On-Demand Test Suite Reduction

Dan Hao*, Lu Zhang*, Xingxia Wu*, Hong Mei*, Gregg Rothermel†
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China
Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China
{haod, zhanglu, wuxx10, meih}@sei.pku.edu.cn
†Department of Computer Science and Engineering, University of Nebraska, Lincoln, 68588, USA
grother@cse.unl.edu

***Abstract*—Most test suite reduction techniques aim to select, from a given test suite, a minimal representative subset of test cases that retains the same code coverage as the suite. Empirical studies have shown, however, that test suites reduced in this manner may lose fault detection capability. Techniques have been proposed to retain certain redundant test cases in the reduced test suite so as to reduce the loss in fault-detection capability, but these still do concede some degree of loss. Thus, these techniques may be applicable only in cases where loose demands are placed on the upper limit of loss in fault-detection capability. In this work we present an *on-demand test suite reduction* approach, which attempts to select a representative subset satisfying the same test requirements as an initial test suite conceding at most $l\%$ loss in fault-detection capability for at least $c\%$ of the instances in which it is applied. Our technique collects statistics about loss in fault-detection capability at the level of individual statements and models the problem of test suite reduction as an integer linear programming problem. We have evaluated our approach in the contexts of three scenarios in which it might be used. Our results show that most test suites reduced by our approach satisfy given fault detection capability demands, and that the approach compares favorably with an existing test suite reduction approach.**

***Keywords*-software testing; test suite reduction; on-demand**

I. Introduction

During regression testing, test engineers are often faced with test suites containing large numbers of test cases [1]. In some cases, executing all of these test cases can be excessively time consuming. Test suite reduction techniques [2], [3], [4], [5], [6], [7] attempt to extract, from a given test suite, a representative subset of test cases satisfying the same testing requirements as the given suite, to reduce the time required for testing.

Most existing test suite reduction techniques (e.g., [2], [3], [4]) aim to select a minimal representative subset of a test suite. In such techniques, a code-coverage criterion (e.g., statement coverage or branch coverage) serves as the test requirements, and the selected minimal representative subset retains the same code coverage as the original test suite. However, even when reduced test suites retain coverage, research has shown that the suites may lose fault detection ability, in some cases dramatically [8], [9].

To limit losses in fault-detection capability, some recent test suite reduction techniques [5], [10] intentionally retain redundant test cases in the reduced subset. However, these techniques still concede some degree of loss in fault-detection capability. Therefore, these techniques may be applicable only in cases where relatively loose demands on fault-detection ability exist. Even worse, our own empirical data (reported in Section II) shows that even when a test suite reduction technique concedes only a small average loss in fault-detection capability, the probability that the technique can concede a much larger loss in fault detection given a particular test suite is still non-negligible. Thus, a test suite reduction technique with a small average loss in fault-detection capability may still not be applicable in situations where strict demands are placed on the upper limits of loss in fault-detection capability.

To allow engineers to set specific upper limits on loss in fault-detection capability in test suite reduction, we propose an *on-demand test suite reduction* approach. Given a confidence level $c\%$ and an upper limit $l\%$ on acceptable loss in fault-detection capability, our approach aims to select a representative subset that satisfies the same test requirements as a given test suite and concedes at most $l\%$ loss in fault-detection capability in at least $c\%$ of the instances in which it is applied. Our approach works as follows. First, we collect statistics on loss in fault-detection capability at the level of individual statements for various levels of confidence, and use these to construct a fault-detection-loss table. Second, based on these statistics, we model the problem of on-demand test suite reduction as an Integer Linear Programming (ILP) problem. We define two ILP models for on-demand test suite reduction using either local constraints or global constraints. Third, by solving the ILP problem, we produce a representative subset for on-demand test suite reduction.

We have evaluated our technique in three scenarios in which it could be employed in practice, applying it to eight C programs and one Java program. In the first scenario, we used part of each C program to build a fault-detection-loss table and used that table to apply test suite reduction to the other parts of the programs. In the second scenario, we used one version of the Java program to construct a fault-detection-loss table and used that table to apply test suite reduction to its subsequent versions. In the third scenario,

we used some of the C programs to construct a fault-detection-loss table and then used that to apply test suite reduction to the other C programs. Our results show that the proposed technique (using either local constraints or global constraints) is usually effective for satisfying a given demand (i.e., an upper limit of loss in fault-detection capability and a confidence level) during test suite reduction in each of the three scenarios. Moreover, the sizes of reduced test suites obtained using local constraints are usually larger than or equal to those obtained using global constraints, and the percentage of test suites found to satisfy given demands in the former case is larger than or equal to the percentage found to satisfy demands in the latter.

Finally, we compare our results to those produced by Harrold et al.'s technique [2] (abbreviated as the HGS algorithm), which cannot be used to reduce the size of a test suite while also guaranteeing that any loss in fault-detection capability satisfies engineers' demand. Our results show that to obtain a loss in fault-detection capability similar to that of the HGS algorithm, the on-demand technique selected at most 80% more test cases than the HGS algorithm.

The main contributions of this paper are as follows: (1) empirical evidence that the distribution of losses in fault-detection capabilities of existing test suite reduction techniques is very much skewed across the basis pairs of original test suites and faulty versions; (2) to the best of our knowledge, the first approach to on-demand test suite reduction; and (3) empirical studies evaluating the effectiveness of our approach to on-demand test suite reduction.

II. Motivation

Prior empirical studies of test-suite reduction have reported only *average losses* in the fault-detection capabilities of test suites. To further motivate our research, we conducted an exploratory study investigating the *distribution of losses* in fault-detection capabilities exhibited by existing test-suite reduction techniques.

Study Setup: We used a set of small programs that have been extensively utilized in test suite reduction studies [4], [5], [11]; namely, the Siemens programs. Based on the pool of test cases for each program, we constructed a set of test suites in a manner similar to that used by Jeffrey and Gupta [5] in their study of their test suite reduction technique. When constructing each test suite, we first randomly selected from the test case pool $X * LOC$ test cases (where X is a random variable in the range [0, 0.5] and LOC represents the number of lines of executable code in the program). Then we continued selecting test cases and adding them to the test suite as long as they increased statement coverage, until the test suite was statement-coverage adequate. In this way we constructed 100 test suites for each program. For each test suite for each program, we then applied two representative test suite reduction techniques (i.e., Harrold et al.'s [2] and Jeffrey and Gupta's [5]).

Table I
Loss in Fault-Detection Capability for Two Test Suite Reduction Techniques

Program	Harrold et al.'s Technique				Jeffrey and Gupta's Technique			
	Min.	Max.	Avg.	SD.	Min.	Max.	Avg.	SD.
print_tokens	0.000	0.800	0.243	0.221	0.000	0.800	0.221	0.217
print_tokens2	0.000	0.444	0.157	0.128	0.000	0.444	0.116	0.111
replace	0.000	0.913	0.517	0.208	0.000	0.727	0.239	0.163
schedule	0.000	1.000	0.592	0.261	0.000	1.000	0.460	0.243
schedule2	0.000	1.000	0.441	0.333	0.000	1.000	0.286	0.317
tcas	0.000	0.920	0.449	0.249	0.000	0.867	0.373	0.229
tot_info	0.000	0.750	0.326	0.216	0.000	0.565	0.199	0.175

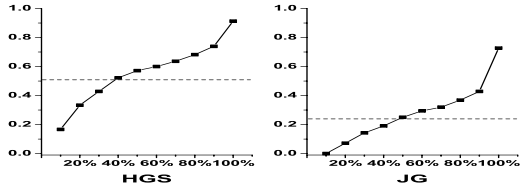

Figure 1. Fault-Detection Loss Distribution for $replace$ (horizontal axes depict percentages of test suites, and vertical axes depict the upper limit on loss in fault-detection capability for that percentage of test suites.)

Observations: Table I depicts statistics on losses[1] in fault-detection capability for the two representative techniques, where "Avg." denotes the average loss and "SD." denotes the standard deviations of loss. For Harrold et al's technique, the average losses in fault-detection capability range from 0.157 to 0.592. Moreover, the ranges of loss in fault-detection capability are all quite large. For example, for $schedule$ and $schedule2$, some reduced test suites concede no loss in fault-detection capability, while others concede 100% loss in fault-detection capability. The standard deviations of loss in fault-detection capability (i.e., from 0.128 to 0.333) also corroborate this observation. The results of Jeffrey and Gupta's technique provide a similar view on the distribution of losses in fault-detection capability.

Figure 1 presents the ranges of losses in fault-detection capability across different test suites of Harrold et al.'s technique (abbreviated as HGS in the figure) as well as Jeffrey and Gupta's technique (abbreviated as JG in the figure) for $replace$. (Results for the other objects are similar and are omitted for space considerations.) Specifically, the curves in this figure show the upper limits on losses in fault-detection capability across percentages of test suites. In each subfigure, the dotted line shows the average loss in fault-detection capability across all the program's test suites. From this figure, we observe that there are a substantial number of reduced test suites whose loss in fault-detection capability

[1]The loss in fault-detection capability is defined as the ratio between the number of faults that become unexposed following test suite reduction and the total number of faults detected by the original test suite. Furthermore, the loss in fault-detection capability can be calculated in a way similar to Formula 1 (Section IV). Specifically, if reduced test suites concede no loss in fault-detection capability, the loss is 0; if reduced test suites concede 100% loss in fault-detection capability, the loss is 1.

is significantly larger than the average loss in fault-detection capability. Although the average loss in fault-detection capability of the HGS algorithm is 0.517 for $replace$, more than 20% of the reduced test suites have losses in fault-detection capability larger than 0.6. The average loss in fault-detection capability of the JG algorithm is 0.239, but more than 20% of the reduced test suites have losses in fault-detection capability larger than 0.4.

These observations show that it may be difficult for the actual loss in fault-detection capability on a given program and test suite to be similar to the average loss in fault-detection capability. That is to say, given a specific faulty program and a specific test suite, the probability for the loss in fault-detection capability to be much higher than the average is not negligible when applying an existing technique for test-suite reduction. Therefore, a developer may typically find existing techniques for test-suite reduction inapplicable, if he or she wishes to set an upper limit on loss in fault-detection capability.

III. Problem Definition

We formally define the problem of on-demand test suite reduction as follows. Let T be a test suite, let $l\%$ be an upper limit on loss in fault-detection capability, and let $c\%$ be a confidence level. Let $f_{l,c}(S)$ denote the fact that S is a subset of T and that S's loss in fault-detection capability is at most $l\%$ in at least $c\%$ of circumstances. The problem of on-demand test suite reduction is to find $T' \subseteq T$ such that $f_{l,c}(T')$ and $\forall T'' \subseteq T\ (f_{l,c}(T'') \rightarrow |T'| \leq |T''|)$ (where $|T|$ denotes the size of T).

In the preceding definition, we use $f_{l,c}$ to ensure that the loss of fault-detection capability is at most $l\%$ at confidence level $c\%$. According to statistical theory, if we set c to be a number near 100, it is likely that we can ensure that the loss in fault-detection capability of an arbitrary reduced test suite is at most $l\%$.[2] To ensure that we select as few test cases as possible in on-demand test suite reduction, our definition requires the reduced test suite to be the smallest among all the subsets of test cases satisfying the demand on loss in fault-detection capability.

IV. Our Approach

Similar to previous techniques for test suite reduction, our approach to on-demand test suite reduction uses coverage information. Given a program under test, coverage information can be based on any structural units present in the program (e.g., statements). In this section, we present our approach in terms of statement coverage for simplicity, as the implementation and evaluation of our approach is also based on statement coverage. However, it is straightforward to extend the basic idea of our approach to other types of structural coverage.

[2]In statistical theory, the confidence level is typically set to 90%, 95%, or 99%.

When reducing a test suite T into one of its subsets, T', if a statement s contains a fault that T can detect but T' cannot, the number of times T' covers s must be smaller than the number of times T covers s. That is to say, when there is a difference between the original test suite T and the reduced test suite T' in the number of times statement s is covered, there is a possibility that T' may fail to detect some fault in s. Thus, our approach to on-demand test suite reduction involves two steps. First, we collect statistics on losses in fault-detection capability at the level of individual statements. Second, we define two Integer Linear Programming (ILP) models using the collected statistics to reduce test suites.

A. Collection of Statistics at the Statement Level

Given two integers i_1 and i_2 ($i_1 > i_2$), our goal in the first step is to collect the distribution of losses in fault-detection capability for a statement when coverage of those statements changes from i_1 to i_2. To achieve this goal, we employ the following empirical methodology for sampling.

To provide a sufficient sampling space, we use mutation faults[3] when collecting statistics. Note that previous research (e.g., Andrews et al. [12]) has demonstrated the reasonableness of using mutation faults in experimentation with testing techniques. For any statement that contains mutation faults and has been executed by i_1 test cases in an original test suite T but executed by i_2 test cases in a subset T' of T, we use $CovDiff(i_1, i_2)$ to represent the loss in fault-detection capabilities for that statement when the level of coverage changes from i_1 to i_2. Furthermore, we replace T by T_{i_1} since test suite T contains i_1 test cases that execute some statement, and replace T' by T'_{i_2} since test suite T' contains i_2 test cases that execute the same statement. The value of $CovDiff(i_1, i_2)$ is defined by Formula 1, where $\#Faults(T_{i_1})$ represents the number of mutation faults detected by T_{i_1} and $\#Faults(T'_{i_2})$ represents the number of mutation faults detected by T'_{i_2}.

$$CovDiff(i_1, i_2) = \frac{\#Faults(T_{i_1}) - \#Faults(T'_{i_2})}{\#Faults(T_{i_1})} \quad (1)$$

Based on the preceding definition, if we sample a sufficiently large number of statements and a sufficiently large number of pairs of T_{i_1} and T'_{i_2}, we are able to record a series of values of $CovDiff(i_1, i_2)$ for various pairs of i_1 and i_2 ($i_1 > i_2$). Given a pair of i_1 and i_2 ($i_1 > i_2$), all the values of $CovDiff(i_1, i_2)$ represent the distribution in loss of fault-detection capability for one statement when the level of coverage changes from i_1 to i_2. According to statistical theory, we are primarily interested in the 90th, 95th, and 99th percentiles. That is to say, given the distribution of all

[3]We currently collect statistics based on mutation faults since it is possible to generate a large number of mutation faults for any program. However, when sufficient historical faults are available, our approach can also collect statistics based on those.

Table II
PARTIAL FAULT-DETECTION-LOSS TABLE (c=99)

				...			
...				...			
6	1	1	0.903	0.750	0.543	0.106	0
5	1	1	0.875	0.667	0.188	0	
4	1	0.957	0.833	0.533	0		
3	1	0.947	0.758	0			
2	1	0.939	0				
1	1	0					
	0	**1**	**2**	**3**	**4**	**5**	**6** ...

This table presents the upper limit of loss (i.e., $V_c(i_1, i_2)$) in fault-detection capability in at least $c\%$ circumstances when coverage of statements changes from i_1 (depicted by the first column) to i_2 (depicted by the last row).

$CovDiff(i_1, i_2)$ values for (i_1, i_2) we record the following three values of loss in fault-detection capability: the value that 90% of the $CovDiff(i_1, i_2)$ values are no larger than (denoted by $V_{90}(i_1, i_2)$), the value that 95% of the $CovDiff(i_1, i_2)$ values are no larger than (denoted by $V_{95}(i_1, i_2)$), and the value that 99% of the $CovDiff(i_1, i_2)$ values are no larger than (denoted by $V_{99}(i_1, i_2)$). Thus, given confidence level $c\%$ (c=90, 95, or 99), all the values of $V_c(i_1, i_2)$ ($i_1 > i_2$) for different pairs (i_1, i_2) represent the values of loss in fault-detection capability in coverage of one statement at confidence level $c\%$. Note that, when i_2 equals i_1 for a statement, there is no loss of fault-detection capability for that statement. That is to say, $V_c(i, i)$ would always be 0 for any c = 90, 95, or 99 and any i.

Following the preceding procedure, we generate a fault-detection-loss table for a program ignoring the difference within statements. Table II illustrates part of a fault-detection-loss table whose statistics are collected based on the Siemens suite. According to our empirical study (and shown by this table), for any i, $V_c(i, 0)$ and $V_c(i, i)$ are always 1 and 0, respectively. Moreover, $V_c(i, j_1)$ is always no less than $V_c(i, j_2)$ when $j_1 \leq j_2$. This observation complies with the intuition that less coverage of a statement may reduce the fault-detection capability of a test suite.

B. On-Demand Test Suite Reduction via ILP

Based on the statistics collected in the first step of our approach, we use two methods for modeling on-demand test suite reduction as an Integer Linear Programming problem. The two ILP models share the same objective function and the same decision variables, but use different constraints.

1) Objective Function: As the problem of on-demand test suite reduction requires the smallest reduced test suite satisfying the demand, both ILP problems use the objective function defined in Formula 2, where x_i is a Boolean variable that represents whether test case t_i has been selected for inclusion in the reduced test suite and is defined below.

$$min \sum_{i=1}^{n} x_i \tag{2}$$

2) Decision Variables: For test suite $T = \{t_1, t_2, \ldots, t_n\}$, we use n Boolean decision variables (denoted by x_i ($1 \leq i \leq n$)) to represent whether test case t_i has been selected for the reduced test suite T'. The n Boolean decision variables are defined in Formula 3.

$$x_i = \begin{cases} 1, & \text{if test case } t_i \text{ is selected;} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

For a program P under test, consisting of m statements ($S = \{s_1, s_2, \ldots, s_m\}$), we use the Boolean decision variables defined in Formula 4 to represent changes in coverage when reducing T into T'.

$$w_{j,q} = \begin{cases} 1, & \text{iff } q \text{ test cases in } T' \text{ cover } s_j; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

In Formula 4, given one j, if there are p_j test cases in T covering s_j, q must be less than or equal to p_j and q must be greater than or equal to 1. Therefore, there are p_j decision variables of the form $w_{j,q}$ for s_j. That is to say, the total number of decision variables of the form $w_{j,q}$ is $\sum_{j=1}^{m} p_j$.

3) Constraints: For any $1 \leq j \leq m$, exactly one variable of the form $w_{j,q}$ can be true, since there is just one value for the number of test cases in T' that cover s_j. Thus, we need the constraints described in Formula 5, where p_j is the number of test cases in T that cover s_j.

$$\sum_{q=1}^{p_j} w_{j,q} = 1 \tag{5}$$

Furthermore, for any $1 \leq j \leq m$, the number of test cases in T' covering s_j can be represented either by variables of the form $w_{j,q}$ or by variables of the form x_i. Therefore, we need a set of constraints to ensure that the two manners of representing coverage information comply with each other. To define these constraints, we need the coverage information shown in Formula 6 denoting whether test case t_i ($1 \leq i \leq n$) in T covers statement s_j ($1 \leq j \leq m$).

$$C(i, j) = \begin{cases} 1, & \text{if test case } t_i \text{ covers statement } s_j; \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

The number of test cases in T' covering s_j can be represented as $\sum_{q=1}^{p_j} w_{j,q} * q$ or $\sum_{i=1}^{n} x_i * C(i, j)$. Therefore, we have the set of constraints shown in Formula 7.

$$\sum_{i=1}^{n} x_i * C(i, j) = \sum_{q=1}^{p_j} w_{j,q} * q \tag{7}$$

Finally, we need a set of constraints to ensure that T' satisfies the developer's demand. T' should concede at most $l\%$ loss in fault-detection capability at confidence level $c\%$. We have two ways to define this set of constraints.

The first way to define the constraints is to use local constraints. We define a set of constraints to ensure that T' concedes at most $l\%$ loss in fault-detection capability at

confidence level $c\%$ for **each statement** in the program. In particular, Formula 8 defines the local constraints on loss in fault-detection capability for any j ($1 \leq j \leq m$).

$$\sum_{q=1}^{p_j} w_{j,q} * V_c(p_j, q) \leq l\% \tag{8}$$

Note that, in Formula 8, $V_c(p_j, q)$ represents the loss in fault-detection capability for one statement at confidence level $c\%$ when the coverage changes from p_j to q.

The second way to define the constraints is to use one global constraint. We define one constraint to ensure that T' concedes at most $l\%$ loss in fault-detection capability at confidence level $c\%$ for **the program under test**. In particular, Formula 9 defines the global constraint on loss in fault-detection capability.

$$\sum_{j=1}^{m} \sum_{q=1}^{p_j} w_{j,q} * V_c(p_j, q) \leq m * l\% \tag{9}$$

Thus, we have two ILP models for on-demand test suite reduction. One ILP model uses the constraints in Formulae 5, 7, and 8, while the other ILP model uses the constraints in Formulae 5, 7, and 9. Note that the global constraint is looser than the local constraints. Thus, the ILP model using the global constraint may allow us to select fewer test cases in the reduced test suite T' than when using the local constraints.

C. Practical Simplification

In the preceding ILP models, the number of decision variables could be $|T|+|T| * |S|$ in the worst case, because one statement may be covered by all the test cases in T. Since the number of decision variables is quadratic in the size of input, the computational cost of solving the two ILP models could be too large to be practical.

Intuitively, when a test suite contains a large number of test cases covering a statement, the probability for that test suite to detect all possible faults in that statement should be very near 100%. That is, the value of $V_c(i_1, i_2)$ should be very near to 0 for any c = 90, 95, or 99 and any $i_1 > i_2$, when the value of i_2 become quite large. Actually, our experience in collecting values of $V_c(i_1, i_2)$ also supports this intuition.

Based on this intuition, we are able to simplify our ILP models by setting up a threshold Cov_{max} and considering $V_c(i_1, i_2)$ to be 0 for any $i_1 > i_2 \geq Cov_{max}$. Therefore, for any j, if there are p_j test cases in T covering statement s_j, we can have $\beta(j)$ (instead of p_j) decision variables of the form $w_{j,q}(1 \leq q \leq \beta(j))$ for simplification, where we use $\beta(j)$ to denote $min(Cov_{max}, p_j)$. Among these $\beta(j)$ decision variables, the meaning of $w_{j,\beta(j)}$ is different from the definition in Formula 4, and the meaning of other decision variables remains the same. In our simplification, the value of $w_{j,\beta(j)}$ is 1 if and only if at least $\beta(j)$ test cases in T' cover s_j. Note that when $\beta(j)$ is equal to p_j, the meaning of $w_{j,\beta(j)}$ is still the same as that defined in Formula 4, because at most p_j test cases in T' can cover s_j.

We further simplify constraints in Formulae 5, 7, 8, and 9 as follows. First, we replace p_j with $\beta(j)$ in Formula 5. Second, we change the equalities in Formula 7 into the inequalities in Formula 10. The change from Formula 7 to Formula 10 is necessary, because when more than Cov_{max} test cases in T' cover statement s_j, it would be infeasible for $\sum_{i=1}^{n} x_i * C(i, j)$ to be equal to $\sum_{q=1}^{\beta(j)} w_{j,q} * q$. Note that, as the objective function requires the number of test cases in T' to be as small as possible, $\sum_{i=1}^{n} x_i * C(i, j)$ would always be equal to $\sum_{q=1}^{\beta(j)} w_{j,q} * q$ when feasible.

$$\sum_{i=1}^{n} x_i * C(i, j) \geq \sum_{q=1}^{\beta(j)} w_{j,q} * q \tag{10}$$

Third, we change Formulae 8 and 9 to Formulae 11 and 12, respectively. Note that, as we deem the value of $V_c(p_j, q)(\beta(j) \leq q \leq p_j)$ as 0, the upper bound of the summarization can be simplified to $\beta(j) - 1$.

$$\sum_{q=1}^{\beta(j)-1} w_{j,q} * V_c(p_j, q) \leq l\% \tag{11}$$

$$\sum_{j=1}^{m} \sum_{q=1}^{\beta(j)-1} w_{j,q} * V_c(p_j, q) \leq m * l\% \tag{12}$$

After simplification, the number of decision variables in the two ILP models is at most $|T|+Cov_{max}*|S|$. That is, the number of decision variables in our simplified ILP models becomes linear in the size of input. Thus, we are able to select a proper value of Cov_{max} to make the computational cost of solving the two ILP models acceptable.

V. Empirical Studies

We have conducted three empirical studies to investigate the performance[4] of our on-demand test suite reduction approach in three scenarios.

In scenario (1), engineers may gather statistics on loss in fault-detection capability from part of a program, and use the collected statistics to perform on-demand test suite reduction on other portions of the program. In this case we are interested in the following research question (**RQ1**): Is the on-demand test suite reduction approach effective at satisfying engineers' demands when being applied within a single program?

In scenario (2), engineers may gather statistics on loss in fault-detection capability on a version of a program, and use those statistics to perform on-demand test suite reduction on subsequent versions of the program, i.e., in the context

[4]Due to space limitations, we evaluate only the effectiveness of the proposed approach in this paper; we will evaluate the cost of the approach in our future work.

Table III
OBJECTS OF STUDY

Program	Abbreviation	Description	LOC	Test Pool
print_tokens	pt	lexical analyzer	203	4,072
print_tokens2	pt2	lexical analyzer	203	4,057
replace	rep	pattern replacer	289	5,542
schedule	sch	priority scheduler	162	2,627
schedule2	sch2	priority scheduler	144	2,683
tcas	tca	altitude separator	67	1,592
tot_info	tot	info measurer	136	1,026
space	spa	ADL interpreter	6,218	13,585
jtopas-v0	jt0	parsing text data	1,814	95
jtopas-v1	jt1	parsing text data	1,878	126
jtopas-v2	jt2	parsing text data	2,010	128

of software evolution. In this case we are interested in the following research question (**RQ2**): Is the on-demand test suite reduction approach effective at satisfying engineers' demands when being applied to subsequent versions of a program?

In scenario (3), engineers may gather statistics on loss in fault-detection capability on a set of similar programs, and use those statistics to perform on-demand test suite reduction on other similar programs. In this case we are interested in the following research question (**RQ3**): Is the on-demand test suite reduction approach effective at satisfying engineers' demand when being applied to sets of similar programs?

A. Object Programs

Table III describes the objects used in our studies, all of which are available in the Software-artifact Infrastructure Repository (SIR) [13]. In addition to the seven programs of the Siemens suite and $space$, which are written in C, the other objects are three versions of $jtopas$, which are written in Java. The second column provides abbreviations used in the following sections, the third describes program functionality, and the fourth lists the total number of lines of executable code in the programs. Each of the object programs comes with existing test cases, and the rightmost column of Table III lists the numbers of those test cases.

The first study used all eight C programs listed in Table III, the second study used $jtopas$ and its sequential versions, and the third study used the seven programs of the Siemens suite.

B. Study Process

In each of our three studies, we first prepared test suites and faulty programs with which to collect statistics on loss in fault-detection capability. As all of the objects have a test pool, for each object, by randomly selecting test cases that increase statement coverage we constructed 4900 test suites, consisting of 100 of each size from 2 test cases to 50 test cases. If the statement coverage of a test suite under construction reaches the same level of statement coverage as the entire test pool prior to reaching its desired size, we selected additional test cases randomly until the desired size was reached.

To collect statistics on loss in fault-detection capability due to reduced coverage of faulty statements, we first used Proteum [14] and MuJava[5] [15] to generate all mutants for each object program, and then constructed mutant groups by grouping all the mutants whose mutation operation occurred in the same statement. In statistics collection, each mutant group is viewed as a multiple-fault program so that the loss in fault-detection capability in each application of test suite reduction comes from the reduced coverage of the same faulty statement.[6] Moreover, groups that contain fewer than three mutants were removed to reduce the possibility of biased data.

For each constructed test suite T, we generated a reduced test suite T' by randomly removing i ($1 \leq i \leq |T|$) test cases from T, recording (1) the number of faults detected by T (denoted by $\#Faults(T)$), (2) the number of faults detected by the reduced test suite T' (denoted by $\#Faults(T')$), (3) the number of test cases in T executing the mutated statement (denoted by i_1), and (4) the number of test cases in T' executing the mutated statement (denoted by i_2). Then we calculated the loss in fault-detection capability from i_1 to i_2 using Formula 1 and thus constructed fault-detection-loss tables at confidence levels $c\%$, where c was set to 90%, 95%, and 99% as described in Section IV-A.

The foregoing process provided fault-detection capability statistics; next, we needed to prepare test suites and faulty programs with which to evaluate on-demand test suite reduction using those statistics. For each object program in the Siemens suite, we used the 100 test suites constructed for use in the preliminary study described in Section II. For $space$, we constructed 100 test suites in the manner described in Section II. Since our Java programs have a small number of test cases in their test pool, we constructed only 20 test suites for these programs in the same manner followed for $space$.

Because each object program is provided with only a small number of faulty versions, for our purposes we constructed multiple-fault C and Java programs using mutants generated by the mutation analysis tools Proteum and MuJava. To reduce the potential for biased data, we followed a procedure similar to that used by Hutchins et al. [16] and retained only mutants that were "neither too easy nor too hard to detect" [16] by removing mutants detected by more than 90% of the test cases in the test pool and removing mutants detected by fewer than 10% of the test cases in the test pool. Finally, as the number of faults within a real program is likely to be related in some way to its number of lines of code, for each object, we constructed multiple-fault programs whose number of faults (denoted as n_f) is

[5]http://cs.gmu.edu/~offutt/mujava/.

[6]Those mutants whose mutation operation occurred in different statements cannot be used in statistics collection because in that case, it is difficult to determine how the reduced coverage of particular statements contributes to the loss in fault-detection capability in test suite reduction.

$\frac{LOC}{10}$, $\frac{LOC}{50}$, $\frac{LOC}{100}$, $\frac{LOC}{500}$, or $\frac{LOC}{1000}$, respectively. Specifically, for each given number of faults n_f, we constructed five multiple-fault programs each of which contains n_f randomly selected mutants.

Given the foregoing, we applied our on-demand test suite reduction techniques (using local and global constraints) to each of these test suites, recording $|T|$, $|T'|$, $\#Faults(T)$, and $\#Faults(T')$. We set the engineers' demands on loss in fault-detection capability (i.e., the upper limit of loss in fault-detection capability) to 5%, 15%, 25%, 35%, and 50%. We used IBM's ILOG CFLEX[7] to solve the two ILP models (i.e., the two practical simplified models in Section IV-C) used in our test suite reduction technique.

Because our first study aims to evaluate the proposed approach within programs, we randomly selected 50% of the statements in each C object to collect statistics on loss in fault-detection capability, ignoring the other statements. That is, the mutants whose mutation operations occurred in the remaining 50% of the statements were removed in the process of statistics collection in the first study. In the process of on-demand test suite reduction using the collected statistics, we first used only the remaining statements by focusing on only the multiple-fault programs whose mutation operations occur in these remaining statements, and then we used the whole program.

In our second study, we used each version of $jtopas$ to collect statistics on loss in fault-detection capability and the subsequent versions to perform on-demand test suite reduction using the collected statistics.

In our third study, we used one program from the Siemens suite to perform on-demand test suite reduction and the other programs to collect statistics on loss in fault-detection capability.

C. Measurements

To assess the results of the application of our test suite reduction algorithm, as independent variables we used test suite size reduction and fault-detection capability loss. Test suite size reduction is defined as $\frac{|T|-|T'|}{|T|} * 100\%$, and fault-detection capability loss is defined as $\frac{\#Faults(T)-\#Faults(T')}{\#Faults(T)}$.

D. Threats to Validity

The primary internal threat to validity for our study relates to possible problems in the implementations of our techniques. To reduce this threat, we used IBM's integer linear programming tool ILOG CFLEX to implement the on-demand test suite reduction technique and reviewed all of our code before conducting the experiment. The primary external threat to validity for our study relates to the particular objects studied, which may be not representative. A second threat relates to the program and the faults used in the faulty programs, and a third threat lies in the test suites constructed for the process of test suite reduction. These threats can be addressed only through further studies using other larger object programs, faults, and test suites. Finally, the measures we use to assess results do not include other sources of costs and effectiveness that may matter in practice, such as the severity of faults and the cost of the algorithms applied.

[7]http://www.ibm.com/software/websphere/ilog.

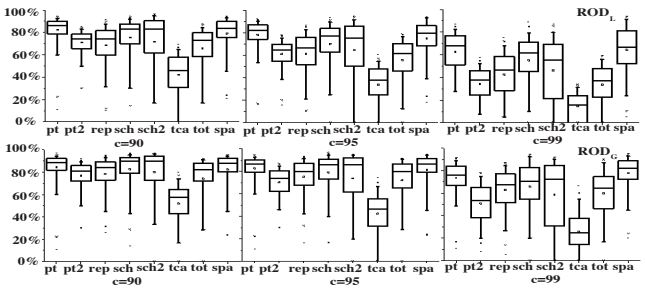

Figure 2. Test suite size reduction in Study I (the horizontal axis presents abbreviated object names, and the vertical axis depicts test suite size reduction using the formula given in Section V-C.)

VI. Results and Analysis

A. Study I – Within Programs

Table IV presents the percentages of reduced test suites whose actual loss in fault-detection capability satisfies engineers' demands (i.e., the specified upper limit on loss in fault-detection capability) for each object program when part of the object is used to calculate loss of fault-detection capability for the other part of the object or the entire object is used in on-demand test suite reduction.

In the table, results that are smaller than the confidence level $c\%$ given by the engineers are presented in bold font. Most results are larger than or equal to the confidence level; that is, most test suites generated by the approach using either local constraints or global constraints satisfy engineers' demands including the limit on loss in fault-detection capability and the confidence level, except for a few cases involving $print\_tokens$, $schedule$, $tcas$ and tot_info. Moreover, for any given object, upper limit of loss in fault-detection capability $l\%$, and confidence level $c\%$, the percentage of reduced test suites satisfying engineers' demands achieved by our approach using local constraints is usually larger than or equal to that achieved by our approach using global constraints.

Figure 2 presents the distributions of test suite size reduction for our approach where actual loss in fault-detection capability satisfies engineers' demands. To distinguish the two different ILP models (i.e., the ILP model using local constraints and the ILP model using global constraints), we abbreviate the former as ROD_L and the latter as ROD_G. As the figure shows, as we increase the confidence level, test suite size reduction decreases, and thus the size of reduced test suites increases. That is, our approach increases

Table IV
PCT. OF RESULTS SATISFYING ENGINEERS' DEMAND, STUDY I

c	l	pt	pt2	rep	sch	sch2	tca	tot	spa
partial object is used in on-demand test suite reduction using local constraints									
90	5	**87.50**	99.67	99.73	**89.80**	98.20	95.80	95.60	100.00
	15	100.00	99.60	99.53	98.80	99.00	**87.60**	98.20	100.00
	25	100.00	100.00	99.47	97.20	98.90	96.20	98.30	100.00
	35	100.00	100.00	99.40	99.40	98.30	99.60	98.00	100.00
	50	100.00	100.00	100.00	99.50	100.00	100.00	99.70	100.00
95	5	**92.70**	100.00	99.80	**94.70**	99.40	99.20	97.20	100.00
	15	100.00	99.87	99.67	99.60	99.90	96.00	99.60	100.00
	25	100.00	100.00	99.87	99.40	99.60	98.80	99.30	100.00
	35	100.00	100.00	99.80	99.70	99.20	99.80	99.30	100.00
	50	100.00	100.00	100.00	99.90	100.00	100.00	99.90	100.00
99	5	99.00	100.00	100.00	99.90	99.80	100.00	99.70	100.00
	15	100.00	100.00	100.00	99.80	100.00	100.00	100.00	100.00
	25	100.00	100.00	100.00	100.00	100.00	99.80	100.00	100.00
	35	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
entire object is used in on-demand test suite reduction using local constraints									
90	5	94.90	98.80	99.87	**84.90**	99.90	98.40	96.50	100.00
	15	99.00	98.67	99.47	95.30	100.00	96.60	97.10	100.00
	25	99.40	99.67	100.00	94.30	99.70	98.80	97.70	100.00
	35	100.00	99.73	99.73	99.70	98.90	98.60	97.40	100.00
	50	99.90	100.00	100.00	99.80	100.00	100.00	100.00	100.00
95	5	96.80	99.73	99.93	**93.40**	100.00	100.00	98.80	100.00
	15	99.60	99.67	99.80	98.50	100.00	98.40	98.60	100.00
	25	99.30	99.93	100.00	96.40	100.00	100.00	98.60	100.00
	35	100.00	99.93	100.00	99.80	100.00	100.00	98.50	100.00
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
99	5	99.60	100.00	100.00	100.00	100.00	100.00	99.70	100.00
	15	100.00	100.00	100.00	99.90	100.00	100.00	99.60	100.00
	25	100.00	100.00	100.00	99.30	100.00	100.00	99.70	100.00
	35	100.00	100.00	100.00	100.00	100.00	100.00	99.70	100.00
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
partial object is used in on-demand test suite reduction using global constraints									
90	5	**85.10**	99.53	99.00	**86.30**	97.60	95.20	95.20	100.00
	15	100.00	99.67	98.47	**93.50**	98.90	91.00	97.80	100.00
	25	100.00	100.00	98.87	**93.80**	98.90	98.60	97.90	100.00
	35	100.00	100.00	97.67	98.00	96.50	98.60	97.70	100.00
	50	99.90	100.00	100.00	98.40	100.00	99.80	99.90	100.00
95	5	**88.00**	99.93	99.40	**93.90**	99.00	99.60	97.50	100.00
	15	100.00	99.73	99.20	99.50	99.90	98.00	99.30	100.00
	25	100.00	100.00	99.53	98.70	100.00	99.20	99.40	100.00
	35	100.00	100.00	99.20	98.90	99.90	99.60	98.60	100.00
	50	100.00	100.00	100.00	98.80	100.00	100.00	99.90	100.00
99	5	**94.80**	100.00	100.00	99.60	99.90	100.00	99.00	100.00
	15	100.00	100.00	99.93	99.80	100.00	100.00	99.70	100.00
	25	100.00	100.00	99.93	99.80	100.00	100.00	99.80	100.00
	35	100.00	100.00	99.80	100.00	100.00	100.00	99.60	100.00
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
entire object is used in on-demand test suite reduction using global constraints									
90	5	91.50	98.33	98.80	**83.90**	98.70	98.20	91.30	100.00
	15	98.10	98.93	96.80	92.90	99.20	**85.00**	94.50	100.00
	25	98.50	99.80	98.60	**89.50**	99.00	94.80	95.50	100.00
	35	99.80	99.73	98.13	97.40	98.40	94.60	95.50	100.00
	50	99.80	100.00	100.00	98.70	99.80	99.40	100.00	100.00
95	5	**93.80**	99.40	99.53	95.10	100.00	100.00	**94.00**	100.00
	15	99.20	99.47	99.00	97.50	100.00	97.40	**94.50**	100.00
	25	99.00	100.00	99.60	96.00	99.70	99.00	95.10	100.00
	35	100.00	99.93	99.27	99.30	99.50	98.00	95.50	100.00
	50	99.90	100.00	100.00	99.50	100.00	99.80	100.00	100.00
99	5	**97.40**	100.00	100.00	99.90	100.00	100.00	**98.70**	100.00
	15	99.80	100.00	100.00	99.80	100.00	100.00	**97.70**	100.00
	25	99.80	100.00	99.93	99.10	100.00	100.00	**97.10**	100.00
	35	100.00	100.00	99.93	100.00	99.90	100.00	**96.40**	100.00
	50	100.00	100.00	100.00	99.90	100.00	100.00	100.00	100.00

engineers' confidence in loss of fault-detection capability by selecting more test cases. Note that the test suite size reduction for $tcas$ is much smaller than that for the other object programs. We suspect the reason for this to be that for $tcas$ the number of faulty programs is small and thus, the test suites constructed initially for it are small.

Table V
PCT. OF RESULTS SATISFYING ENGINEERS' DEMAND, STUDY II

c	l	ROD_L			ROD_G		
		jt0-jt1	jt1-jt2	jt0-jt2	jt0-jt1	jt1-jt2	jt0-jt2
90	5	**63.75**	**83.25**	**76.50**	**48.75**	**71.25**	**66.25**
	15	**79.50**	92.75	**89.75**	**70.25**	**71.00**	**69.50**
	25	92.50	97.25	96.00	91.50	93.75	92.50
	35	92.50	97.00	96.50	91.50	94.00	94.00
	50	96.75	100.00	100.00	96.00	100.00	100.00
95	5	**87.75**	99.25	**88.00**	**63.75**	**90.50**	**85.50**
	15	**93.25**	99.75	**94.75**	**72.00**	**86.75**	**86.50**
	25	98.00	99.75	98.25	**92.00**	**94.00**	**94.75**
	35	98.00	99.00	98.25	**92.50**	**94.75**	95.25
	50	99.00	100.00	100.00	95.75	100.00	100.00
99	5	100.00	100.00	100.00	**97.00**	100.00	**96.75**
	15	100.00	100.00	100.00	**81.00**	**91.00**	**92.50**
	25	100.00	99.75	100.00	**94.00**	**98.50**	**98.00**
	35	100.00	99.75	100.00	**93.00**	**97.75**	**98.00**
	50	100.00	100.00	100.00	**96.25**	100.00	100.00

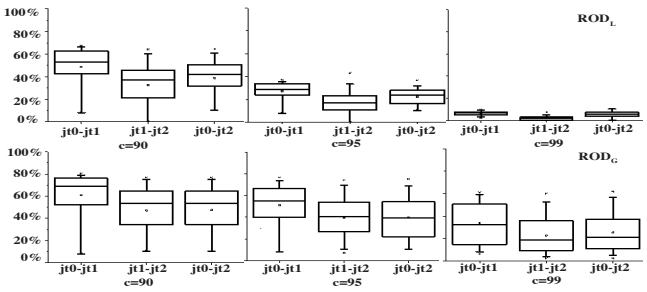

Figure 3. Test suite size reduction in Study II

B. Study II – Between Versions

Table V presents the percentages of reduced test suites whose actual loss in fault-detection capability satisfies engineers' demand when some version of $jtopas$ is used to collect statistics and on-demand test suite reduction is applied to later versions. Specifically, $jti - jtj$ shows that $jtopas - vi$ is used to collect statistics and $jtopas - vj$ is used in on-demand test suite reduction.

Most results for ROD_L are larger than the confidence level, but many results for ROD_G are smaller than the confidence level. We suspect the reason for this to be that the total number of mutants generated by MuJava is relatively small, and the reduced test suites are skewed since ROD_G uses the looser global constraint. Further, ROD_L is better than ROD_G at controlling the loss in fault-detection capability. This observation is consistent with that seen in Table IV, but the difference between ROD_L and ROD_G in controlling the loss in fault-detection capability in this study is larger than that seen in Study I. For the on-demand test suite reduction approach (either ROD_L or ROD_G) with any specified confidence level, the results of $jt1 - jt2$ are slightly larger than those of $jt0 - jt2$. That is, on-demand test suite reduction is more effective for a program when its most recent previous version is used in statistics collection.

Figure 3 presents the distributions of test suite size reduction for our approach where actual loss in fault-detection

Table VI
PCT. OF RESULTS SATISFYING ENGINEERS' DEMAND, STUDY III

ROD_L								
c	l	**pt**	**pt2**	**rep**	**sch**	**sch2**	**tca**	**tot**
90	5	97.70	99.87	99.60	93.20	98.50	**71.20**	98.10
	15	99.90	99.67	99.20	96.80	100.00	**60.20**	98.10
	25	99.90	99.93	99.87	96.30	99.50	**87.80**	98.50
	35	100.00	99.93	99.67	99.70	99.60	92.40	97.90
	50	100.00	100.00	100.00	100.00	100.00	98.60	100.00
95	5	99.10	100.00	99.93	98.00	99.80	**88.60**	99.60
	15	100.00	99.93	99.87	98.80	100.00	**75.40**	99.20
	25	100.00	99.93	100.00	97.90	99.70	95.20	99.40
	35	100.00	100.00	99.93	99.80	99.90	97.80	99.40
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00
99	5	100.00	100.00	100.00	100.00	100.00	99.20	99.80
	15	100.00	100.00	100.00	100.00	100.00	**98.60**	99.90
	25	100.00	100.00	100.00	99.70	100.00	100.00	99.70
	35	100.00	100.00	100.00	100.00	100.00	99.20	99.70
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00
ROD_G								
90	5	95.30	98.80	98.40	94.10	99.00	**60.40**	94.40
	15	99.70	99.00	97.67	97.40	99.70	**42.20**	95.30
	25	99.60	99.60	99.27	96.50	99.70	**70.40**	95.10
	35	100.00	99.73	98.60	99.30	99.70	**85.20**	95.10
	50	100.00	100.00	100.00	99.70	100.00	98.20	100.00
95	5	97.10	99.60	99.40	98.40	99.90	**82.00**	96.90
	15	99.80	99.60	99.13	99.40	99.70	**63.40**	96.10
	25	99.80	99.93	99.53	99.10	99.80	**81.20**	96.00
	35	100.00	99.73	99.40	99.80	99.70	**87.80**	95.90
	50	100.00	100.00	100.00	99.90	100.00	97.00	100.00
99	5	99.50	100.00	100.00	100.00	100.00	99.20	99.50
	15	100.00	100.00	100.00	99.90	100.00	**94.20**	99.00
	25	99.90	100.00	100.00	99.90	100.00	99.20	**98.70**
	35	100.00	100.00	99.93	100.00	100.00	**98.00**	**98.80**
	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00

capability satisfies engineers' demands, in a manner similar to that of Figure 2. As the confidence level increases, test suite size reduction decreases and thus the size of reduced test suites increases. Moreover, test suite size reduction under ROD_L is usually smaller than or equal to that under ROD_G. That is, the size of the reduced test suites produced by ROD_L is usually larger than or equal to the size of the reduced test suites produced by ROD_G.

C. Study III – Across Programs

Table VI presents the percentages of reduced test suites whose actual loss in fault-detection capability satisfies engineers' demand for the object program listed in the first row when the other object programs are used to collect statistics and these are used in on-demand test suite reduction.

On all programs but $tcas$ and tot_info, results are larger than the confidence level used. Because $tcas$ and tot_info are the smallest programs, we suspect that the technique is less effective when the objects used in statistics collection and test suite reduction differ more widely in scale. Moreover, the percentage of reduced test suites satisfying engineers' demands generated by ROD_L is usually greater than or equal to that of those generated by ROD_G.

Figure 4 presents the distributions of test suite size reduction for our approach where actual loss in fault-detection capability satisfies engineers' demands. As the figure shows, test suite size reduction decreases as confidence level increases, and reduced test suites generated by ROD_G are usually smaller than or the same size as those generated by ROD_L.

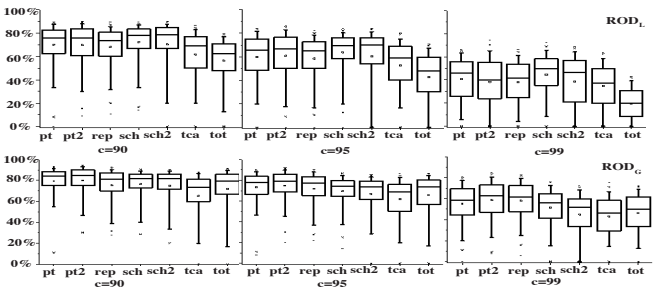

Figure 4. Test suite size reduction in Study III

D. Summary

Generally speaking, the observations derived from our three studies can be summarized as follows.

- On-demand test suite reduction is effective at producing reduced test suites whose actual loss in fault-detection capability is no larger than the specified upper limit for losses in fault-detection capability, when applied to one object, various versions of an object, or various objects.
- The higher the confidence level used in a fault-detection-loss table, the larger the reduced test suites produced by our approach using the table.
- On-demand test suite reduction using global constraints is more effective in reducing the scale of test suites than the approach using local constraints, but less effective at guaranteeing losses in fault-detection capability.

VII. DISCUSSION

Our approach assumes that $V_c(p_j, q_1) \geq V_c(p_j, q_2)$ when $q_1 \leq q_2$ and simplifies the ILP models by setting $V_c(p_j, q)$ to 0 when $p_j \geq q \geq \beta(j)$. As it is impossible to collect statistics on loss for $V_c(i_1, i_2)$ for every possible pair of i_1 and i_2, such a simplification renders the proposed approach more practical. Moreover, as less coverage of a statement may reduce fault-detection capability, the assumption is reasonable.

In our studies, we did not formally compare our approach to existing test suite reduction approaches, because these approaches do not attempt to control losses in fault-detection capability. However, it is interesting to reflect on differences between the approaches, and thus we implemented the HGS algorithm [2] and applied it to the Siemens suite using the faulty programs and test suites utilized in the third scenario of our studies. Because our on-demand approach is meant to select a representative subset whose upper limit loss in fault-detection capability is $l\%$ in at least $c\%$ of its applications, we derive one upper limit l that is slightly smaller than or equal to the upper limit of loss in fault-detection capability for at least $c\%$ (i.e., 90%, 95%, or 99%) of the reduced test

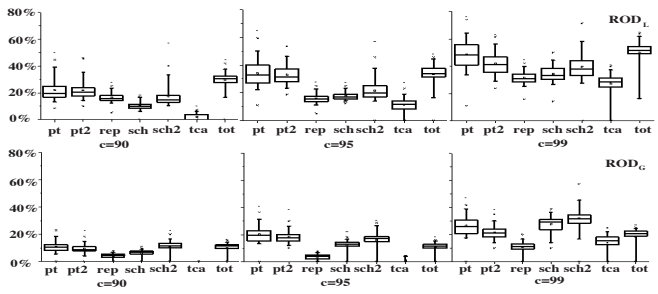

Figure 5. Test suite size reduction comparison: On-demand approach versus HGS algorithm (the horizontal axis depicts the abbreviated object name, and the vertical axis depicts the differences in test size reduction.)

suites created by the HGS algorithm, and subtract the test suite size reduction achieved by the HGS algorithm from that achieved by the on-demand approach at the same confidence level and closest upper limit l.

Figure 5 uses the foregoing data to show the percentages of redundant test cases the on-demand approach selects to achieve losses in fault-detection similar to those exhibited by the HGS algorithm. (The results in this figure may enlarge the percentage of redundant test cases since the upper limit used by the on-demand approach is no larger than that created by the HGS algorithm.) As the boxplots show, to achieve similar losses in fault-detection capability, our on-demand approach based on local constraints (i.e., ROD_L) selects at most 80% more test cases than the HGS algorithm, and the on-demand approach based on global constraints (i.e., ROD_G) selects at most 60% more test cases than the HGS algorithm. To summarize, the on-demand test suite reduction approach achieves the ability to control losses in fault-detection capability by selecting at most 80% more test cases than the HGS algorithm, which cannot control losses in fault-detection capability.

VIII. Related Work

Research on test suite reduction and minimization [2], [3], [4], [6], [17], [18], [19] can be classified into two categories: research on algorithms to reduce the size of a test suite; and research on coverage criteria to improve the fault-detection effectiveness of the reduced test suite.

As it is a NP-complete problem [20], [21] to find the smallest subset of a given set of test cases that maintains a set of testing requirements, research on test suite reduction has focused on heuristics that approximate solutions. Harrold et al. [2] propose an heuristic (HGS) that selects representative test cases that are responsible for test requirements by eliminating redundant and obsolete test cases in the test suite. Chen and Lau [6], [22] propose another heuristic (GRE) that combines a greedy algorithm with what they refer to as an "essential strategy" and a "1-to-1 redundancy strategy". Mansour and EI-Fakin [23] adapt a hybrid genetic algorithm to evolve optimal solutions by reproduction and selection. Black et al. [4] propose two integer linear programming (ILP) models to solve single and multi-objective test suite reduction problems, respectively. As the ILP approach may require exponential time, Chen et al. [24] propose a degraded ILP approach based on the ILP method and traditional heuristic methods. Yoo and Harman [17], [19] formalize the problem of test suite minimization as a multi-objective optimization problem and apply a hybrid algorithm combining the greedy approach and a multi-objective evolutionary algorithm. Note that Fischer et al. [25] also apply the linear programming to regression test selection.

Where research on coverage criteria is concerned, Jones and Harrold [3] consider the modified condition/decision coverage. Other test suite reduction techniques [5], [10] use multiple coverage criteria (e.g., branch coverage and definition-use pair coverage) to increase fault-detection effectiveness. Hsu and Orso [26] propose a framework to support test suite minimization based on any number of criteria using an ILP solver. von Ronne [27] proposes a multi-hit minimization technique, whose test requirements (e.g., statements) are satisfied by more than one test case. Covering test requirements more than once may improve fault-detection effectiveness, but this technique does not include mechanisms for determining how many times test requirements should be covered enough to guarantee certain levels fault-detection effectiveness.

All of the foregoing test suite reduction techniques produce a representative subset of test cases by conceding some degree of loss in fault-detection capability. However, these techniques cannot respond to engineers' demands for placing limits on loss in fault-detection capability.

IX. Conclusion

We have presented a new, on-demand approach to test suite reduction. Given a confidence level $c\%$ and an upper limit of loss in fault-detection capability $l\%$, our technique selects a representative subset of a test suite that satisfies the same test requirements as an initial test suite and concedes at most an $l\%$ loss in fault-detection capability for at least $c\%$ of the instances in which it is applied. Our empirical studies show that the on-demand test suite reduction approach can be effective when it is applied to programs, program versions, and sets of similar programs.

Acknowledgments

This work is supported by the National Basic Research Program of China under Grant No.2009CB320703, the Science Fund for Creative Research Groups of China No. 61121063, and the National Natural Science Foundation of China under Grant No. 60803012. This work was also supported in part by the U.S. Air Force Office of Strategic Research through award FA9550-10-1-0406 and by the U.S. National Science Foundation through award CNS-0958346 to the University of Nebraska - Lincoln.

REFERENCES

[1] H. K. N. Leung and L. White, "Insight into regression testing," in *International Conference on Software Maintenance*, October 1989, pp. 60–69.

[2] 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*, vol. 2, no. 3, pp. 270–285, July 1993.

[3] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 195–209, March 2003.

[4] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *26th International Conference on Software Engineering*, May 2004, pp. 106–115.

[5] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering*, pp. 108–123, 2007.

[6] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction," *Information and Software Technology*, no. 40, pp. 347–354, 1998.

[7] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Journal of Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[8] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *14th International Conference on Software Maintenance*, 1998, pp. 34–43.

[9] W. Wong, J. Horgan, S. London, and A. Mathur, "Effect of test set minimization on fault detection effectiveness," *Software Practice and Experience*, vol. 28, no. 4, pp. 347–369, 1998.

[10] J.-W. Lin and C.-Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques," *Information and Software Technology*, no. 51, pp. 679–690, 2009.

[11] H. Zhong, L. Zhang, and H. Mei, "An experimental study of four typical test suite reduction techniques," *Information and Software Technology*, vol. 50, no. 6, pp. 534–546, 2008.

[12] L. C. B. James H. Andrews and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *27th International Conference on Software Engineering*, 2005, pp. 402–411.

[13] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[14] M.E.Delamaro and J. C. Maldonado, "Proteum - A tool for the assessment of test adequacy for c programs," in *International Conference on Performability in Computing Systems*, 1996, pp. 79–95.

[15] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava:An automated class mutation system," *Journal of Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[16] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and controlflow- based test adequacy criteria," in *16th International Conference on Software Engineering*, 1994, pp. 191–200.

[17] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *ACM International Symposium on Software Testing and Analysis*, 2007, pp. 140–150.

[18] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of junit test-suite reduction," in *22nd International Symposium on Software Reliability Engineering*, 2011.

[19] S. Yoo and M. Harman, "Using hybrid algorithm for pareto efficient multi-objective test suite minimisation," *Journal of Systems and Software*, vol. 83, no. 4, pp. 689–701, 2010.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithm*. MIT Press, Cambridge, MA, 2001.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, New York, 1979.

[22] T. Y. Chen and M. Lau, "Dividing strategies for the optimization of a test suite," *Information Processing Letters*, vol. 60, no. 3, pp. 135–141, 1996.

[23] N. Mansour and K. Ei-Fakih, "Simulated annealing and genetic algorithms for optimal regression testing," *Jounal of Software Maintenance*, vol. 11, no. 1, pp. 19–34, 1999.

[24] Z. Y. Chen, X. F. Zhang, and B. W. Xu, "A degraded ILP approach for test suite reduction," in *20th International Conference on Software Engineering and Knowledge Engineering*, 2008, pp. 494–499.

[25] K. Fischer, "A test case selection method for the validation of software maintenance modification," in *International Computer Software and Applications Conference*, 1977, pp. 421–426.

[26] H.-Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *31st International Conference on Software Engineering and Knowledge Engineering*, 2009, pp. 419–429.

[27] J. von Ronne, *Test Suite Minimization An Empirical Study*. Master's Thesis, Oregon State University, 1999.