3–6 present a summary of the experimental results with respect to mutation, 10% mutation, and *abs/ror* mutation for **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT**, respectively. We identify the following implications from these figures:

- Ten percent mutation leads to a small number of mutants to be examined by a tester and results in an average of 97.59, 97.25, 96.37, and 99.01% mutation scores on **FIND**, **STRMAT1**, **STRMAT2**, and **TEXTFMT**, respectively. This implies that a tester can do reasonably well in distinguishing nonequivalent mutants by targeting a small number of randomly selected mutants.
- The above implication also holds for *abs/ror* mutation.
- A tester can obtain a high all-uses score by targeting only a small percentage of mutants selected either randomly or selectively.

Although experimental results with respect to other $x\%$ mutation adequacy criteria for $x \in \{15, 20,$

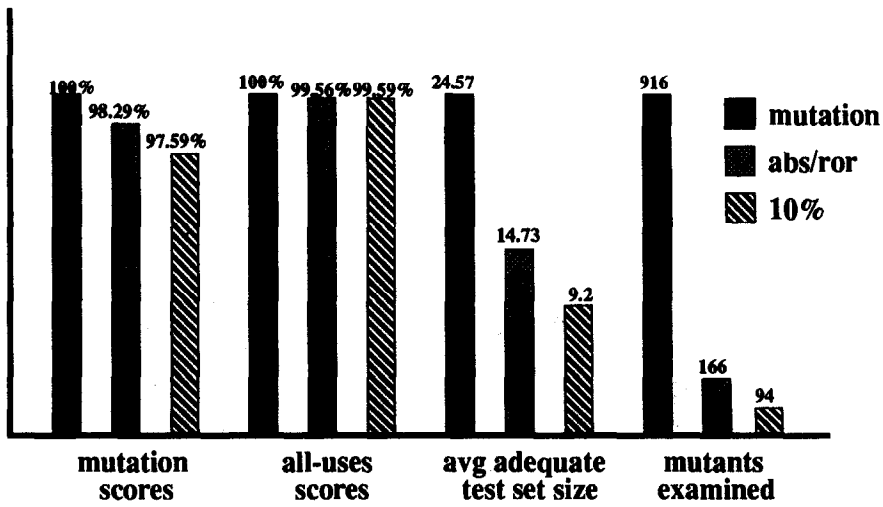

Figure 3. A comparison between mutation, 10% mutation, and *abs/ror* mutation on **FIND**.

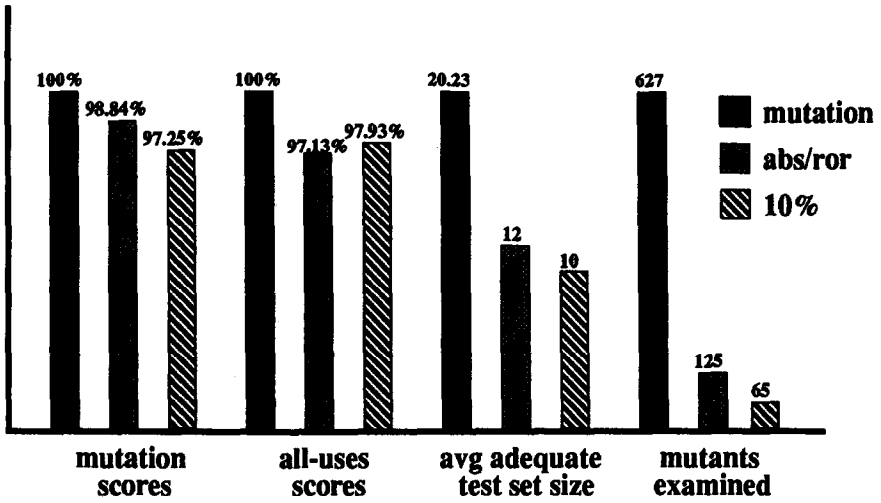

Figure 4. A comparison between mutation, 10% mutation, and *abs/ror* mutation on **STRMAT1**.

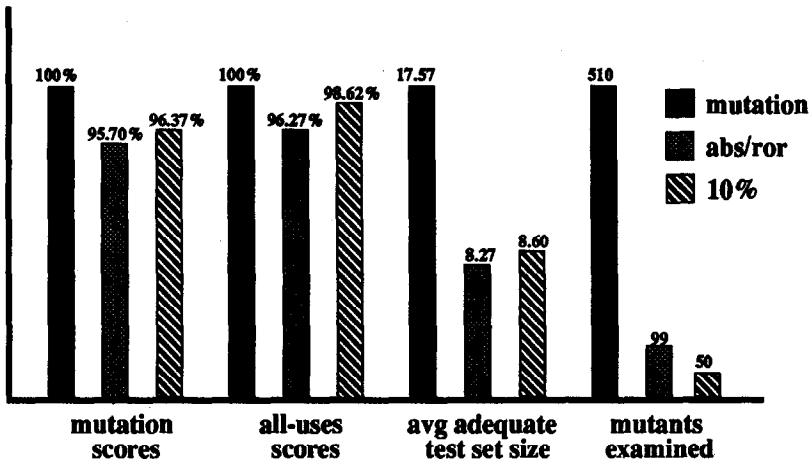

Figure 5. A comparison between mutation, 10% mutation, and *abs/ror* mutation on **STRMAT2**.

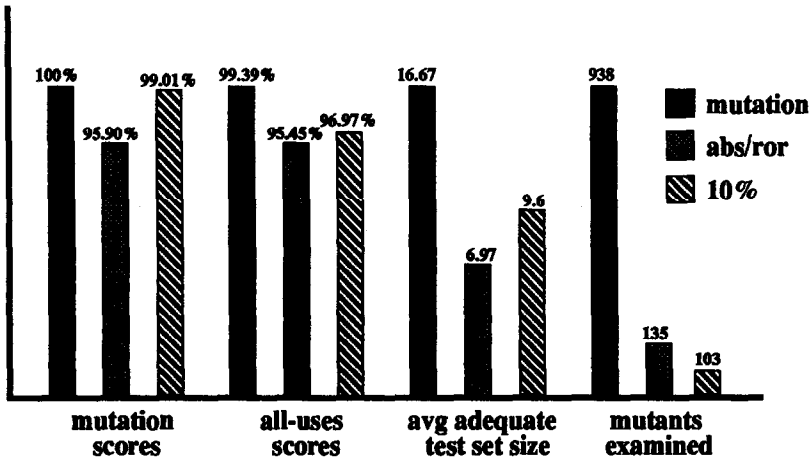

Figure 6. A comparison between mutation, 10% mutation, and *abs/ror* mutation on **TEXTFMT**.

25, 30, 35, 40} are not included in Figures 3–6 because of space limitations, they also support these implications. This implies that within tight budget and time constraints, a tester may use the randomly selected $x\%$ or constrained mutation testing and still obtain a good test set with respect to the mutation and all-uses criteria. Another implication is that for a given program P and a test set T, one may not need to examine the remaining uncovered all-uses in P to improve T. The test set T can be improved by generating only new test cases to distinguish the remaining live mutants in P. This is because if T distinguishes all nonequivalent mutants in P, then the probability is high that T also covers all feasible def-uses in P (Wong and Mathur, 1994).

7. CONCLUSIONS

In this study, we found that, compared with mutation testing, both the randomly selected $x\%$ mutation with $x \in \{10, 15, 20, 25, 30, 35, 40\}$ and the constrained *abs/ror* mutation provide significant reductions in terms of the number of test cases required and the number of mutants examined. This gain is, however, accompanied by a small loss in the ability to distinguish nonequivalent mutants and cover feasible all-uses. These data suggest that examining only a small percentage of the mutants may be a useful heuristic for evaluating and constructing test sets in practice. We conclude that these alternate mutations may serve as cost-effective alternatives to mutation testing in applications that do not demand an ultra-high-level of mutation coverage.

Based on cost and strength comparisons, our data indicate no clear preference for the *abs/ror* or $x\%$ mutation with $x \in \{10, 15, 20, 25, 30, 35, 40\}$. Experiments with constrained mutation using different combinations of mutant operators and larger sized programs in terms of the number of mutants to be examined are ongoing. Results from these studies may shed more light on the value of constrained mutation.

APPENDIX: Mothra Mutant Operators

Mutant Operator	Description	Example†
aar	Array reference for array reference replacement	$A(I) = B(J) + 3 \rightarrow A(I) = A(I) + 3$
abs	Absolute value insertion	$P = X * Y + 1 \rightarrow P = zpush(X) * Y + 1$
acr	Array reference for constant replacement	FOUND = 0 → FOUND = $A(I)$
aor	Arithmetic operator replacement	$A = B + C \rightarrow A = B - C$
asr	Array reference for scalar variable replacement	$P = X * A(I) \rightarrow P = X * A(A(I))$
car	Constant for array reference replacement	$P = X * A(I) \rightarrow P = X * 0$
cnr	Comparable array name replacement	$A(I) = B(J) + 3 \rightarrow A(I) = A(J) + 3$
crp	Constant replacement	DO 10 $I = 1, N$ → DO 10 $I = 1, N, 2$
csr	Constant for scalar variable replacement	$P = X * A(I) \rightarrow P = X * A(1)$
der	DO statement end replacement	DO 10$I = 1, N$ → *onetrip* 10$I = 1, N$
dsa	DATA statement alterations	DATA $X/0/ \rightarrow$ DATA $X/1/$
glr	GOTO label replacement	GOTO 20 → GOTO 10
lcr	Logical connector replacement	X .AND. $Y \rightarrow X$ .OR. Y
ror	Relational operator replacement	X .EQ. $A(I) \rightarrow X$ .GE. $A(I)$
rsr	RETURN statement replacement	$P = X * A(I) \rightarrow$ RETURN
san	Statement analysis	Each statement replaced by *trap*.
sar	Scalar variable for array reference replacement	$P = X * A(I) \rightarrow P = X * Y$
scr	Scalar for constant replacement	$P = 0 \rightarrow P = X$
sdl	Statement deletion	Each statement is deleted.
src	Source constant replacement	$X = 5 \rightarrow X = 1$
svr	Scalar variable replacement	$P = X * A(I) \rightarrow P = Y * A(I)$
uoi	Unary operator insertion	$X = 5 \rightarrow X = -1$

†$x \rightarrow y$ means that string x in P is replaced by string y to obtain a mutant.

REFERENCES

Acree, A. T., On Mutation, Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia, 1980.

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

Agrawal, H., DeMillo, R. A., Hathaway, R., Hsu, W., Hsu, W., Krauser, E. W., Martin, R. J., Mathur, A. P., and Spafford, E. H., Design of Mutant Operators for the C Programming Language, Technical Report SERC-TR-

41-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, 1989.

Boyer, R. S., Elspas, B., and Levitt, K. N., SELECT—A Formal System for Testing and Debugging Programs by Symbolic Execution, *Sigplan Not.* 10, 234–245 (1975).

Budd, T. A., Mutation Analysis of Program Test Data, Ph.D. Thesis, Yale University, New Haven, Connecticut, 1980.

Budd, T. A., Mutation analysis: Ideas, examples, problems and prospect, in *Computer Program Testing* (B. Chandrasekaran and S. Radicchi, eds.), North-Holland, Amsterdam, 1981.

Budd, T. A., DeMillo, R. A., Lipton, R. J., and Sayward, F. G., The design of a prototype mutation system for program testing, in *Proceedings* 1978 *National Computer Conference*, 1978.

Budd, T. A., DeMillo, R. A., Lipton, R. J., and Sayward, F. G., Theoretical and empirical studies on using program mutation to test the functional correctness of programs, in *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, 1980.

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

Clarke, L. A., A System to Generate Test Data and Symbolically Execute Programs, *IEEE Trans. Software Eng.* SE-2, 215–222 (1976).

Clarke, L. A., Podgurski, A., Richardson, D. J., and Zeil, S. J., A Formal Evaluation of Data Flow Path Selection Criteria, IEEE Trans. Software Eng. 15(11), 1318–1332 (1989).

DeMillo, R. A., and Offutt, A. J., Constraint Based Automatic Test Data Generation, *IEEE Trans. Software Eng.* 17(9), 900–910 (1991).

DeMillo, R. A., Lipton, R. J., and Sayward, F. G., Hints on Test Data Selection: Help for the Practicing Programmer, *IEEE Comp.* 11(4), 34–41 (1978).

DeMillo, R. A., Guindi, D. S., King, K. N., McCracken, W. M., and Offutt, A. J., An extended overview of the Mothra software testing environment, in *Proceedings of the Second Workshop on Software Testing, Verification and Analysis*, 1988, pp. 142–151.

DeMillo, R. A. and Martin, R. J., The Mothra Software Testing Environment User's Manual, Technical Report SERC-TR-4-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, 1987.

Frankl, P. G., and Weiss, S. N., Is data flow testing more effective than branch testing: An empirical study, in *Proceedings Quality Week 1991*, 1991.

Goodenough, J. B., and Gerhart, S. L., Toward a Theory of Test Data Selection, *IEEE Trans. Software Eng.* SE-1, 156–173 (1975).

Hoare, C., Proof of a Program: Find, *Commun. ACM* 14, 39–45 (1971).

Horgan, J. R., and London, S. A., Data flow coverage and the C language, in *Proceedings of the Fourth Symposium on Software Testing, Analysis, and Verification*, 1991, pp. 87–97.

Horgan, J. R., and London, S. A., ATAC: A data flow coverage testing tool for C, in *Proceedings of Symposium on Assessment of Quality Software Development Tools*, 1992, pp. 2–10.

Horgan, J. R., and Mathur, A. P., Assessing Testing Tools in Research and Education, *IEEE Software* 9, 61–69 (1992).

Howden, W. E., Reliability of the Path Analysis Testing Strategy, *IEEE Trans. Software Eng.* SE-2, 208–214 (1976).

Mathur, A. P., On the relative strengths of data flow and mutation testing, in *Proceedings of the Ninth Annual Pacific Northwest Software Quality Conference*, 1991.

Mathur, A. P., Performance, effectiveness, and reliability issues in software testing, in *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, 1991, pp. 604–605.

Mathur, A. P., Mutation testing, in *Encyclopedia of Software Engineering* (J. J. Marciniak, ed.) John Wiley and Sons Inc., 1994, pp. 707–713.

Mathur, A. P., and Wong, W. E., An Empirical Comparison of Mutation and Data Flow Based Test Adequacy Criteria, Technical Report SERC-TR-135-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, 1993.

Rapps, S., and Weyuker, E. J., Selecting Software Test Data Using Data Flow Information, *IEEE Trans. Software Eng.* SE-11, 367–375 (1985).

Walsh, P. J., A Measure of Test Case Completeness, Ph.D. Thesis, The State University of New York at Binghamton, Binghamton, New York, 1985.

Weyuker, E. J., Weiss, S. N., and Hamlet, R. G., Comparison of program testing strategies, in *Proceedings of the Fourth Symposium on Software Testing, Analysis and Verification*, 1991, pp. 154–164.

Wong, W. E., and Mathur, A. P., An Empirical Comparison of Data Flow and Mutation Based Test Adequacy Criteria, *J. Software Test. Verificat. Reliabil.*, 4(1), 9–31 (1994).