12-1-1999

Experiments to Assess the Cost-Benefits of Test-Suite Reduction

Gregg Rothermel
University of Nebraska-Lincoln, gerother@ncsu.edu

Mary Jean Harrold
Georgia Institute of Technology

Jeffery von Ronne
Oregon State University

Christie Hang
Ohio State University

Jeffery Ostrin
Oregon State University

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-1999-002; issued 12/1/1999

Technical Report GIT-99-29, College of Computing, Georgia Institute of Technology, December 1999

Experiments to Assess the Cost-Benefits of Test-Suite Reduction

Gregg Rothermel* Mary Jean Harrold† Jeffery von Ronne‡ Christie Hong§
Jeffery Ostrin¶

Abstract

Test-suite reduction techniques attempt to reduce the cost of saving and reusing test cases during software maintenance by eliminating redundant test cases from test suites. A potential drawback of these techniques is that in reducing a test suite they might reduce the ability of that test suite to reveal faults in the software. Previous studies suggested that test-suite reduction techniques can reduce test suite size without significantly reducing the fault-detection capabilities of test suites. To further investigate this issue we performed experiments in which we examined the costs and benefits of reducing test suites of various sizes for several programs and investigated factors that influence those costs and benefits. In contrast to the previous studies, our results reveal that the fault-detection capabilities of test suites can be severely compromised by test-suite reduction.

Keywords: software testing, test-suite reduction, test-suite minimization, empirical studies

1 Introduction

Because test development is expensive, software developers often save the test suites they develop, so that they can reuse those test suites later as their software undergoes maintenance. As the software evolves its test suites also evolve: new test cases are added to exercise new functionality or to maintain test adequacy. As a result, the sizes of test suites increase and the costs of managing and using those test suites increase. Therefore, researchers have investigated the notion that when several test cases in a test suite execute the same program components, that test suite can be reduced to a smaller suite that guarantees equivalent coverage. This research has produced several *test-suite reduction* algorithms (e.g., [2, 6, 8, 13]).

The motivation for test-suite reduction is straightforward: by reducing test-suite size, test-suite reduction techniques reduce the costs of executing, validating, and managing those test suites over future releases of the software. A potential drawback of test-suite reduction, however, is that the removal of test cases from a test suite may significantly alter the fault-detecting capabilities of that test suite. This tradeoff between the time required to execute, validate, and manage test suites, and the fault-detection effectiveness of test suites, is central to any decision to employ test-suite reduction.

Previous studies [19, 20, 21] suggest that test-suite reduction may produce dramatic savings in test-suite size, at little cost to the fault-detection effectiveness of those test suites. To further explore this issue we

*Department of Computer Science, Oregon State University, grother@cs.orst.edu
†College of Computing, Georgia Institute of Technology, harrold@cc.gatech.edu
‡Department of Computer Science, Oregon State University
§Department of Computer and Information Science, The Ohio State University
¶Department of Computer Science, Oregon State University

performed several experiments. In contrast to the previous studies, our experiments show that the fault-detection capabilities of test suites can be severely compromised by test-suite reduction.

The next section of this paper provides background information and reviews relevant literature. Section 3 describes our experiments, including their design, analysis, and results. Section 4 discusses our results and relates them to the results of previous studies. Section 5 presents conclusions.

2 Test-Suite Reduction Summary and Literature Review

2.1 Test-suite reduction and test-suite minimization

The test-suite reduction problem may be stated as follows [6, p. 272]:

> *Given*: Test suite T, a set of test-case requirements $r_1, r_2, \ldots, r_n$ that must be satisfied to provide the desired test coverage of the program, and subsets of T, $T_1, T_2, \ldots, T_n$, one associated with each of the r_is such that any one of the test cases t_j belonging to T_i can be used to test r_i.
>
> *Problem*: Find a representative set of test cases from T that satisfies all r_is.

The r_is in the foregoing statement can represent various test-case requirements, such as source statements, decisions, definition-use associations, or specification items.

A representative set of test cases that satisfies all of the r_is must contain at least one test case from each T_i; such a set is called a *hitting set* of the group of sets $T_1, T_2, \ldots, T_n$. To achieve a maximum reduction, it is necessary to find the smallest representative set of test cases. However, this subset of the test suite is the minimum cardinality hitting set of the T_is, and the problem of finding such a set is NP-hard [4]. Thus, most so-called "test-suite minimization" techniques resort to heuristics that do not always yield minimal sets. For this reason, we have chosen the more general terminology of *test-suite reduction* to describe such techniques.

Several test-suite reduction techniques have been proposed (e.g., [2, 6, 8, 13]); in this work we utilize the technique of Harrold, Gupta, and Soffa [6].

2.2 Previous empirical work

Many empirical studies of software testing have been performed. Some of these studies, such as those reported in References [3, 9, 18], provide indirect data about the effects of test-suite reduction through consideration of the effects of test-suite size on costs and benefits of testing. Other studies, such as the study reported in Reference [5], provide indirect data about the effects of test-suite reduction through a comparison of regression test selection techniques that do or do not attempt to select minimal test suites.[1]

Recent studies by Wong, Horgan, London, and Mathur [19, 20][2] and Wong, Horgan, Mathur, and Pasquini [21], however, directly examine the costs and benefits of test-suite reduction. We refer to these studies col-

[1] Whereas test-suite reduction considers a program and test suite, regression test selection considers a program, test suite, and modified program version, and selects test cases that are appropriate for that version without removing them from the test suite. The problems of regression test selection and test-suite reduction are thus related but distinct. For further discussion of regression test selection see Reference [16].

[2] Reference [20] (1998) extends work reported earlier in Reference [19] (1995); thus, we here focus on the most recent (1998) reference.

lectively as the "WHLMP" studies, and individually as the "WHLM" and "WHMP" studies. We summarize the results of these studies here; the references provide further details.

2.2.1 The WHLM study

The WHLM study [20] involved ten common C UNIX utility programs, including nine programs ranging in size from 90 to 289 lines of code, and one program of 842 lines of code. For each of these programs, the researchers used a random domain-based test case generator to generate an initial test-case pool; the number of test cases in these pools ranged from 156 to 997. In generating these pools, no attempt was made to achieve complete coverage of program components (blocks, decisions, or definition-use associations).

The researchers next drew multiple distinct test suites from their test-case pools by randomly selecting test cases. The resulting test suites achieved basic block coverages ranging from 50% to 95%; overall, 1198 test suites were generated. Reference [20] reports the sizes of the resulting test suites as averages over groups of test cases that achieved similar coverage: 270 test suites belonged to groups in which average test-suite size ranged from 9.07 to 33.73 test cases, and 928 test suites belonged to groups in which average test-suite size ranged from 1 to 4.43 test cases.

The researchers enlisted graduate students to inject simple mutation-like faults into each of the subject programs. The researchers excluded faults that could not be detected by any test case. All told, 181 faulty versions of the programs were retained for use in the study.

To assess the difficulty of detecting these faults, the researchers measured the percentages of test cases, in the associated test pools, that were able to detect the faults. Of the 181 faults, 78 (43%) were Quartile I faults detectable by fewer than 25% of the associated test cases, 42 (23%) were Quartile II faults detectable by between 25% and 50% of the associated test cases, 37 (20%) were Quartile III faults detectable by between 50% and 75% of the associated test cases, and 24 (13%) were Quartile IV faults detectable by at least 75% of the associated test cases.

The researchers reduced their test suites using ATAC [8], a tool based on an implicit enumeration algorithm that found exact minimization solutions for all of the test suites utilized in the study. Test suites were reduced with respect to block, decision, and all-uses data-flow coverage. The researchers measured the reduction in size and the reduction in fault-detection effectiveness of the reduced test suites as compared to the original test suites. The researchers also repeated this procedure on the entire test pools – effectively, treating these test pools as if they were test suites. Finally, they used null-hypothesis testing to determine whether the reduced test suites had fault-detection capabilities equal to test suites of the same size generated randomly from the unreduced test suites.

The researchers drew several conclusions from the study, including the following:

- As the coverage achieved by initial test suites increased, test-suite reduction produced greater savings with respect to those test suites, at rates ranging from 0% (for several of the 50-55% coverage suites) to 72.79% (for one of the 90-95% block coverage suites).
- As the coverage achieved by initial test suites increased, test-suite reduction produced greater losses in the fault-detection effectiveness of those suites. However, losses in fault-detection effectiveness were small compared to savings in test-suite size: in all but one case, losses were less than 7.27 percent, and most losses were less than 4.99 percent.
- Fault difficulty partially determined whether test-suite reduction caused losses in fault-detection effectiveness: Quartile I and II faults were more easily missed than Quartile III and IV faults following test-suite reduction.
- The null-hypothesis testing showed that test suites reduced by ATAC retain a size/effectiveness advantage over their corresponding randomly-reduced test suites.

The researchers generalized their results as follows:

> ....when the size of a test set is reduced while the coverage is kept constant, there is little or no reduction in its fault-detection effectiveness.... A test set which is minimized to preserve its coverage is likely to be as effective for detecting faults at a lower execution cost [20, page 368].

2.2.2 The WHMP study

Whereas the WHLM study examined test-suite reduction on 10 common Unix utilities, the WHMP study [21] involved a single C program developed for the European Space Agency as an interface to software that aids in the management of large antenna arrays. At 9,564 lines of code (6,218 executable), this program is several times the size of the largest program used in the WHLM study. Unlike the WHLM study, which used an initial pool of test cases generated randomly based solely on program specifications, the WHMP study used a pool of 1000 test cases generated based on an operational profile.

In the WHLM study, test suites were generated and categorized based on block coverage. In the WHMP study, two different procedures were followed for generating test suites: the first to create test suites of fixed size and the second to create test suites of fixed block coverage. For the fixed-size test suites, test cases were chosen randomly from the test pool until the desired number of test cases had been selected. In all, 120 test suites were generated in this manner: 30 distinct test suites for each of the target sizes of 50, 100, 150, 200. For the fixed-coverage test suites, test cases were chosen randomly from the test pool until the test suite reached the desired coverage. Only test cases that added coverage were added to the fixed-coverage test suites. In all, 180 test suites were generated in this manner: 30 distinct test suites for each of the target coverages ranging from 50% to 75% block coverage.

Whereas the faults in the WHLM study were injected by graduate students, the faults used in the WHMP study were obtained from an error log maintained during the creation of the program. The researchers selected eighteen of these faults, of which seventeen were detected by fewer than 7% of the test cases, making

them similar in detection difficulty to the "Quartile I" faults used in the WHLM study. The sixteenth fault was detected by 320 (32%) of the test cases.

As in the WHLM study, all of the test suites were reduced using ATAC. In both studies, the size of each test suite was reduced while the coverage was kept constant. In the WHMP study, however, reduction with respect to block coverage was the only reduction attempted. Reduction in test-suite size and in fault detection effectiveness were measured. Finally, null-hypothesis testing was used to compare test suites reduced for coverage to test suites that were randomly minimized.

The researchers drew the following overall conclusions from the study:

- There were substantial reductions in size achieved from reducing the fixed-size test suites. For the fixed-coverage test suites, reductions in size also occurred but were smaller.
- As in the WHLM study, the effectiveness losses of the reduced test suites were smaller than the size reductions, creating reduced test suites with a size/effectiveness advantage over the nonreduced test suites. The average effectiveness reduction due to test-suite reduction was less than 7.3%, and most reductions were less than 3.6%.
- The null-hypothesis testing again showed that reduced test suites retain a size/effectiveness advantage over their corresponding randomly-reduced test suites.

Thus, the WHMP study supports the findings of the WHLM study, while broadening the scope of the study in terms of both the programs under scrutiny and the types of initial test suites utilized.

3 Experiments

The WHLMP studies leave a number of open research questions, primarily concerning the extent to which the results observed in those studies generalize to other testing situations. Among the open questions are the following, which motivate the present work.

1. How does test-suite reduction fare in terms of costs and benefits when test suites have a wider range of sizes than the test suites utilized in the WHLMP studies?
2. How does test-suite reduction fare in terms of costs and benefits when test suites are coverage-adequate?
3. How does test-suite reduction fare in terms of costs and benefits when test suites contain additional coverage-redundant test cases?

The first and third questions are addressed by the WHLM study in its use of fixed-size test suites; however, that study examines only one program. Neither of the WHLMP studies considers the second question.

Test suites used in practice often contain test cases designed not for code coverage, but rather, designed to exercise product features, specification items, or exceptional behaviors. Such test suites may contain larger numbers of test cases, and larger numbers of coverage-redundant test cases, than the test suites utilized in the WHMP study, or than the coverage-based test suites utilized in the WHLM study.

Similarly, a typical tactic for utilizing coverage-based testing is to begin with a base of specification-based test cases, and add additional test cases to achieve complete coverage. Such test suites may also contain greater coverage-redundancy than the coverage-based test suites utilized in the WHLMP studies, but can be expected to distribute coverage more evenly than the fixed-size test suites constructed by random selection for the WHLM study.

It is important to understand the cost-benefit tradeoffs involved in minimizing such test suites. Thus, to investigate these tradeoffs, we performed a family of experiments.

3.1 Measures and Tools

We now discuss the measures and tools utilized in our experiments; subsequent sections discuss the individual experiments. Let T be a test suite, and let T_{min} be the reduced test suite that results from the application of a test-suite reduction technique to T.

3.1.1 Measures

We need to measure the costs and savings of test-suite reduction.

Measuring savings. Test suite reduction lets testers spend less time executing test cases, examining test results, and managing the data associated with testing. These savings in time are dependent on the extent to which test-suite reduction reduces test-suite size. Thus, to measure the savings that can result from test-suite reduction, we can follow the methodology used in the WHLMP studies and measure the reduction in test-suite size achieved by test-suite reduction. For each program, we measure savings in terms of the number and the percentage of test cases eliminated by test-suite reduction. (The former measure provides a notion of the magnitude of the savings; the latter lets us compare and contrast savings across test suites of varying sizes.) The *number of test cases eliminated* is given by $(\mid T \mid - \mid T_{min} \mid)$, and the *percentage of test cases eliminated* is given by $(\frac{|T|-|T_{min}|}{|T|} * 100)$.

This approach makes several assumptions: it assumes that all test cases have uniform costs, it does not differentiate between components of cost such as CPU time or human time, and it does not directly measure the compounding of savings that results from using the reduced test suites over a sequence of subsequent releases. This approach, however, has the advantage of simplicity, and using it we can draw several conclusions and compare our results with those achieved in the WHLMP studies.

Measuring costs. There are two costs to consider with respect to test-suite reduction. The first cost is the cost of executing a test-suite reduction tool to produce the reduced test suite. However, a test-suite reduction tool can be run following the release of a product, automatically and during off-peak hours, and in this case the cost of running the tool may be noncritical. Moreover, having reduced a test suite, the cost of test-suite reduction is amortized over the uses of that suite on subsequent product releases, and thus assumes progressively less significance in relation to other costs.

The second cost to consider is more significant. Test suite reduction may discard some test cases that, if executed, would reveal defects in the software. Discarding these test cases reduces the fault detection

effectiveness of the test suite. The cost of this reduced effectiveness may be compounded over uses of the test suite on subsequent product releases, and the effects of the missed faults may be critical. Thus, in this experiment, we focus on the costs associated with discarding fault-revealing test cases.

We considered two methods for calculating reductions in fault-detection effectiveness.

On a per-test-case basis: Given faulty program P and test suite T, one way to measure the cost of test-suite reduction in terms of effects on fault detection is to identify the test cases in T that reveal a fault in P but are not in T_{min}. This quantity can be normalized by the number of fault-revealing test cases in T. One problem with this approach is that multiple test cases may reveal a given fault. In this case some test cases could be discarded without reducing fault-detection effectiveness; this measure penalizes such a decision.

On a per-test-suite basis: Another approach is to classify the results of test-suite reduction, relative to a given fault in P, in one of three ways: (1) no test case in T is fault-revealing, and, thus, no test case in T_{min} is fault-revealing; (2) some test case in both T and T_{min} is fault-revealing; or (3) some test case in T is fault-revealing, but no test case in T_{min} is fault-revealing. Case 1 denotes situations in which T is ineffective. Case 2 indicates a use of test-suite reduction that does not reduce fault detection, and Case 3 captures situations in which test-suite reduction compromises fault detection.

The WHLMP experiments utilized the second approach; we do the same. For each program, we measure reduced effectiveness in terms of the number and the percentage of faults for which T_{min} contains no fault-revealing test cases, but T does contain fault-revealing test cases. More precisely, if F denotes the number of distinct faults revealed by T over the faulty versions of program P, and F_{min} denotes the number of distinct faults revealed by T_{min} over those versions, the *number of faults lost* is given by $(F - F_{min})$, and the *percentage reduction in fault-detection effectiveness* of test-suite reduction is given by $(\frac{F-F_{min}}{F} * 100)$.

Note that this method of measuring the cost of test-suite reduction calculates cost relative to a fixed set of faults. This approach also assumes that missed faults have equal costs, an assumption that typically does not hold in practice.

3.1.2 Tool infrastructure.

To perform our experiments we required several tools. First, we required a test-suite reduction tool; to obtain this, we implemented the algorithm of Harrold, Gupta and Soffa [6] within the Aristotle program analysis system [7]. The Aristotle system also provided data-dependence information for use in determining data-flow coverage, and code instrumenters for use in determining edge coverage.

3.2 Experiments with smaller C programs

Our first three experiments address our research questions on several small C programs similar in size to the C utilities utilized in the WHLM study. In this section we first describe details common to these three experiments, and then we report the results of the experiments in turn.

Program	Lines of Code	No. of Versions	Test Pool Size	Description
totinfo	346	23	1052	information measure
schedule1	299	9	2650	priority scheduler
schedule2	297	10	2710	priority scheduler
tcas	138	41	1608	altitude separation
printtok1	402	7	4130	lexical analyzer
printtok2	483	10	4115	lexical analyzer
replace	516	32	5542	pattern replacement

Table 1: Subject programs.

3.2.1 Subject programs, faulty versions, test cases, and test suites.

We used seven C programs as subjects (see Table 1). The programs range in size from 138 to 516 lines of C code and perform a variety of functions. Each program has several faulty versions, each containing a single fault. Each program also has a large test pool. The programs, versions, and test pools were assembled by researchers at Siemens Corporate Research for a study of the fault-detection capabilities of control-flow and data-flow coverage criteria [9]. We refer to these programs collectively as the "Siemens" programs.

The researchers at Siemens sought to study the fault-detecting effectiveness of coverage criteria. Therefore, they created faulty versions of the seven base programs by manually seeding those programs with faults, usually by modifying a single line of code. Their goal was to introduce faults that were as realistic as possible, based on their experience with real programs. Ten people performed the fault seeding, working "mostly without knowledge of each other's work" [9, p. 196].

For each of the seven programs, the researchers at Siemens created a large *test pool* containing possible test cases for the program. To populate these test pools, they first created an initial set of black-box test cases "according to good testing practices, based on the tester's understanding of the program's functionality and knowledge of special values and boundary points that are easily observable in the code" [9, p. 194], using the *category partition method* and the Siemens Test Specification Language tool [1, 14]. They then augmented this set with manually-created white-box test cases to ensure that each executable statement, edge, and definition-use pair in the base program or its control-flow graph was exercised by at least 30 test cases. To obtain meaningful results with the seeded versions of the programs, the researchers retained only faults that were "neither too easy nor too hard to detect" [9, p. 196], which they defined as being detectable by at least three and at most 350 test cases in the test pool associated with each program.

Figure 1 shows the sensitivity to detection of the faults in the Siemens versions relative to the test pools; the boxplots[3] illustrate that the sensitivities of the faults vary within and between versions, but overall are all lower than 19.77%. Therefore, all of these faults were, in the terminology of the WHLMP studies, Quadrant I faults, detectable by fewer than 25% of the test-pool inputs.

[3] A boxplot is a standard statistical device for representing data sets [11]. In these plots, each data set's distribution is represented by a box. The box's height spans the central 50% of the data and its upper and lower ends mark the upper and lower quartiles. The middle of the three horizontal lines within the box represents the median. The vertical lines attached to the box indicate the tails of the distribution.

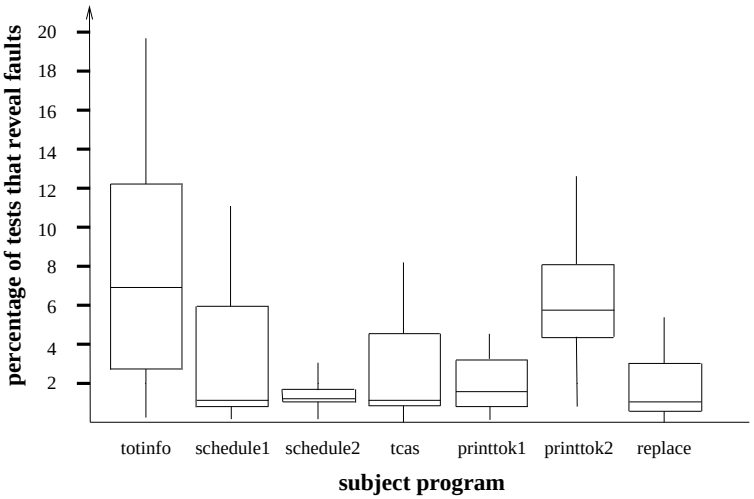

Figure 1: Boxplots that show, for each of the seven Siemens programs, the distribution, over the versions of that program, of the percentages of inputs in the test pools for the program that expose faults in that version.

To investigate our research questions we required coverage-adequate test suites that exhibit redundancy in coverage, and we required these in a range of sizes. To create these test suites we utilized two criteria: *edge coverage* and *all-uses data-flow coverage*. The edge coverage criterion is similar to the decision coverage criterion used in the WHLM study, but is defined on control flow graphs.[4] The all-uses data-flow coverage criterion involves testing each definition in the program to each use that it may reach (in the program's control flow graph) [10, 12, 15].

We used the Siemens test pools to obtain the various edge-coverage-adequate and all-uses data-flow-coverage-adequate test suites for each subject program. Our test suites consist of a varying number of test cases selected randomly from the associated test pool, together with additional test cases required to achieve 100% coverage of coverable edges.[5] We did not add any particular test case to any particular test suite more than once. To ensure that these test suites would possess varying ranges of coverage redundancy, we randomly varied the number of randomly-selected test cases over sizes ranging from 0 to .5 times the number of lines of code in the program. Altogether, we generated 1000 test suites for each program.

Figure 2 provides views of the range of sizes of test suites created by the process just described. The boxplots illustrate that for each subject program, our test-suite generation procedure yielded a collection of test suites of sizes that are relatively evenly distributed across the range of sizes utilized for that program. The all-uses-coverage-adequate suites are larger on average than the edge-coverage-adequate suites because in general, more test cases are required to achieve all-uses coverage than to achieve edge coverage.

[4] A test suite T is *edge-coverage adequate* for program P iff, for each edge e in each control flow graph for some procedure in P, if e is dynamically exercisable, then there exists at least one test case $t \in T$ that exercises e. A test case t *exercises* an edge $e = (n_1, n_2)$ in control flow graph G if t causes execution of the statement associated with n_1, followed immediately by the statement associated with n_2.

[5] To randomly select test cases from the test pools, we used the C pseudo-random-number generator "rand", seeded initially with the output of the C "time" system call, to obtain an integer which we treated as an index i into the test pool (modulo the size of that pool).

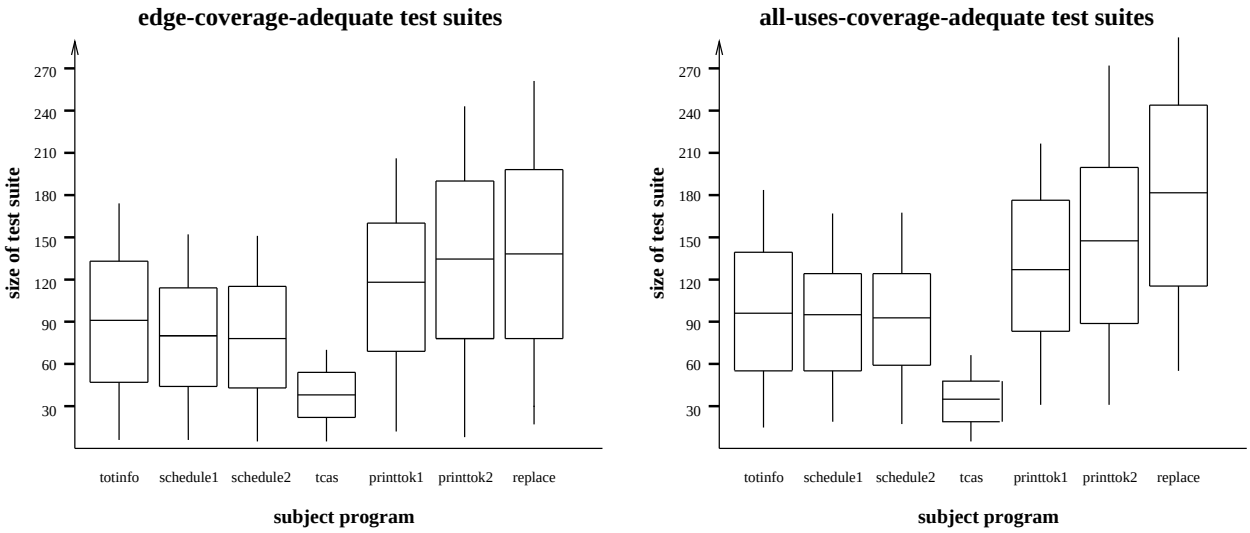

Figure 2: Boxplots that show, for each of the seven Siemens programs, the distribution of sizes among the unreduced edge-coverage-adequate test suites for that program (left) and the distribution of sizes among the unreduced all-uses-coverage-adequate test suites for that program (right).

Analysis of the fault-detection effectiveness of these test suites shows that, except for eight of the edge-coverage-adequate test suites for `schedule2`, each test suite revealed at least one fault in the set of faulty versions of the associated program. Thus, although each fault individually is difficult to detect relative to the entire test pool for the program, almost every test suite utilized in the study possessed at least some fault-detection effectiveness relative to the set of faulty programs utilized.

3.2.2 Experiment design.

The experiments were run using a full-factorial design with 1000 size-reduction and 1000 effectiveness-reduction measures per cell.[6] The independent variables manipulated were:

- The subject program (the seven programs, each with a variety of faulty versions).
- Test suite size (for a program of n lines of code, between 0 and $n/2$ test cases randomly selected from the test pool, together with additional test cases as necessary to achieve code coverage).

For each subject program, we applied test-suite reduction techniques to each of the test suites for that program. We then computed the size and effectiveness reductions for these test suites.

3.2.3 Threats to validity.

In this section, we discuss potential threats to the validity of our experiments with the Siemens programs.

Threats to internal validity are influences that can affect the dependent variables without the researcher's knowledge, and that thus affect any supposition of a causal relationship between the phenomena underlying

[6] The single exception involved `schedule2`, for which only 992 measures were available with respect to edge-coverage-adequate test suites, due to exclusion of the eight test suites that did not expose any faults.

the independent and dependent variables. In these experiments, our greatest concerns for internal validity involve the fact that we do not control for the structure of the programs or the locality of changes.

Threats to external validity are conditions that limit our ability to generalize our results. The primary threats to external validity for this study concern the representativeness of the artifacts utilized. The Siemens programs, though nontrivial, are small, and larger programs may be subject to different cost-benefit tradeoffs. Also, there is exactly one seeded fault in each Siemens program; in practice, programs have much more complex error patterns. Furthermore, the faults in the Siemens programs were deliberately chosen (by the Siemens researchers) to be faults that were relatively difficult to detect. (However, the fact that the faults in these programs were not chosen by us does eliminate one potential source of bias.) Finally, the test suites we utilized represent only two types of test suite that could occur in practice if a mix of non-coverage-based and coverage-based testing were utilized. These threats can be addressed only by additional studies utilizing a wider range of artifacts.

Threats to construct validity arise when measurement instruments do not adequately capture the concepts they are supposed to measure. For example, in this experiment our measures of cost and effectiveness are very coarse: they treat all faults as equally severe, and all test cases as equally expensive.

3.2.4 Experiment 1: Reduction of edge-coverage-adequate test suites

Our first experiment addresses our research questions by applying test-suite reduction techniques to the Siemens programs and their edge-coverage-adequate test suites. In reporting results, we first consider test-suite size reduction, and then we consider fault-detection effectiveness reduction.

Test suite size reduction

Figure 3 depicts the relation between the sizes of the reduced edge-coverage-adequate test suites for the seven Siemens programs and the sizes of the original test-suites. The data for each program P is depicted by a scatterplot containing a point for each of the test suites utilized for P; the points plot sizes of test suites for edge coverage (vertical axis) versus sizes of original test suites (horizontal axis). Solid lines indicate the average reduced test-suite size across the range of original test-suite sizes, computed as running averages over each set of fifty consecutive points. As the figure shows, the average sizes of the reduced test suites ranges from approximately five (for `tcas`) to twelve (for `replace`). For each program, the reduced test suites demonstrate little variance in size: `tcas` exhibiting the least variance (between four and five test cases), and `printtok1` showing the greatest variance (between five and fourteen test cases). Considered across the range of original test-suite sizes, reduced test-suite size for each program is relatively stable.

Figure 4 depicts the percentage reduction in test-suite size produced by test-suite reduction for each of the subject programs. The data for each program P is represented by a scatterplot containing a point for each of the test suites utilized for P; each point shows the percentage size reduction achieved for a test suite versus the size of that test suite prior to test-suite reduction. Visual inspection of the plots indicates an initial sharp increase in test-suite size reduction, tapering off as size increases. The data gives the impression of fitting a hyperbolic curve.

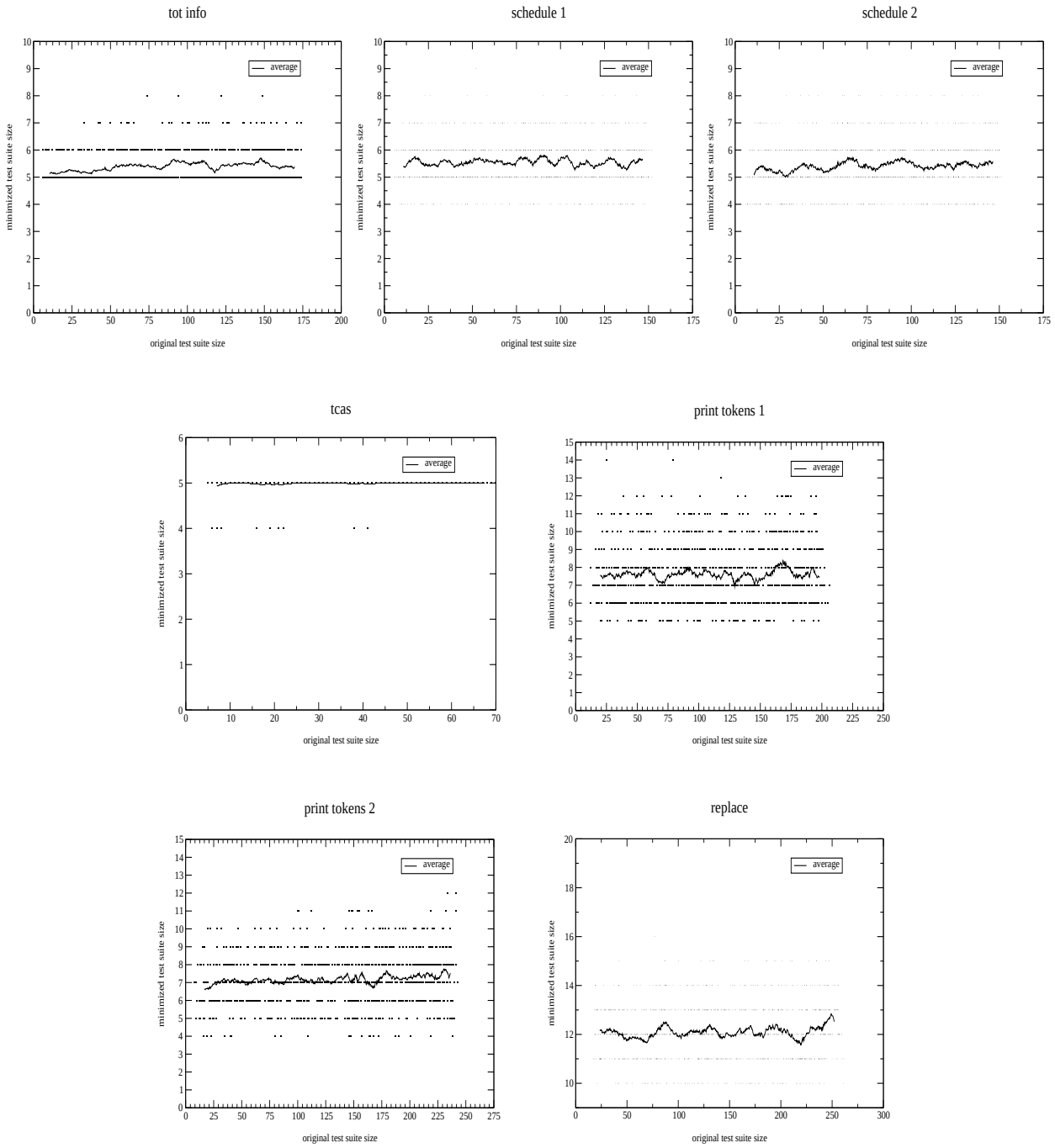

Figure 3: Sizes of test suites reduced for edge coverage versus sizes of original test suites, for edge-coverage-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote sizes of reduced test suites. Average reduced test-suite size across the range of original test-suite sizes (computed as running averages over each set of fifty consecutive points) is denoted by the solid lines.

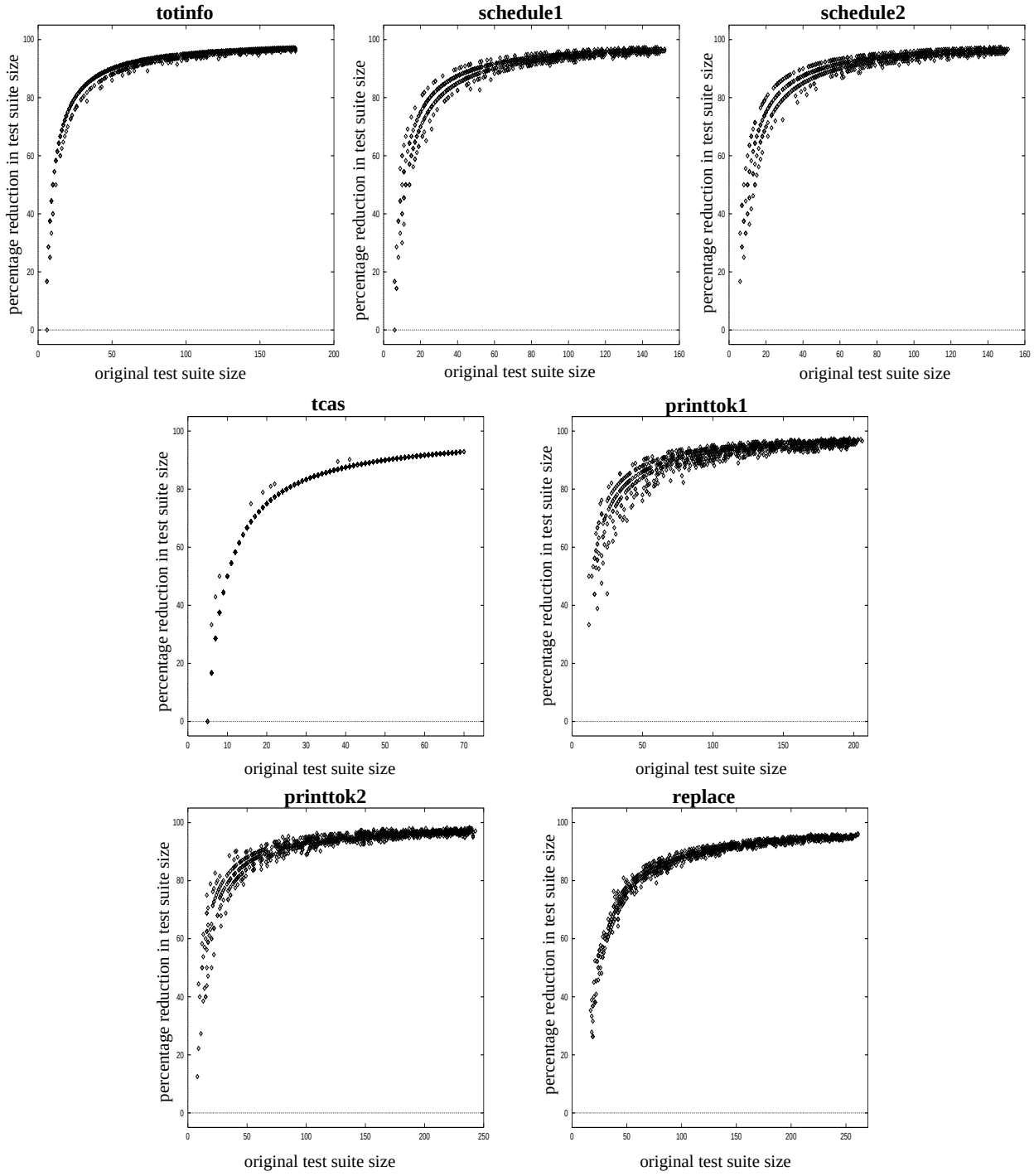

Figure 4: Percentage reduction in test-suite size as a result of test-suite reduction versus sizes of original test suites, for edge-coverage-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote percentage reductions in test-suite size.

To verify the correctness of this impression, we performed least-squares regression to fit the data depicted in these plots with a hyperbolic curve. Table 2 shows the best-fit curve for each of the subjects, along with its square of correlation, r^2.[7] The data indicates a strong hyperbolic correlation between percentage reduction in test-suite size (savings of test-suite reduction) and original test-suite size.

program	regression equation	r^2
totinfo	$y = 100 * (1 - (5.21/x))$	0.99
schedule1	$y = 100 * (1 - (5.46/x))$	0.96
schedule2	$y = 100 * (1 - (5.12/x))$	0.94
tcas	$y = 100 * (1 - (4.97/x))$	1.00
printtok1	$y = 100 * (1 - (7.49/x))$	0.90
printtok2	$y = 100 * (1 - (6.77/x))$	0.93
replace	$y = 100 * (1 - (12.10/x))$	0.99

Table 2: Correlation between test-suite size reduction and size of original test suite.

Our experiment's results indicate that test-suite reduction can produce savings in test-suite size on coverage-adequate, coverage-redundant test suites. The results also indicate that as test-suite size increases, the savings produced by test-suite reduction increase; a consequence of the relatively stable size of the reduced suites. Significantly, these results are relatively consistent across the seven subject programs, despite the differences in size, structure, and functionality among those programs.

Fault-detection effectiveness reduction

Figure 5 depicts the cost (reduction in fault-detection effectiveness) incurred by test-suite reduction for each of the seven subject programs. The data for each program P is represented by a scatterplot containing a point for each of the test suites utilized for P; each point shows the percentage reduction in fault-detection effectiveness observed for a test suite versus the size of that test suite prior to test-suite reduction.

Figure 6 illustrates the magnitude of the fault-detection effectiveness reduction observed for the seven subject programs. Again, this figure contains a scatterplot for each program; however, we find it most revealing to depict *faults detected* versus original test-suite size, simultaneously for both test suites reduced for edge-coverage (black) and for original test suites (grey). The solid lines in the plots denote average numbers of faults detected over the range of original test-suite sizes, the gap between these lines indicates the magnitude of the fault-detection effectiveness reduction for test suites reduced for edge coverage.

The plots show that the fault-detection effectiveness of test suites can be severely compromised by test-suite reduction. For example, on `replace`, the largest of the programs, test-suite reduction reduces fault-detection effectiveness by over 50%, with average fault loss ranging from four faults to twenty across the range of test-suite sizes, on more than half of the test suites. Also, although there are cases in which test-suite reduction does not reduce fault-detection effectiveness (e.g., on `printtok1`), there are also cases in which test-suite reduction reduces the fault-detection effectiveness of test suites by 100% (e.g., on `schedule2`).

[7] r^2 is a dimensionless index that ranges from zero to 1.0, inclusive, and is "the fraction of variation in the values of y that is explained by the least-squares regression of y on x" [11].

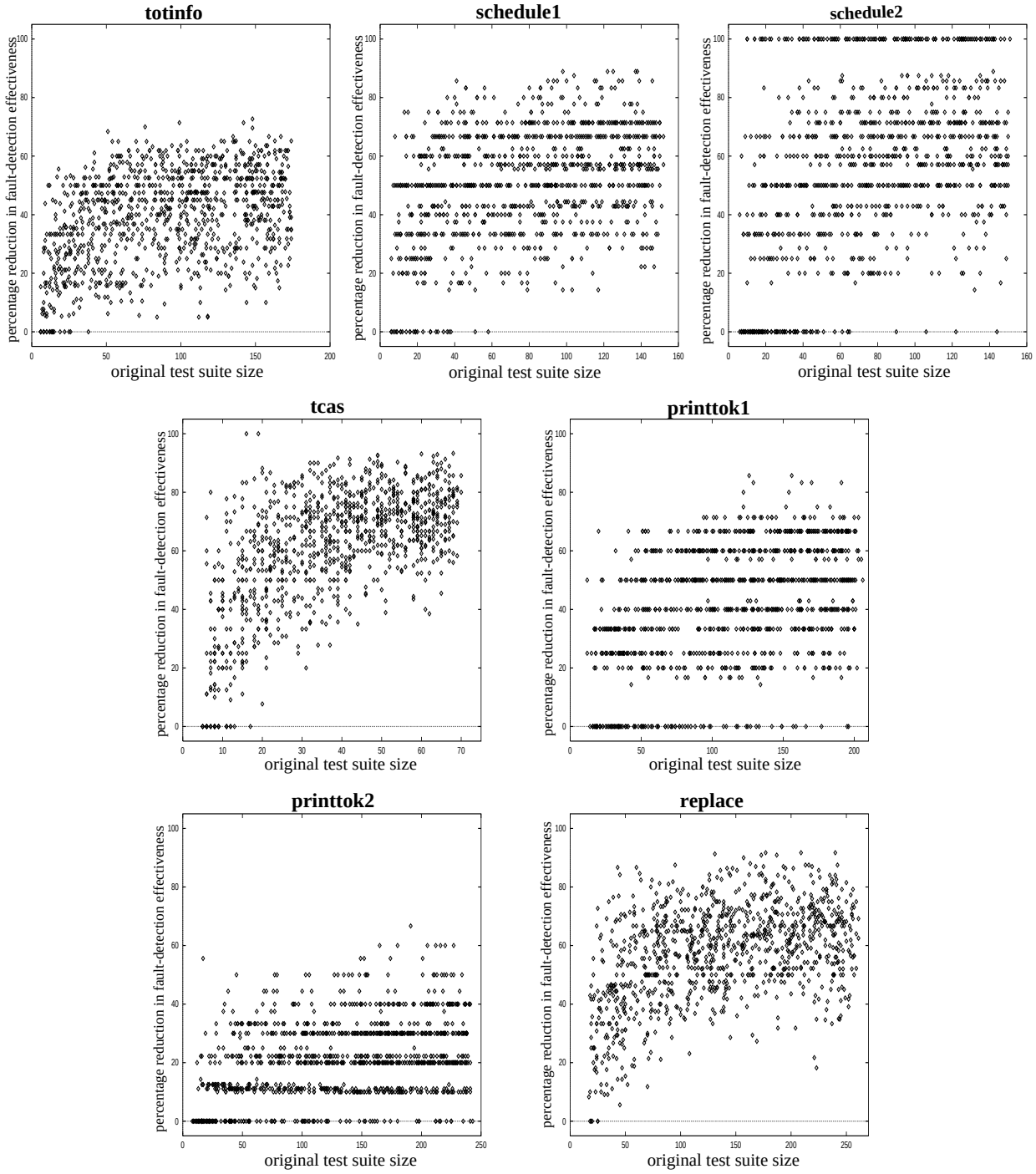

Figure 5: Percentage reduction in fault-detection effectiveness as a result of test-suite reduction versus sizes of original test suites, for edge-coverage-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote percentage reductions in fault-detection effectiveness.

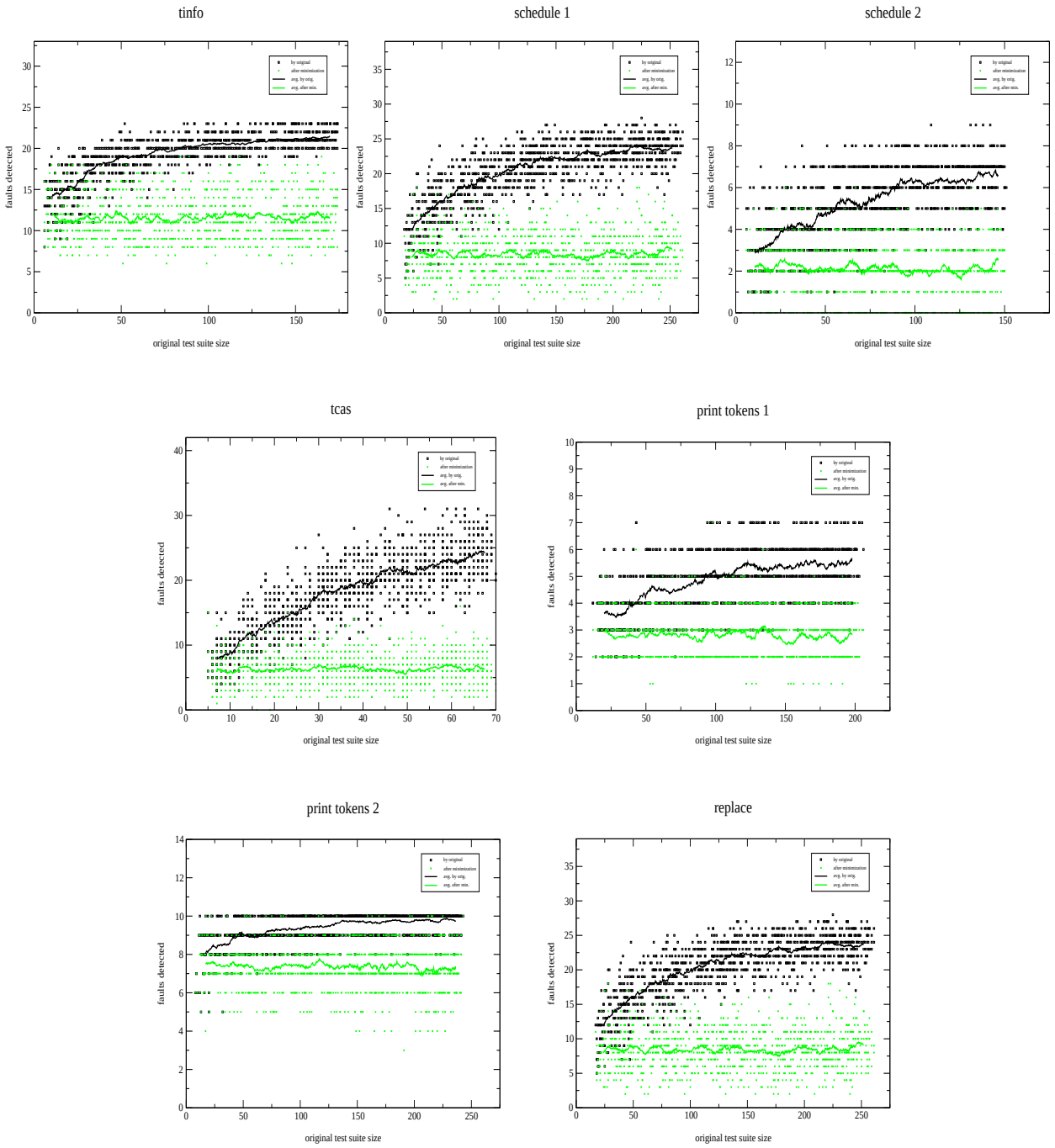

Figure 6: Fault-detection for edge-coverage-adequate and original test suites versus sizes of original test suites. Horizontal axes denote sizes of original test suites and vertical axes denote numbers of faults detected by test suites. Black dots and lines represent data for test suites reduced for edge-coverage, grey dots and lines represent data for original test suites. Averages are computed as running averages over each set of 50 consecutive points.

program	regression line 1	r^2	regression line 2	r^2	regression line 3	r^2
totinfo	$y = 0.13x + 27.79$	0.16	$y = 9.56Ln(x) - 1.71$	0.22	$y = -0.002x^2 + 0.44x + 17.74$	0.21
schedule1	$y = 0.15x + 38.92$	0.12	$y = 10.03Ln(x) + 9.25$	0.15	$y = -0.002x^2 + 0.47x + 29.80$	0.15
schedule2	$y = 0.28x + 34.86$	0.16	$y = 17.70Ln(x) - 17.12$	0.20	$y = -0.004x^2 + 0.89x + 17.07$	0.21
tcas	$y = 0.68x + 34.89$	0.38	$y = 22.18Ln(x) - 16.28$	0.47	$y = -0.020x^2 + 2.18x + 13.41$	0.46
printtok1	$y = 0.16x + 22.48$	0.18	$y = 14.68Ln(x) - 26.34$	0.20	$y = -0.001x^2 + 0.44x + 10.94$	0.20
printtok2	$y = 0.07x + 12.57$	0.11	$y = 6.82Ln(x) - 10.73$	0.13	$y = -0.001x^2 + 0.19x + 6.95$	0.13
replace	$y = 0.11x + 42.67$	0.20	$y = 13.07Ln(x) - 4.82$	0.27	$y = -0.001x^2 + 0.41x + 26.79$	0.28

Table 3: Correlation between reduction in fault-detection effectiveness and size of original test suite.

Visual inspection of the plots suggests that reduction in fault-detection effectiveness increases as test-suite size increases. Test suites in the smallest size ranges do produce effectiveness losses of less than 50% more frequently than they produce losses in excess of 50%, a situation not true of the larger test suites. Even the smallest test cases, however, exhibit effectiveness reductions in most cases: for example, on `replace`, test suites containing fewer than fifty test cases exhibit an average effectiveness reduction of nearly 40% (fault-detection reduction ranging from four to eight faults), and few such test suites do not lose effectiveness.

In contrast to the plots of size reduction effectiveness, the plots of fault-detection effectiveness reduction do not give a strong impression of closely fitting any curve or line: the data is much more scattered than the data for test-suite size reduction. Our attempts to fit linear, logarithmic, and quadratic regression curves to the data validate this impression: the data in Table 3 reveals little linear, logarithmic, or quadratic correlation between reduction in fault-detection effectiveness and original test-suite size.

These results indicate that test-suite reduction can compromise the fault-detection effectiveness of edge-coverage-adequate, coverage-redundant test suites. Moreover, the results also suggest that as test-suite size increases, the reduction in the fault-detection effectiveness of those test suites will increase.

One additional feature of the scatterplots of Figure 5 warrants discussion: on several of the graphs, there are markedly visible horizontal lines of points. In the graph for `printtok1`, for example, there are horizontal lines of points visible at 0%, 20%, 25%, 33%, 40%, 50%, 60%, and 67%. Such lines indicate a tendency for test-suite reduction to exclude particular percentages of faults for the programs on which they occur.

This tendency is partially explained by our use of a discrete number of faults in each subject program. Given a test suite that exposes k faults, test-suite reduction can exclude test cases that detect between 0 and k of these faults, yielding discrete percentages of reductions in fault-detection effectiveness. For `printtok1`, for example, there are seven faults, of which the unreduced test suites may reveal between zero and seven. When test-suite reduction is applied to the test suites for `printtok1`, only 19 distinct percentages of fault-detection effectiveness reduction can occur: 100%, 86%, 83%, 80%, 75%, 71%, 67%, 60%, 57%, 50%, 43%, 40%, 33%, 29%, 25%, 20%, 17%, 14%, and 0%. Each of these percentages except 29% and 100% is evident in the scatterplot for `printtok1`. With all points occurring on these 16 percentages, the appearance of lines in the graph is unsurprising.

It follows that as the number of faults utilized for a program increases, the presence of horizontal lines should decrease; this is easily verified by inspection, considering in turn `printtok1` with 7 faults, `schedule1` with 9, `schedule2` with 10, `printtok2` with 10, `totinfo` with 23, `replace` with 32, and `tcas` with 41.

This explanation, however, is only partial: if it were complete, we would expect points to lie more equally among the various reduction percentages (with allowances for the fact that there may be multiple ways to achieve particular reduction percentages). The fact that the occurrences of reduction percentages are not thus distributed reflects, we believe, variance in fault locations across the programs, coupled with variance in test-coverage patterns of faulty statements.

3.2.5 Experiment 2: Reduction of randomly generated test suites

Our second experiment addresses the question of how edge-coverage-based test-suite reduction compares to random selection as a test-suite reduction technique. To facilitate discussion, we refer to test suites whose size was reduced while keeping coverage constant as *edge-coverage-reduced test suites*, and we refer to test suites whose size was reduced by random selection as *randomly-reduced test suites*.

The experiment follows a paired-T test design [11]. A paired-T test is an experiment in which two large sets of data (populations) are compared by comparing the corresponding pairs in the two populations, in such a way that the pairs control extraneous variables. In this case, a one-to-one pairing of our edge-coverage-reduced test suites with randomly-reduced test suites let us control for differences among the unreduced test suites and differences in reduced test-suite sizes. As a result, we were able to compare the overall fault-detection effectiveness of edge-coverage-reduced test suites with the overall fault-detection effectiveness of the randomly-reduced test suites.

To randomly reduce test suites, we used Perl's built in pseudo-random number generator, seeded with system time, process-ID, and various other system variables.[8] From each original test suite T, we randomly selected test cases until we had selected a subset of T whose size corresponded to the size of the edge-coverage-reduced test suite previously obtained from T.

Test suite size reduction

By design, we produced randomly-reduced test suites of the same size as those produced by test-suite reduction in our first experiment. Thus, the test suites produce the same size reductions as those depicted in Figures 3 and 4.

Fault-detection effectiveness reduction

Figure 7 depicts the cost (reduction in fault detection) incurred by randomly selecting a subset of the original test suite. These scatterplots look similar to those of Figure 5, which depict the reduction in fault detection incurred by test-suite reduction. The only noticeable difference is that the scatterplot for the randomly selected test suites is somewhat denser for high failure rates.

Figure 8 illustrates the magnitude of the fault-detection effectiveness reduction observed for the seven subject programs for random test-suite reduction, compared with the reduction for edge-coverage-based test-suite reduction. Again, this figure contains a scatterplot for each program, and we depict *faults detected* versus original test-suite size, simultaneously for both test suites reduced for edge-coverage (black) and for

[8] This is dependent on the version of Perl and is described in the perlfunc man page for Perl 5.004.

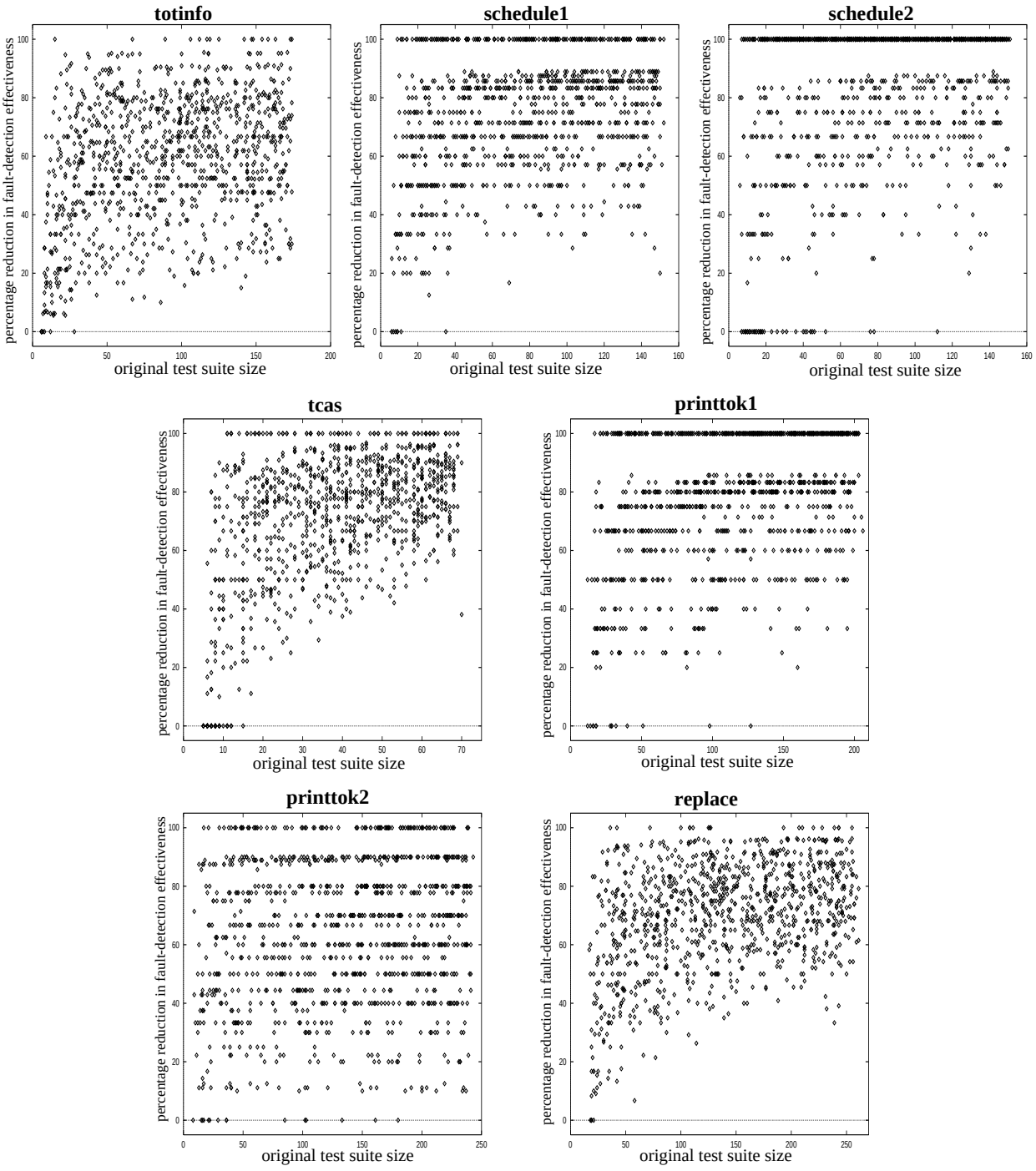

Figure 7: Percentage reduction in fault-detection effectiveness as a result of test-suite reduction versus sizes of original test suites, for randomly-reduced test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote percentage reductions in fault-detection effectiveness.

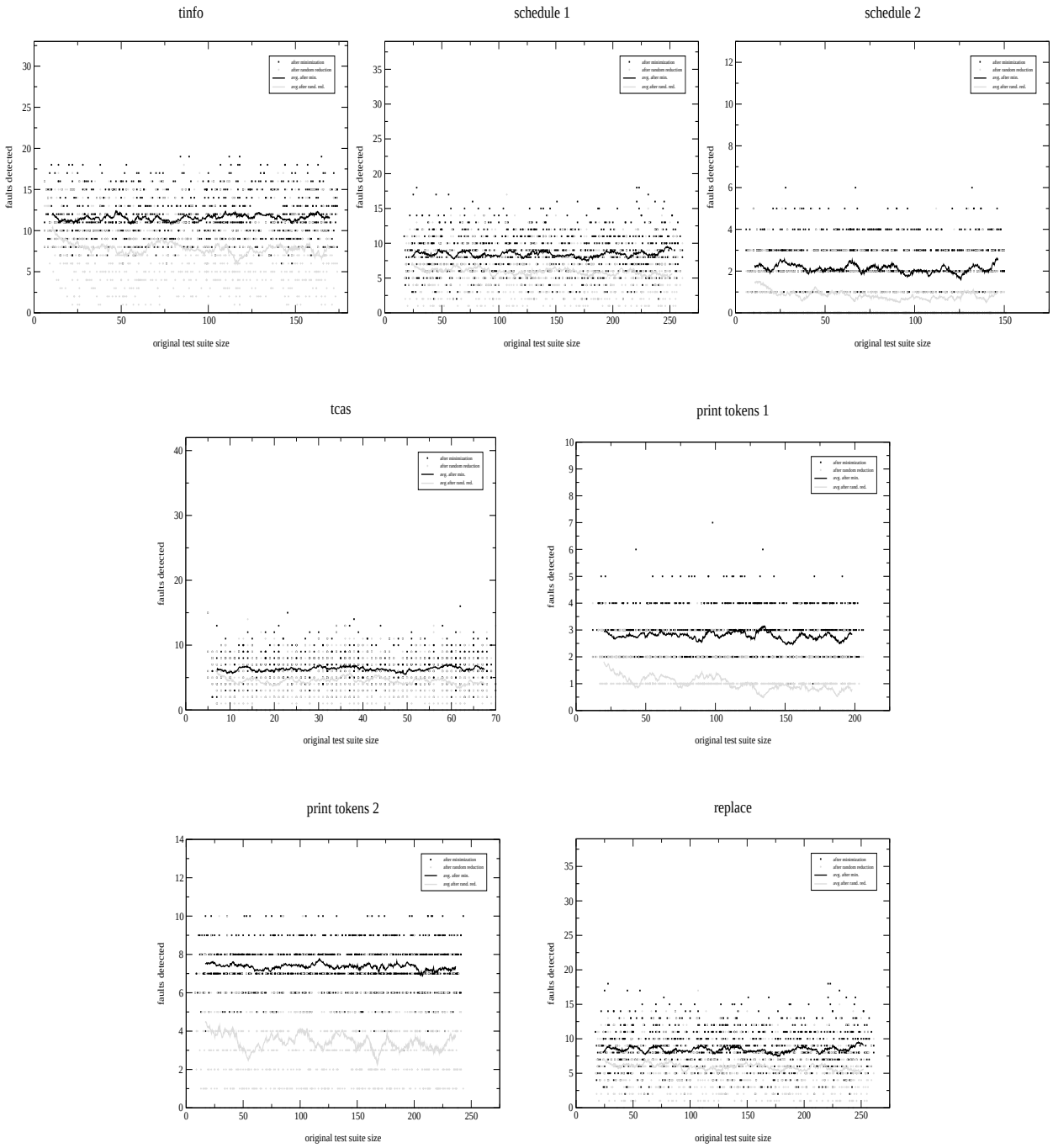

Figure 8: Fault-detection for randomly-reduced test suites and test suites reduced for edge coverage versus sizes of original test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote numbers of faults detected by test suites. Black dots and lines represent data for randomly-reduced test suites, grey dots and lines represent data for test suites reduced for edge coverage. Averages are computed as running averages over each set of fifty consecutive points.

program	regression line 1	r^2	regression line 2	r^2	reg ression line 3	r^2
totinfo	$y = 0.14x + 45.49$	0.10	$y = 11.35Ln(x) + 9.67$	0.16	$y = -0.002x^2 + 0.55x + 32.53$	0.15
schedule1	$y = 0.15x + 62.37$	0.09	$y = 10.63Ln(x) + 29.92$	0.14	$y = -0.003x^2 + 0.58x + 50.16$	0.13
schedule2	$y = 0.19x + 66.50$	0.09	$y = 13.49Ln(x) + 25.43$	0.14	$y = -0.004x^2 + 0.78x + 49.73$	0.14
tcas	$y = 0.61x + 48.30$	0.24	$y = 20.61Ln(x) - 0.24$	0.32	$y = -0.020x^2 + 2.10x + 27.01$	0.30
printtok1	$y = 0.15x + 60.11$	0.13	$y = 14.62Ln(x) + 10.81$	0.16	$y = -0.001x^2 + 0.44x + 48.32$	0.15
printtok2	$y = 0.06x + 55.26$	0.02	$y = 6.52Ln(x) + 32.39$	0.03	$y = -0.001x^2 + 0.20x + 48.81$	0.03
replace	$y = 0.10x + 55.44$	0.15	$y = 12.53Ln(x) + 9.96$	0.20	$y = -0.001x^2 + 0.36x + 42.27$	0.19

Table 4: Correlation between reduction in fault-detection effectiveness due to random reduction and size of original test suite.

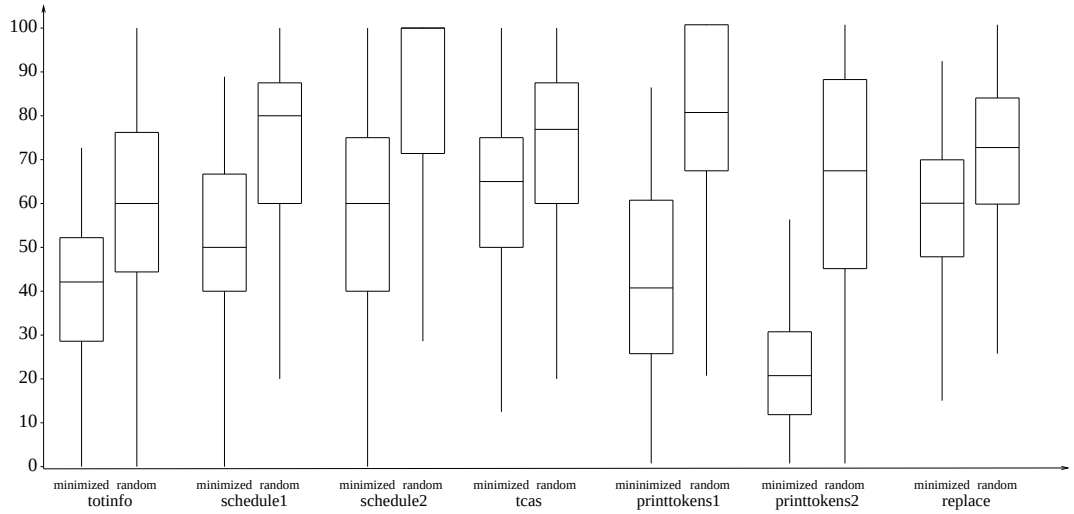

Figure 9: Percentage reduction in fault-detection effectiveness as a result of edge-coverage reduction and random reduction for the subject programs.

randomly-reduced test suites (grey). The solid lines in the plots denote average numbers of faults detected over the range of original test-suite sizes, the gap between these lines indicates the difference between the two test-suite reduction techniques. The plots indicate a noticeable difference between the two techniques.

As with the test suites reduced by edge-coverage-based test-suite reduction, we attempted to fit the data points for fault-detection effectiveness reduction of randomly reduced test suites (Figure 7) to some simple functions. The results of this attempt (shown in Table 4) were similar to those for edge-coverage-based test-suite reduction (Table 3) as both show little linear, logarithmic, and quadratic correlation between reduction in fault-detection effectiveness and the size of the original test suite. The randomly-reduced test suites, however, have even lower correlation coefficients, reflecting the more variable nature of random reduction.

Figure 9 shows boxplots representing (comparatively) the span of the fault detection reductions for the various Siemens subjects. Boxplots for the edge-coverage-reduced and randomly-reduced test suites are shown side by side. The boxplots show a consistent pattern of greater loss in fault detection in the randomly-reduced test suites than in their associated reduced test suites.

program	Average Reduction in Fault Detection for EC-Reduced Test Suites	Average Reduction in Fault Detection for Randomly Reduced Suites	Average Difference in Fault Detection Reduction
totinfo	39.2	58.2	19.0 ± 2.5
schedule1	51.1	74.2	23.2 ± 2.7
schedule2	56.7	81.7	25.0 ± 3.0
tcas	60.9	71.4	10.6 ± 2.4
printtok1	40.8	77.7	36.9 ± 2.7
printtok2	21.3	63.1	41.7 ± 2.8
replace	57.2	69.4	12.2 ± 2.3

Table 5: A comparison between the effectiveness of edge-coverage-reduced and randomly-reduced versions of the edge-coverage-based test suites.

program	Sample Standard Deviation in Reduction in Fault Detection for EC-Reduced Test Suites	Sample Standard Deviation in Reduction in Fault Detection for Randomly Reduced Suites
totinfo	15.7	22.1
schedule1	18.5	20.7
schedule2	23.2	24.0
tcas	20.3	22.9
printtok1	20.5	22.7
printtok2	13.2	25.3
replace	16.7	18.6

Table 6: A comparison between the variance in effectiveness of edge-coverage-reduced and randomly-reduced versions of the unreduced test suites.

Table 5 shows statistical data for randomly-reduced and edge-coverage-reduced test suites. The data confirms conclusions drawn from Figure 9: the edge-coverage-reduced test suites tended to find more faults than their randomly reduced counterparts. The fourth column shows the difference in lost fault detection between edge-coverage-based and random reduction; the margins of error are shown for the 99.9% confidence level. The average advantage ranged from 10.6% for tcas to 41.7% for printtok2. The differences are significant as the entire confidence intervals are entirely positive.

Table 6 further quantifies results not directly apparent in Figure 9: for all programs, the edge-coverage-reduced test suites detected faults more consistently than their randomly-reduced counterparts. The smallest difference was found for `schedule2`, where the reduction in fault detection for the edge-coverage-reduced test suites had a standard deviation of 23.2, and the reduction in fault detection for the randomly-reduced test suites was only slightly less consistent with a standard deviation of 24. The largest difference was for `printtok2`, where the reduction in fault detection for edge-coverage-reduced test suites had a standard deviation of only 13.2, while the reduction in fault detection for the randomly-reduced test suites had a standard deviation of 25.3.

3.2.6 Experiment 3: Reduction of all-uses-coverage-adequate test suites

Our third experiment addresses our research questions by applying test-suite reduction to the Siemens programs and their all-uses-coverage-adequate test suites.

Test suite size reduction

Figure 10 depicts the sizes of the reduced all-uses-coverage-adequate test suites for the seven subject programs, plotted against original test-suite size. The results reveal that the reduced test suites have average (median) sizes ranging from 5 (for `tcas`) to 34 (for `replace`). Similar to Experiment 1, the reduced test suites for each program demonstrate little variance in size: `tcas` showing the least variance (between four and five test cases), and `replace` showing the greatest (between 24 and 42 test cases). Again, considered across the range of original test-suite sizes, reduced test-suite size for each program is relatively stable.

Figure 11 depicts the percentage reduction in test-suite size produced by test-suite reduction in terms of the formula given in Section 3.1.1, for each of the subject programs; each point in each plot shows the percentage size reduction achieved for a test suite versus the size of that test suite prior to test-suite reduction. Visual inspection of the plots indicates a strong similarity to the corresponding plots in the first experiment. As in Experiment 1, the data in this experiment gives the impression of fitting a hyperbolic curve.

To verify the correctness of this impression we performed the same data analysis as in Experiment 1. Table 7 shows the regression results: the relatively large values of r^2 indicate a relatively strong hyperbolic correlation between reduction in test-suite size (savings of test-suite reduction) and original test-suite size.

program	regression equation	r^2
totinfo	$y = 100 * (1 - \frac{11.03}{x})$	0.97
schedule1	$y = 100 * (1 - \frac{13.99}{x})$	0.98
schedule2	$y = 100 * (1 - \frac{16.68}{x})$	0.99
tcas	$y = 100 * (1 - \frac{4.97}{x})$	1.00
printtok1	$y = 100 * (1 - \frac{14.59}{x})$	0.67
printtok2	$y = 100 * (1 - \frac{14.44}{x})$	0.88
replace	$y = 100 * (1 - \frac{33.38}{x})$	0.90

Table 7: Correlation between reduction in test-suite size and size of original test suite, for all-uses-coverage-adequate test suites

These results indicate that when reducing for all-uses adequacy, as when reducing for edge-coverage adequacy, test-suite reduction can produce savings in test-suite size on coverage-adequate, coverage-redundant test suites, and as test-suite size increases, the savings produced by test-suite reduction increase. The resulting reduced test suites are (with exception of the suites for `tcas`), larger than those produced by reducing for edge-coverage, arguably due to the greater number of test cases required by the all-uses-coverage criterion.

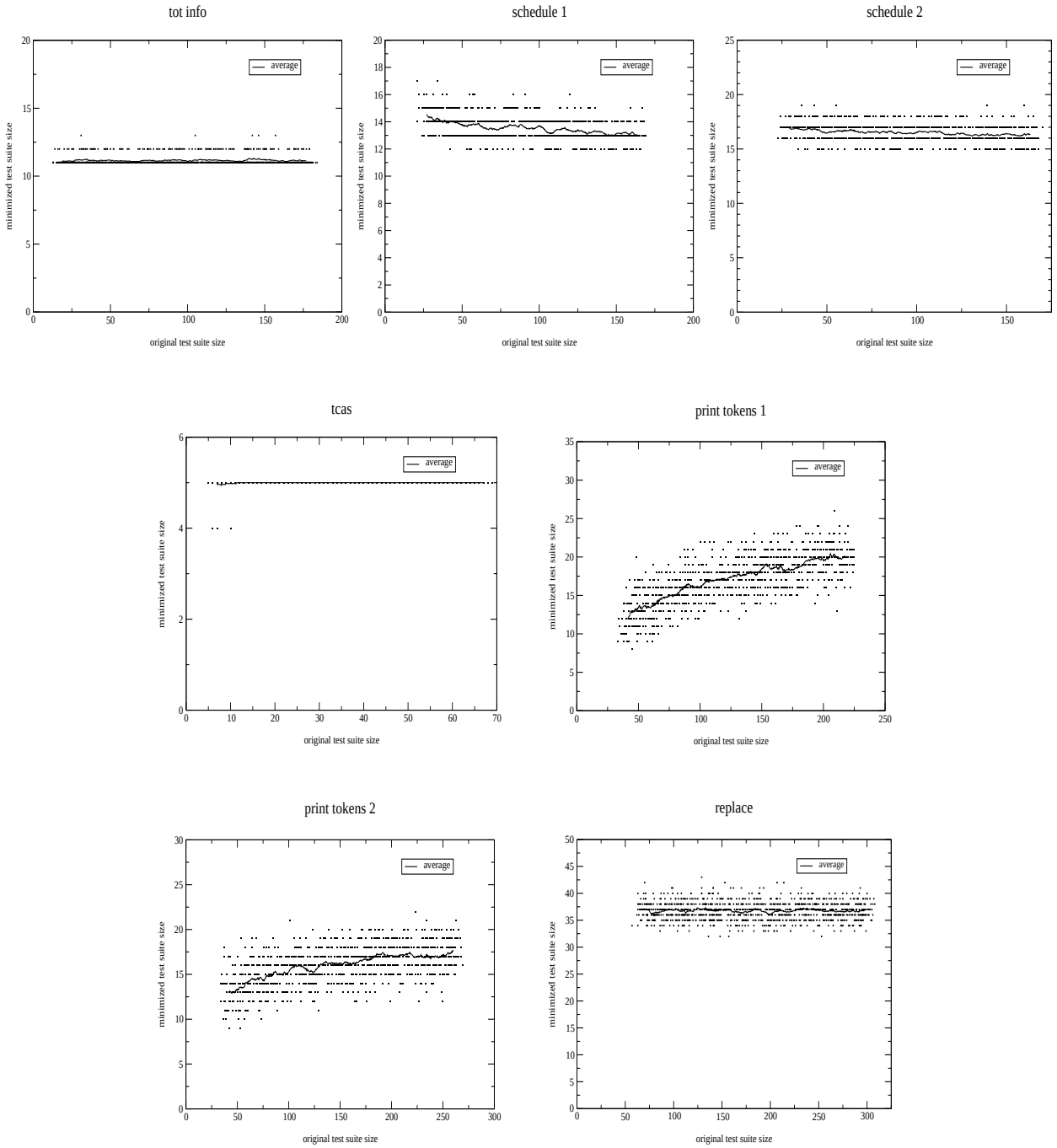

Figure 10: Sizes of test suites reduced for all-uses coverage versus sizes of original test suites, for all-uses-coverage-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote sizes of reduced test suites. Average reduced test-suite size across the range of original test-suite sizes (computed as running averages over each set of fifty consecutive points) is denoted by solid lines.

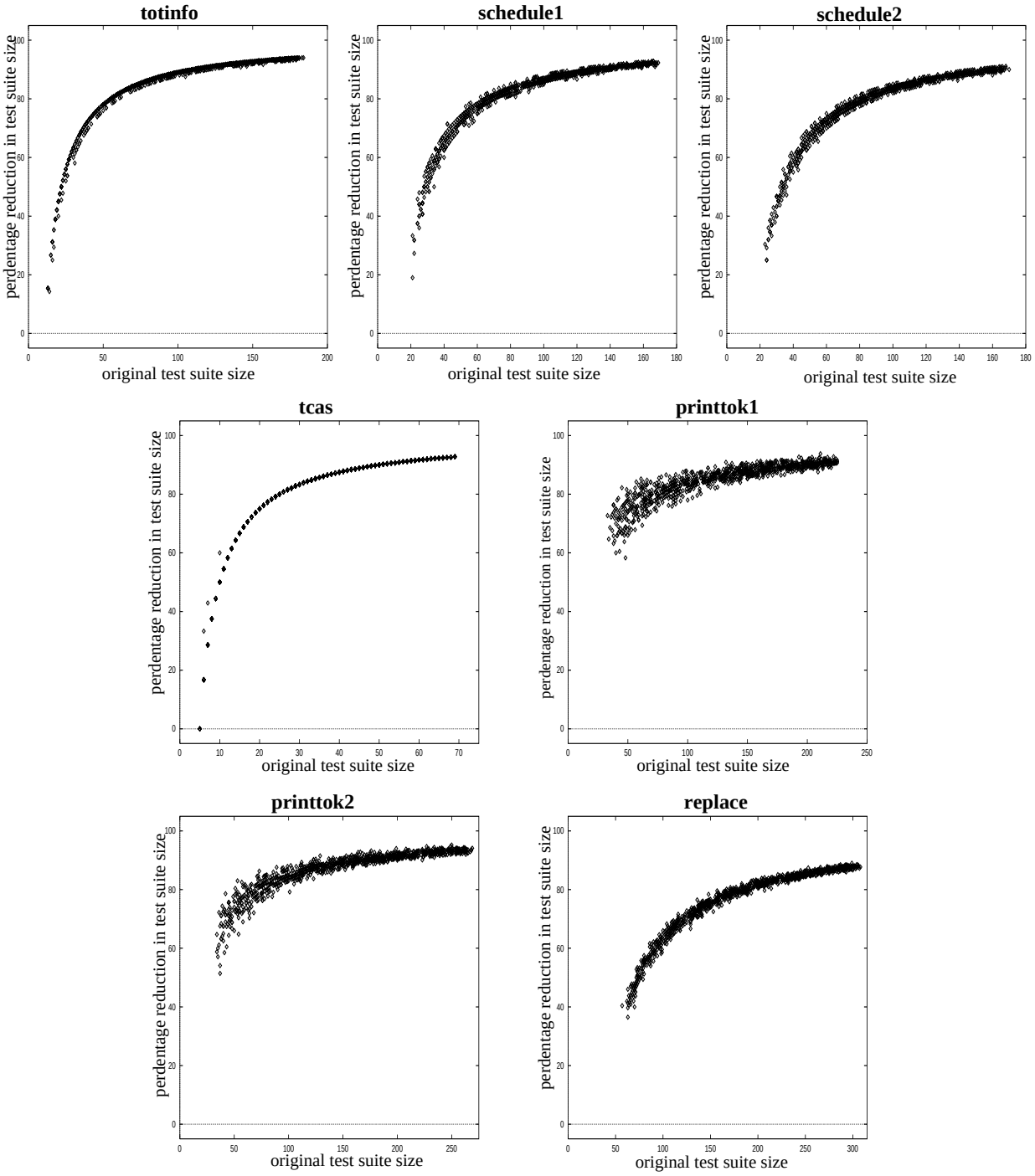

Figure 11: Percentage reduction in test-suite size as a result of test-suite reduction versus sizes of original test suites, for all-uses-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote percentage reductions in test-suite size.

Fault-detection effectiveness reduction

Figure 12 depicts the cost (reduction in fault-detection effectiveness) incurred by test-suite reduction, in terms of the formula discussed in Section 3.1.1, for each of the seven subject programs. As in Experiment 1, the data for each program P is represented by a scatterplot containing a point for each of the test suites utilized for P; each point shows the percentage reduction in fault-detection effectiveness observed for a test suite versus the size of that test suite prior to test-suite reduction.

Figure 13 illustrates the magnitude of the fault-detection effectiveness reduction observed for the seven subject programs. Again, this figure contains a scatterplot for each program and depicts *faults detected* versus original test-suite size, simultaneously for both test suites reduced for all-uses-coverage (black) and for the original test suites (grey). The solid lines in the plots denote average numbers of faults detected over the range of original test-suite sizes, the gap between these lines indicates the magnitude of the fault-detection effectiveness reduction for test suites reduced for edge coverage.

Similar to the results in Experiment 1, the scatterplots show that the fault-detection effectiveness of test suites can be severely compromised by test-suite reduction. There are cases in which test-suite reduction does, and does not, reduce fault-detection effectiveness. Again, effectiveness reduction appears to increase as test-suite size increases. Again, these effects occur even among the smaller test suites.

As was the case for Experiment 1, the scatterplots do not give a strong impression of closely fitting a line or curve. Table 8 reveals that the correlation between fault-detection effectiveness reduction and original test-suite size does not closely fit a linear, logarithmic, or quadratic curve using regression analysis.

program	regression line 1	r^2	regression line 2	r^2	regression line 3	r^2
totinfo	$y = 0.09x + 19.60$	0.09	$y = 07.88Ln(x) - 06.36$	0.12	$y = -0.001x^2 + 0.34x + 10.32$	0.13
schedule1	$y = 0.11x + 14.10$	0.11	$y = 09.12Ln(x) - 15.80$	0.11	$y = -0.001x^2 + 0.23x + 09.54$	0.11
schedule2	$y = 0.13x + 38.50$	0.08	$y = 11.80Ln(x) - 01.46$	0.09	$y = -0.002x^2 + 0.44x + 26.17$	0.09
tcas	$y = 0.72x + 31.50$	0.39	$y = 23.60Ln(x) - 23.00$	0.50	$y = -0.023x^2 + 2.42x + 07.52$	0.50
printtok1	$y = 0.03x + 37.80$	0.01	$y = 02.87Ln(x) + 28.20$	0.01	$y = 0.001x^2 - 0.10x + 44.86$	0.02
printtok2	$y = 0.03x + 11.30$	0.01	$y = 03.35Ln(x) - 01.35$	0.01	$y = -0.001x^2 + 0.06x + 09.44$	0.01
replace	$y = 0.03x + 14.00$	0.02	$y = 04.42Ln(x) - 03.28$	0.02	$y = 0.001x^2 - 0.02x + 18.21$	0.03

Table 8: Correlation between reduction in fault-detection effectiveness and size of original test suite for all-uses-adequate test suites.

The scatterplots in Figure 12 also contain horizontal lines similar to those seen in Experiment 1, and explained in the discussion of that experiment.

This fault-detection effectiveness reduction data indicates that when reducing for all-uses adequacy, as when reducing for edge-coverage adequacy, test-suite reduction can significantly compromise the fault-detection effectiveness of coverage-adequate, coverage-redundant test suites. The results also suggest that as test-suite size increases, the reduction in the fault-detection effectiveness of those test suites will increase.

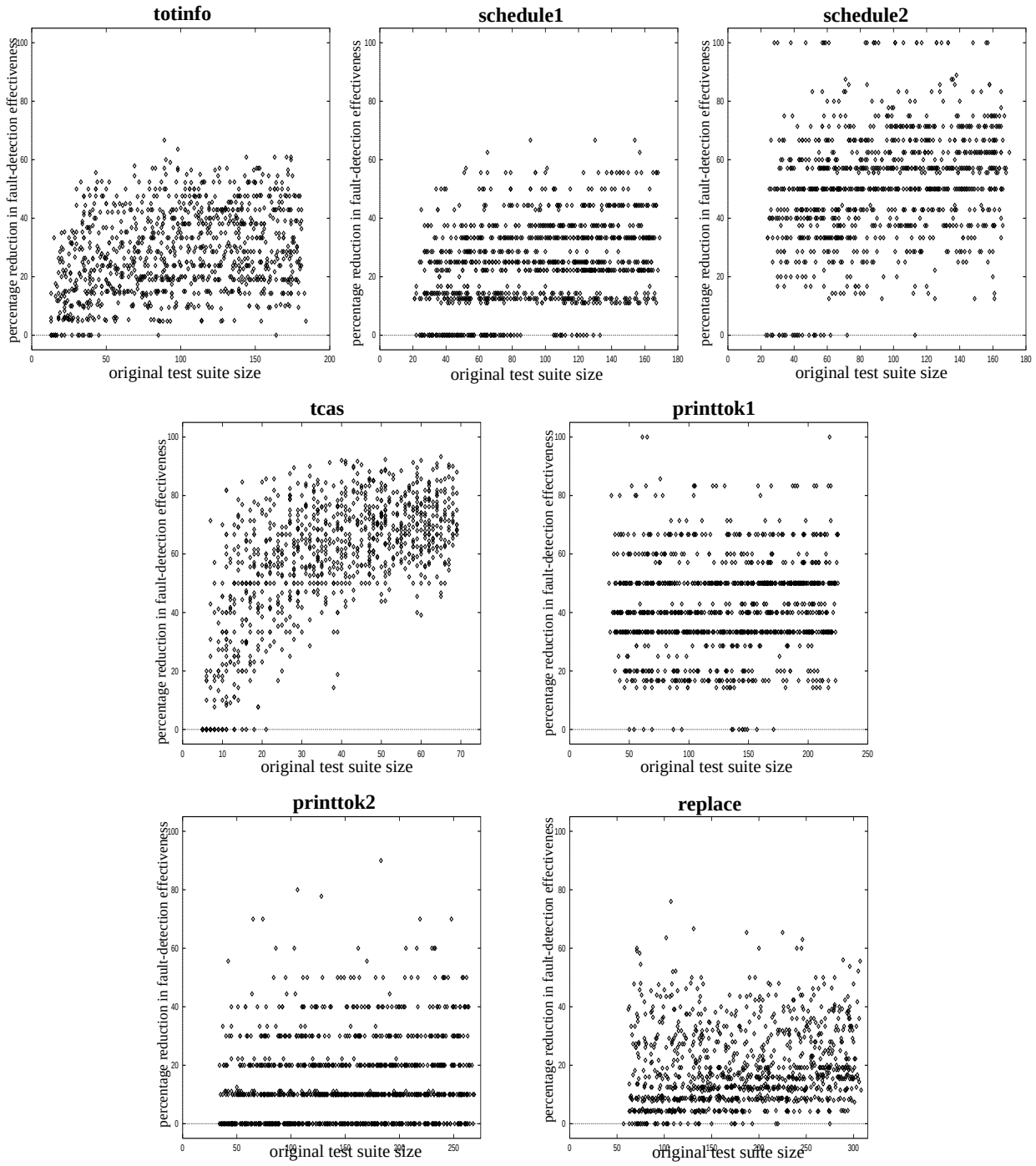

Figure 12: Percentage reduction in fault-detection effectiveness as a result of test-suite reduction versus sizes of original test suites, for all-uses-coverage-adequate test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote percentage reductions in fault-detection effectiveness.

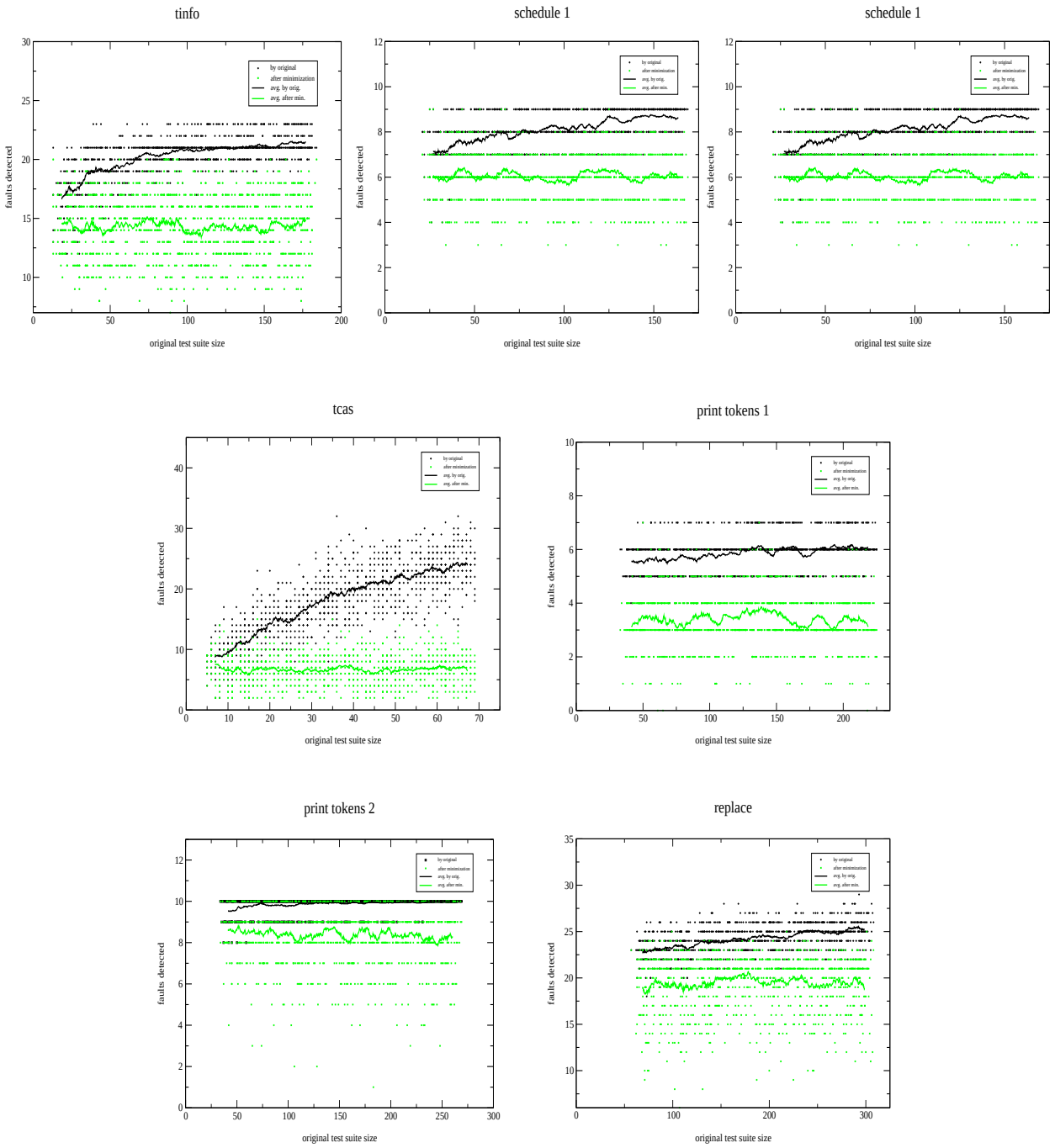

Figure 13: Fault-detection for all-uses-coverage-adequate and original test suites versus sizes of original test suites. Horizontal axes denote sizes of original test suites, and vertical axes denote numbers of faults detected by test suites. Black dots and lines represent data for test suites reduced for all-uses-coverage, grey dots and lines represent data for original test suites. Averages are computed as running averages over each set of fifty consecutive points.

3.3 Experiment 4: the `Space` program

Our next experiment addresses our research questions by applying test-suite reduction to the `Space` program utilized in the WHMP study.

3.3.1 Subject program, faulty versions, test cases, and test suites.

`Space` (see Table 9), consisting of 9564 lines of C code (6218 executable), serves as an interpreter for an array definition language (ADL). The program reads a file that contains several ADL statements, and checks the contents of the file for adherence to the ADL grammar, and to specific consistency rules. If the ADL file is correct, `Space` outputs an array data file containing a list of array elements, positions, and excitations; otherwise the program outputs error messages.

Program	Lines of Code	No. of Versions	Test Pool Size	Description
Space	6218	35	13585	language interpreter

Table 9: Subject program.

`Space` has 33 associated versions, each containing a single fault that had been discovered during the program's development. (The WHMP study utilized only eighteen of these faulty versions.) Through working with this program, we discovered five additional faults, and created versions containing just those faults. We also discovered that three of the "faulty versions" were actually semantically equivalent to the base version. We excluded these from our study; therefore, we ultimately utilized 35 faulty versions.

The test pool for `Space` was constructed in two stages. An initial pool of 10,000 tests was obtained from Frankl and Vokolos, who had constructed the pool for another study by randomly generating test cases [17]. Beginning with this initial pool, we instrumented the program for edge coverage, measured coverage, and then added additional test cases to the pool until it contained, for each dynamically executable edge in the control flow graph for the program, at least 30 test cases that exercised that edge. This process yielded a test pool of 13,585 test cases.

Figure 14 shows the sensitivity to detection of the faults in `Space` relative to the test pools; the bar graph illustrates that the sensitivities of the faults varies, but overall falls between 0.13% and 99.77%. In all, 74% (26/35) of these faults were, in the terminology of the WHLM studies, Quadrant I faults, detectable by fewer than 25% of the test-pool inputs.

As with the Siemens programs, we used the `Space` program's test pools to obtain coverage-adequate test suites for the program; however, due to limitations in our data-flow analyzer, we were able to create only edge-coverage-adequate suites. As with the Siemens programs we utilized test suites consisting of a random number of randomly-selected test cases together with additional test cases necessary to achieve coverage. The test suites ranged in size from 159 to 4712 test cases.

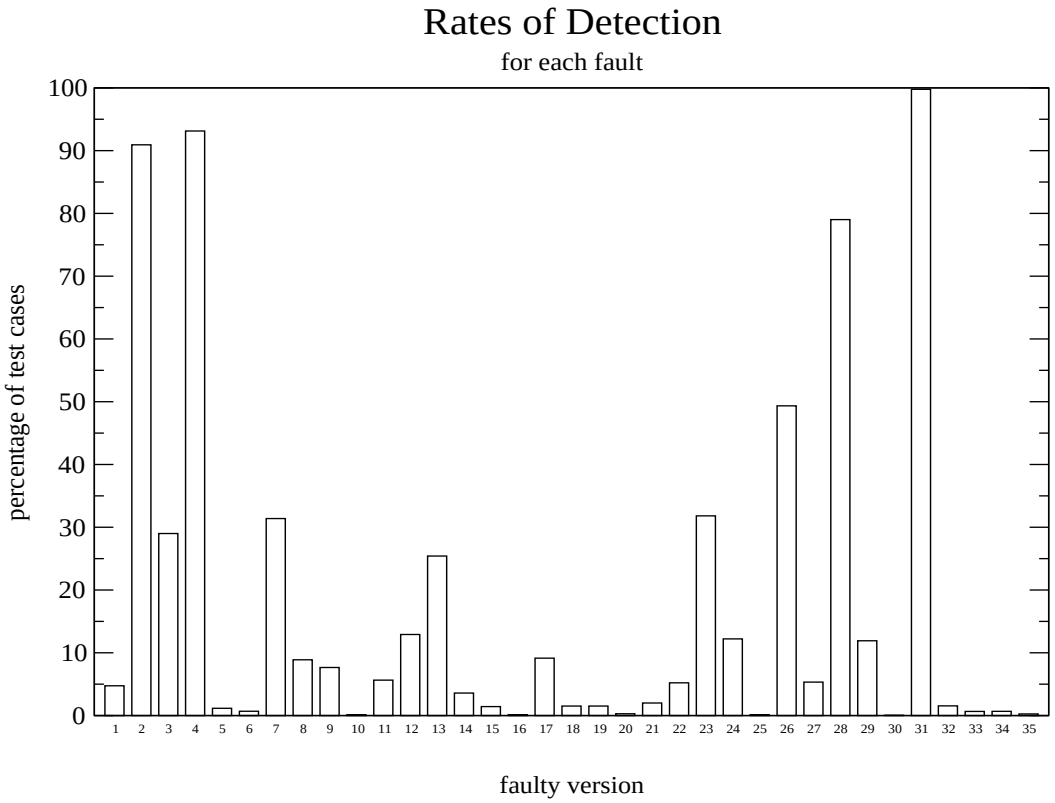

Figure 14: Bar chart that shows the percentages of inputs in the test pool for `Space` that expose faults in the versions of the program.

3.3.2 Experiment design.

We applied test-suite reduction techniques to each of the 1000 sample test suites. We then computed the size and effectiveness reductions for these test suites. Also, as with the Siemens programs, we conducted an additional experimental run utilizing randomly selected test cases, using a paired-T test design. We report the results of both experimental runs together.

3.3.3 Threats to validity.

This experiment shares, with our experiments on the Siemens programs, the threats to validity described in Section 3.2.3. In addition, the program's naturally occurring faults had been separated into single faulty versions, abstracting out effects that could occur from interacting faults; however, this abstraction was necessary in order to be able to attribute failures properly to faults. On the other hand, `Space` is a real program, with real faults uncovered in practice, and its size is an order of magnitude greater than that of the Siemens programs; these factors augment the external validity of the study.

3.3.4 Data and Analysis

We again report results on size reduction first, followed by results on fault-detection effectiveness reduction.

Test-suite size reduction

Figure 15 depicts the sizes of the reduced edge-coverage-adequate test suites for `Space`. These show that the size of the reduced test suites is relatively stable and does not show a large variance.

Figure 16 shows the percentage reduction in reduced test-suite size versus the size of the original test suites. The scatterplot is hyperbolic – and nearly identical to those seen in the experiments on the Siemens programs – reflecting reductions to a nearly constant size across a wide range of original test-suite sizes. Table 10 shows attempts to fit curves to the points in the plot; again, the hyperbolic curve is a good fit.

regression equation	r^2
$y = -13.92Ln(x) - 14.83$	0.79
$y = 0.01x + 74.436$	0.47
$y = -4.03e-06x^2 + 0.025x + 58.45$	0.73
$y = 100 - 100 * 121.01/x$	0.99

Table 10: Correlation between reduction in test-suite size and size of original test suites.

Fault-detection effectiveness reduction

Figures 17 and 18 show scatterplots for the reduction in fault detection for the edge-coverage-based and random reduction of `Space`'s test suites, respectively. For each of the treatments, the highest loss in fault detection effectiveness is less than 40%. Table 11 shows attempts to fit a curve to the effectiveness reductions for the edge-coverage-based case. We found no curves that fit the data well.

Figure 19 illustrates the magnitude of the fault-detection effectiveness reduction observed for `Space`. Similar to the figures presented for the Siemens programs, the figure contain a scatterplot for the test suites. The scatterplot depicts *faults detected* versus original test-suite size, simultaneously for original test suites (black), test suites reduced for all-edges-coverage (dark grey) and randomly reduced test suites (light grey). The solid lines in the plot denote average numbers of faults detected over the range of original test-suite sizes, the gaps between these lines indicate differences between the test suites. The plots reveal noticeable differences between the techniques as test-suite size grows.

Table 12 shows average reductions in fault detection resulting from applying different treatments to `Space`. The test suites exhibited average fault reductions of 18.1% on the randomly generated suites and 8.9% on the edge-coverage-based suites. These average losses in fault-detection effectiveness are here smaller than those observed with the Siemens programs. As in the experiments with the Siemens programs, fault-detection reduction due to random reduction was greater than that due to edge-coverage-based reduction (9.2 ± 0.7), and the margin of errors show that to be significant at $\alpha = 0.001$ (99.9% confidence).

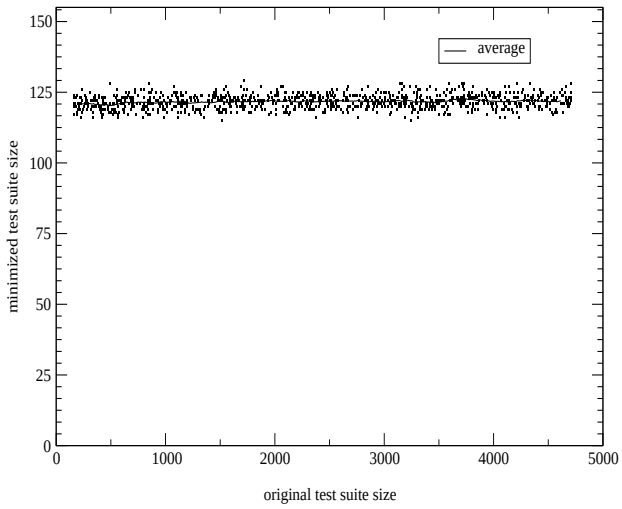

Figure 15: Sizes of test suites reduced for coverage versus sizes of initial test suites. Horizontal axes denote the sizes of initial test suites, and vertical axes denote sizes of reduced test suites. Average reduced test-suite size across the range of original test-suite sizes (computed as running averages over each set of 50 consecutive points) is denoted by the solid line.

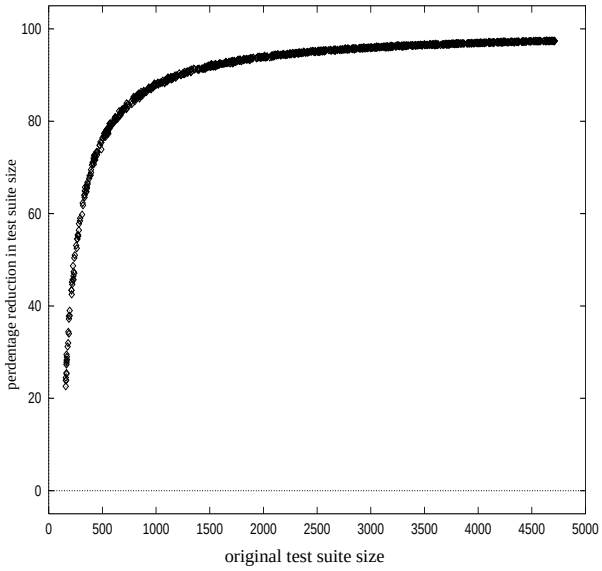

Figure 16: Percentage reduction in test-suite size as a result of test-suite reduction, versus sizes of initial test suites. Horizontal axes denote the sizes of initial test suites, and vertical axes denote percentage reductions in test-suite size.

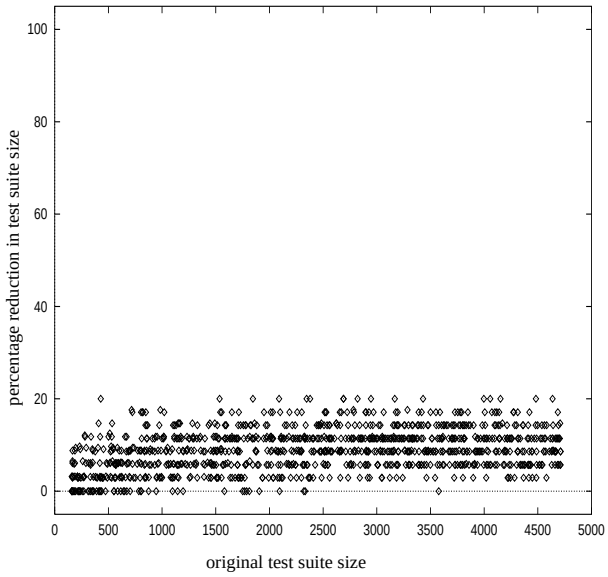

Figure 17: Percentage reduction in fault-detection effectiveness as a result of reduction, versus sizes of original test suites, for edge-coverage-based reduction. Horizontal axes denote sizes of initial test suites, and vertical axes denote percentage reductions in fault-detection effectiveness.

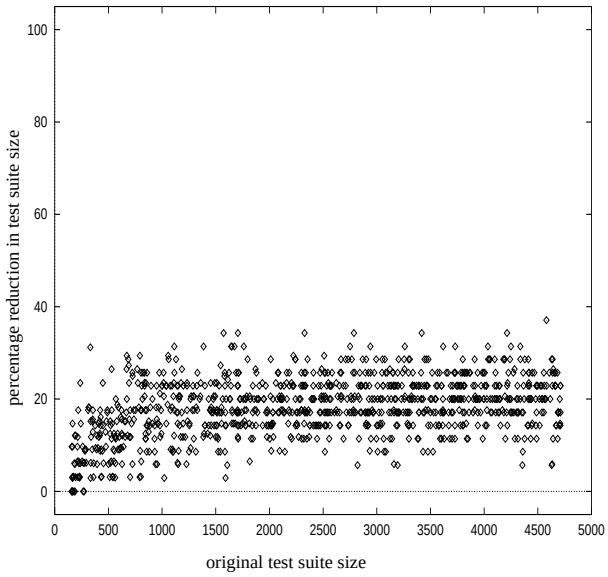

Figure 18: Percentage reduction in fault-detection effectiveness as a result of reduction, versus sizes of original test suites, for random reduction. Horizontal axes denote sizes of initial test suites, and vertical axes denote percentage reductions in fault-detection effectiveness.

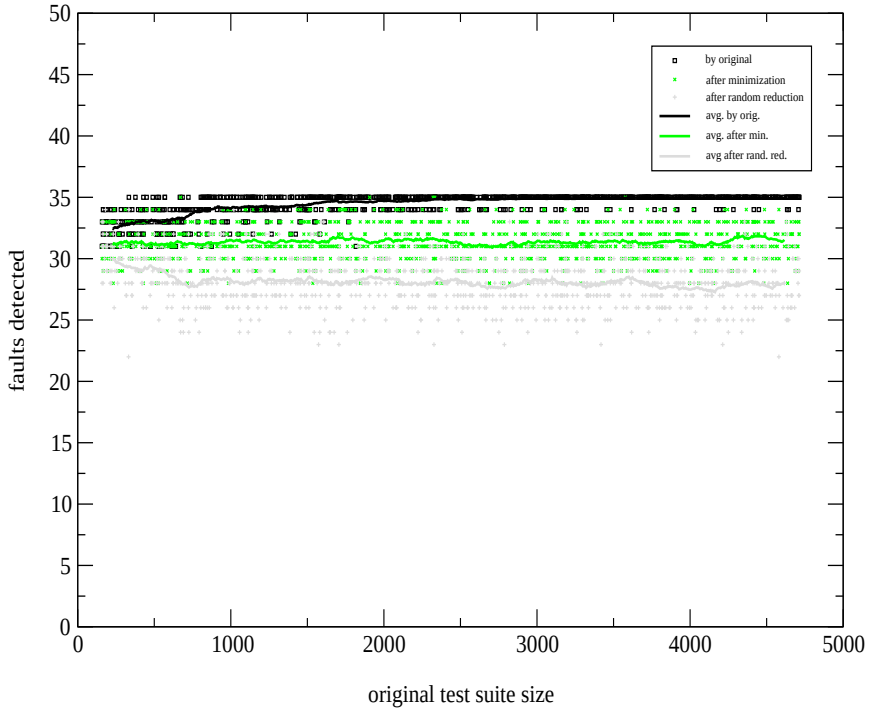

Figure 19: Fault-detection statistics for reduced test suites, versus sizes of original test suites. Horizontal axes denote the size of the original test suites, and vertical axes denote the number of faults detected by suites. Black dots and lines represent data for original test suites, dark gray dots and lines represent data for test suites reduced for edge coverage, and light gray dots and lines represent data for randomly-reduced test suites. Averages are computed as running averages over each set of fifty consecutive points.

regression equation	r^2
$y = 0.001x + 6.23$	0.10
$y = 2.18Ln(x) - 7.49$	0.15
$y = -6.36e-07x^2 + 0.004x + 3.71$	0.15

Table 11: Correlation between reduction in fault-detection effectiveness and size of original test suite.

Average Reduction in Fault Detection for EC-Reduced Test Suites	Average Reduction in Fault Detection for Randomly Reduced Suites	Average Difference in Fault Detection Reduction
8.9	18.1	9.2 ± 0.7

Table 12: Comparison of average reduction in fault detection among edge-coverage-reduced and randomly-reduced test suites.

4 Discussion

As we discussed earlier, these experiments, like any other, have several limits to their validity. In particular, these limits include several threats to external validity that limit our ability to generalize our results; these threats can be addressed only by additional experimentation with a wider variety of programs, faults, and test suites. Keeping this in mind, we now discuss some of the implications of our results, both singly and in relation to previous results.

The WHLMP studies and our studies indicate that test-suite reduction can produce savings by reducing test-suite size, and that these savings increase with test-suite size. The studies also indicate that reduction in fault-detection effectiveness increases as test-suite size increases.

The studies differ substantially, however, in their results pertaining to fault-detection effectiveness reduction. The authors of the WHLMP studies conclude that: (1) test suites that do not add coverage are not likely to detect additional faults, and (2) fault-detection effectiveness reduction is insignificant even for test suites that have high block coverage. Our results on `Space` are similar in this respect to those of the WHMP study. That study shows an average fault-detection effectiveness reduction less than 10% for all of test-suite sizes. While Figure 17 shows somewhat larger losses in fault reduction in some cases, the average reduction in fault-detection capability of 8.9% is not far from that discovered in WHMP.

This conclusion contrasts markedly, however, with our results on the Siemens programs, where fault-detection effectiveness was severely compromised by test-suite reduction. As a practical consideration, if we are considering employing test suite reduction, we would like to know which sort of results would apply in our particular case, for our programs and test cases, and for the types of faults they may contain. Ultimately, this issue must be addressed by further controlled studies. We here suggest and discuss several potential factors that might be responsible for the differences between the results of these studies, and we discuss the implications of these factors for test suite reduction.

First, the WHLM study used ATAC for test-suite reduction, whereas our study employed the algorithm of Reference [6]. Reference [19] reports that ATAC achieved minimal test selection on the cases studied; we have not yet determined whether our algorithm was equally successful. However, if our algorithm is less successful than the algorithm used in the WHLM study, we would expect this to cause us to underestimate possible reductions in fault detection effectiveness. A better algorithm, if possible, could only exacerbate the already large difference in our fault-detection effectiveness reduction results.

Second, the Siemens programs differ from the programs utilized in the WHLM study, and all but one of the Siemens programs is larger than all but one of the programs used in the WHLM study. Differences in program size and structure could certainly contribute to differences in fault-detection effectiveness reduction.

Third, the studies utilized different types of test suites. The WHLM study used test suites that were not coverage-adequate, and used coverage-based suites that were relatively small compared to our test suites. Overall, 928 of the 1198 test suites utilized in the WHLM study belonged to groups of test suites whose average sizes did not exceed 4.5 test cases. Small test-suite size reduces opportunities both for reduction in test-suite size and for reduction in fault-detection effectiveness. Further differences in test suites stem from the fact that the test pools used in the WHLM study as sources for test suites did not necessarily

contain any minimum number of test cases per covered item. These differences may contribute to reduced redundancy in test coverage within test suites, and reduce the likelihood that test-suite reduction will exclude fault-revealing test cases.

Fourth, another factor involves the specific test cases included in the test-case pools. To illustrate, we reproduce, in Figure 13, data on the WHLM study presented in [20]. The table lists the ten programs used in the WHLM study (column 1), the total number of faulty versions of those programs utilized (column 2), and the number of test cases in the test pools created for those programs (column 3). The next two columns report data obtained when the researchers used ATAC to reduce the entire test pools for these programs: column 4 indicates the size of the reduced test pool and column 5 indicates the total number of faults missed by the reduced pools. From this data we derived column 6, the number of faults detected by the reduced test pools.

1	2	3	4	5	6
program	number of faults	test pool size	reduced test-suite size	number of faults missed	number of faults detected
cal	20	162	6	1	19
checkeq	20	166	3	1	19
col	30	156	3	1	29
comm	15	754	11	3	12
crypt	15	156	2	0	15
look	15	193	6	2	13
sort	23	997	11	1	22
spline	13	700	5	1	12
tr	12	870	2	4	8
uniq	18	431	5	0	18

Table 13: Fault detection abilities of test cases used in the WHLM study.

Consider the second, fourth, and sixth columns for program `crypt`. The implications of this data are that two test cases in the test pool for `crypt` were able to detect all 15 faults in that program. Similarly, 3 test cases in the test pool for `col` were able to detect 29 of the 30 faults in that program. These are powerful test cases in relation to the faults; similarly powerful test cases appear to exist for the other programs. When such test cases are included in test suites, reduced versions of those suites may well exhibit little loss in fault-detection effectiveness.

This data, and its indication of the presence of powerful test cases in the WHLM test pools, suggests why reduced *test pools* retain fault-detection effectiveness in the WHLM studies. We cannot know, however, the extent to which such powerful test cases in the WHLM test pools are distributed among the WHLM *test suites*, nor can we know that, distributed among those suites, they would necessarily have been selected by ATAC. Nevertheless, the data supports a conjecture: characteristics of the test cases in a test suite, and their relation in terms of coverage and fault-exposing potential to the subject programs, can affect the performance of test-suite reduction techniques.

A fifth possible factor behind the difference in fault-detection effectiveness results involves the types of faults utilized in the studies. Our experiments with the Siemens programs, and the WHLM study, both

utilized seeded faults that may accurately be described as "mutation-like". However, all of the faults utilized in our study were Quartile I faults, whereas only 41% of the faults used in the WHLM study were Quartile I faults. Easily detected faults are less likely in general to go unrecognized in reduced test suites than faults that are more difficult to detect; thus, we would expect our results overall to show greater reductions in fault-detection effectiveness than the WHLM study. However, the authors of the WHLM study did separately report results for Quartile I faults, and in their study, reduced test suites missed few of these faults.

Finally, focusing only on single factors such as program characteristics, fault types, and test suite design masks the fact that these factors interact. This fact can be further illustrated as follows. In reporting the results of the WHLM study, the researchers also consider the effects of fault difficulty on test suite reduction, and conclude that:

> Faults which can be detected by many test cases ... are likely to be detected after minimization, whereas those which can be detected by only a few test cases are potential candidates for not being detected after minimization [20, page 367].

The researchers conclude that this explains why Quartile I and Quartile II faults are lost more often in the studies.

It seems intuitively obvious that the ease with which a fault can be detected should play a part in determining whether test suite reduction may cause that fault to go undetected. In this context, however, it is important to correctly define "ease of detection". The WHLM researchers measure the ease of detection of a fault in terms of the percentage of test cases, in the test pool, that expose the fault. Where coverage-based test suite reduction is concerned, however, it is not the percentage of program inputs that reach the fault that matter, but rather, the ratio of inputs that reach the fault and reveal it to those that reach the fault. For example, suppose a faulty statement S is executed by k test cases, only one of which reveals the fault in S. Suppose, for simplicity, that statement-based test suite reduction has an equally likely probability of selecting any one of these k test cases. In this case, there is a $(k-1)/k$ probability that test suite reduction will omit the test that reveals the fault in S. This probability is independent of how large k is with respect to the test suite for the program: k may constitute 100% of that test suite, or 1%, and in either case, the probability that statement-based test-suite reduction will omit the fault-revealing test remains the same.

5 Conclusions

A practitioner reading the results of the WHLMP studies may conclude that coverage-based test-suite reduction can offer significant savings, at little cost in test-suite effectiveness. Such a practitioner may be led to employ test-suite reduction techniques in their practice. The results presented in this paper suggest that such a decision may be too hasty. Employing test-suite reduction may, or *may not*, yield substantial losses in the fault-detection effectiveness of test suites.

In fact, our results demonstrate that in some cases, there may be no clear cost-benefit tradeoffs associated with the use of test-suite reduction. In our studies with the Siemens programs, as test-suite size increased,

the savings provided by test-suite reduction increased, but regardless of the level of savings provided, potential losses in fault-detection effectiveness varied widely. This is unfortunate, because as testers, we prefer situations in which the risks inherent in our testing processes relate measurably to the effort we put into those processes; in such cases we can balance those risks against the costs of our efforts. Thus, we would prefer that the risks inherent in reducing test-suite size be related to the potential for savings resulting from reducing test-suite size. The results of this study suggest that such a relationship does not always hold.

Thus, the single most significant result of the work reported in this paper is this: the work suggests that our understanding of test suite reduction and its potential cost-benefits is not complete. We can suggest factors that influence those cost-benefits, but we cannot yet provide predictors that will let practitioners determine what cost-benefits will apply in their particular situations.

This work also has implications for researchers. Given the likelihood that some factors other than test-suite size influence the reduction in fault-detection effectiveness that attends test-suite reduction, a possible conclusion is that we may not want to reduce test suites strictly in terms of code coverage. Alternative test-suite reduction strategies whose cost-benefit tradeoffs are more clear could be beneficial.

Also of interest to researchers, our analysis suggests that the characteristics of programs, faults, and test cases used in experimentation on test suite reduction techniques may be an important factor in determining the results of that experimentation, and may lead us to mistake the effects of non-representative workloads for effects of the techniques that we are investigating. This criticism applies as much to this current study as to previous studies. Together, the two studies support only existentially quantified conclusions: there exist cases where test suite reduction is cost-effective, and there exist cases in which it is not cost-effective. Truly generalizable results can follow only from additional understanding and careful control of the various factors that affect the cost-effectiveness of test-suite reduction.

This work provides a stepping stone from which such results can be pursued. Ultimately we hope that such results can yield an understanding of, and a road to the practical use of, test suite reduction techniques.

6 Acknowledgements

This work was supported in part by grants from Microsoft, Inc. to Ohio State University and Oregon State University, by National Science Foundation Faculty Early Career Development Award CCR-9703108 to Oregon State University, by National Science Foundation Award CCR-9707792 to Ohio State University and Oregon State University, and by National Science Foundation National Young Investigator Award CCR-9696157 to Ohio State University. Siemens Corporate Research supplied the subject programs. Alberto Pasquini, Phyllis Frankl, and Filip Vokolos provided the Space program and many of its test cases. Chengyun Chu assisted with further preparation of the Space program and development of its test cases.

References

[1] M. Balcer, W. Hasling, and T. Ostrand. Automatic generation of test scripts from formal test specifications. In *Proceedings of the 3rd Symposium on Software Testing, Analysis, and Verification*, pages 210–218, December 1989.

[2] T. Y. Chen and M. F. Lau. Dividing strategies for the optimization of a test suite. *Information Processing Letters*, 60(3):135–141, March 1996.

[3] P. G. Frankl and S. 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.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman, New York, 1979.

[5] T. L. Graves, M. J. Harrold, J-M Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. In *Proceedings of the 20th International Conference on Software Engineering*, April 1998.

[6] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, July 1993.

[7] M. J. Harrold and G. Rothermel. Aristotle: A system for research on and development of program analysis based tools. Technical Report OSU-CISRC-3/97-TR17, The Ohio State University, Mar 1997.

[8] J. R. Horgan and S. A. London. ATAC: A data flow coverage testing tool for C. In *Proceedings of the Symposium on Assessment of Quality Software Development Tools*, pages 2–10, May 1992.

[9] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Eng.*, pages 191–200, May 1994.

[10] J. Laski and B. Korel. A data flow oriented program testing strategy. *IEEE Transactions on Software Engineering*, 9(3):347–354, May 1993.

[11] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics.* W.H. Freeman and Company, New York, NY, 3rd edition, 1999.

[12] S. C. Ntafos. On required element testing. *IEEE Transactions on Software Engineering*, 10(6), November 1984.

[13] J. Offutt, J. Pan, and J. M. Voas. Procedures for reducing the size of coverage-based test sets. In *Proceedings of the Twelfth International Conference on Testing Computer Software*, pages 111–123, June 1995.

[14] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6), June 1988.

[15] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375, April 1985.

[16] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.

[17] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Proceedings of the International Conference on Software Maintenance*, pages 44–53, November 1998.

[18] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur. Effect of test set size and block coverage on the fault detection effectiveness. In *Proceedings of the Fifth International Symposium on Software Reliability Engineering*, pages 230–238, November 1994.

[19] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur. Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th International Conference on Software Engineering*, pages 41–50, April 1995.

[20] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur. Effect of test set minimization on fault detection effectiveness. *Software – Practice and Experience*, 28(4):347–369, apr 1998.

[21] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini. Test set size minimization and fault detection effectiveness: A case study in a space application. In *Proceedings of the 21st Annual International Computer Software & Applications Conference*, pages 522–528, August 1997.