Matching Test Cases for Effective Fault Localization

George K. Baah†, Andy Podgurski‡, and Mary Jean Harrold†
†*College of Computing, Georgia Institute of Technology*
Atlanta, GA 30332
‡*Electrical Engineering and Computer Science Department, Case Western Reserve University*
Cleveland, OH 44106
Email:{baah,harrold}@cc.gatech.edu, podgurski@case.edu

***Abstract*—Finding the cause of a program's failure from a causal-analysis perspective requires, for each statement, tests that cover the statement and tests that do not cover the statement. However, in practice the composition of test suites can be detrimental to effective fault localization for two reasons: (1) lack-of-balance, which occurs if the coverage characteristics of tests that cover a statement differ from tests that do not cover the statement, and (2) lack-of-overlap, which occurs if test cases that reach the control-dependence predecessor of a statement cover or do not cover the statement. This paper addresses these two problems. First, the paper presents empirical results that show that, for effective fault localization, the composition of test suites should exhibit balance and overlap. Second, the paper presents new techniques to overcome these problems—matching to address lack-of-balance and causal-effect imputation to overcome lack-of-overlap—and presents empirical evidence that these techniques increase the effectiveness of fault localization.**

***Keywords*-debugging, causal analysis, program analysis, fault localization;**

I. Introduction

Debugging is a tedious and time-consuming process. One of the most difficult tasks in debugging is finding the faults (defects) that caused observed program failures—this task is known as *fault localization*. Over the past decade, a number of statistical fault-localization techniques (e.g. [6], [7], [15], [16], [17], [24]) have been developed to automate fault localization and thus, ease the burden on the developer during debugging. These techniques compute measures of the suspiciousness of program statements (or other program entities, such as branches or methods), using information from execution profiles of passing and failing tests. Statements with high suspiciousness are considered more likely to be the cause of the failures and thus, they are examined first.

One problem with most statistical fault-localization techniques is that the suspiciousness scores they compute may be heavily influenced by *confounding bias* (or *confounding* for short) [18], [20], which occurs, for example, if a faulty statement causes a non-faulty statement to be executed whenever the faulty statement triggers program failures. Most statistical fault-localization techniques are vulnerable to confounding bias because they compute measures of statistical association between the execution of individual statements and the occurrence of program failures. (Linear correlation is a familiar kind of statistical association.) It is well known that *association does not imply causation* [13]. To better estimate the actual causal effect of a statement on the occurrence of failures, it is necessary to reduce the effects of confounding bias.

In previous work [4], we introduced a novel causal model for statistical fault localization, that is grounded in the theory of *causal inference from observational data*, developed by researchers in such diverse fields as economics, epidemiology, social science, statistics, and computer science [12], [18], [20]. This causal model, which we call the *CFL1 model*, lets the causal effect of a statement s on program failures be estimated more accurately than with purely associational measures, because it accounts for the possible confounding influences of other statements on s. In that work, we analyzed the relationships of several associational fault-localization techniques to the CFL1 model and to each other, and presented analytical and empirical evidence that the CFL1 model is more effective for fault localization. Nevertheless, the CFL1 model is just a first step toward making full use of causal-inference theory and methodology in fault localization.

One limitation of the CFL1 model is that it does not fully account for the composition of the test suite used to compute causal effects. This composition can render some causal-effect estimates computed with the model unreliable. One problem occurs when a statement s is either covered by all tests or covered by no tests. In either case, there is no valid basis for estimating its causal effect on failures. Another problem occurs when, the set of tests that cover s and the set of tests that do not cover s differ substantially with respect to other relevant factors (e.g., the control flow paths they induce). In this case, an estimate of the causal effect of s on the occurrence of failures is vulnerable to confounding.

To reduce possible confounding bias due to control flow, the CFL1 model for a statement s includes a binary variable indicating whether a given test covers the control

dependence predecessor[1] of s, which we denote *CDP*(s). However, if there are no tests that cover *CDP*(s) but don't cover s, confounding is likely to bias the causal-effect estimate for s. This is a special case of a problem that occurs in causal inference when there is complete *lack-of-overlap* [11] between the group of subjects that receives some treatment or intervention (the *treatment group*) and the group that received no treatment (the *control group*), with respect to the range of a relevant pre-treatment variable (a confounding covariate[2] of the treatment variable). Here, the "treatment" group is the set of tests that cover *CDP*(s) and the "control" group is the set of tests that do not cover it.

Even if some tests cover *CDP*(s) and other tests do not, confounding may occur because the two sets of tests differ in other important respects. For example, the tests that cover *CDP*(s) may have no branches in common with those that do not cover it. This problem is a special case of another general causal-inference problem, which is *lack-of-balance* between the treatment and the control groups with respect to the probability distribution of a confounding covariate.

Lack-of-overlap and lack-of-balance are mitigated in *randomized controlled experiments* by the random assignment of subjects to the treatment group and to the control group. Unfortunately, test suites are seldom, if ever, obtained by randomly selecting, for each statement s, a set of tests that cover s and a set of tests that do not cover s. (Moreover, doing so is likely to be challenging.)

This paper addresses lack-of-balance and lack-of-overlap in the context of statistical fault localization. First, the paper presents empirical results indicating that it is important for a set of tests used to estimate the causal effect of a statement s on the occurrence of failures to exhibit both overlap and balance with respect to coverage of *CDP*(s). Second, the paper presents our new techniques for addressing lack-of-balance and lack-of-overlap for fault localization. To mitigate lack-of-balance that occurs because tests that miss s also miss *CDP*(s), we employ the classical causal-inference technique *matching* [11], [18]. Matching involves pairing each representative of the treatment group with a representative of the control group, which is similar in terms of its covariate value(s). Our technique pairs each test that covers s with a test that reaches *CDP*(s) but does not branch to s. To mitigate lack-of-overlap in situations where matching is not applicable (i.e., there is a complete lack-of-overlap), we introduce a causal-effect-imputation technique that imputes the causal effects of statements without lack-of-overlap problems to statements with complete lack-of-overlap problems. The paper presents evidence that matching and causal-effect-imputation improve the effectiveness of fault localization over existing techniques.

[1]The *control dependence predecessor* of a statement s is the statement on which s is control dependent.

[2]A *confounding covariate* or *confounder* is a variable that may affect the treatment assignment or the outcome and thus, cause confounding bias. [11]

The main contributions of this paper are as follows:

- Empirical evidence that shows that it is important that a test suite used to estimate the causal effect of a statement s on the occurrence of failures exhibit balance and overlap, with respect to coverage of *CDP*(s).
- Novel techniques that use matching to address lack-of-overlap and causal-effect imputation to address lack-of-balance in a test suite.
- Empirical results that show that matching and causal-effect imputation improve the effectiveness of fault localization over our previous causal-effect estimator by 22%.

II. Background

This section provides background required to understand causal inference, and defines terminology that we use in the rest of the paper.

A. Potential Outcome Model

Our technique is based on the potential outcome model [18], Pearl's Structural Causal Model [20], and program dependence analysis [10]. We will explain our causal model in the context of the potential outcome model, which is the model used to estimate causal effects in areas such as the social sciences [11], [18].

Suppose that a researcher wants to investigate the effect of a new or existing treatment on a set of units or subjects. The potential outcome model associates with each unit a treatment variable representing two states: *treatment applied* and *treatment not applied*. Suppose T is the treatment variable. T is in the treatment state if $T = 1$ (i.e., the treatment has been applied) and T is in the control state if $T = 0$ (i.e., the treatment has not been applied). The units that are treated (i.e., units with $T = 1$) are referred to as the *treatment group* and the units that are not treated (i.e., units with $T = 0$) are referred to as the *control group*.

Corresponding to an observable outcome variable Y, the potential outcome model defines potential-outcome random variables Y^1 and Y^0, which represent the outcome for a unit in the treatment state and in the control state, respectively. After treatments are administered, only one of these two variables is observable for a given subject; the unobserved variable is referred to as the *counterfactural* and it can be modeled statistically.

To estimate the effect of the treatment, the researcher may be able to conduct an experiment in which the units are *randomly* assigned to the treatment group and to the control group. In this case, the causal effect of the treatment on the outcome is

$$\tau = E[Y^1] - E[Y^0] \tag{1}$$

where $E[\cdot]$ denotes the expectation operator. This quantity can be estimated accurately, given sufficient units, by subtracting the average outcomes for the treatment group

and the control group, respectively. By randomly assigning units to the treatment and control groups, the two groups become approximately balanced with respect to the values of variables (both observed and unobserved) that might otherwise confound the effect of the treatment. Randomization thus removes any *confounding bias* that could affect the causal-effect estimate. Variables that confound the effects of treatments are referred to as *confounders*. For example, the age and gender of patients may be confounders for the effects of a medical treatment.

Although a randomized experiment is the ideal approach to estimating causal effects, in practice, researchers often must rely on data from observational studies (e.g., surveys or retrospective studies). In attempting to estimate the causal effect of a treatment with observational data, there is a greater chance of confounding bias than with a randomized experiment. To reduce confounding bias in observational studies, a statistical model is often used to "condition on" or "control for" observed confounders. Controlling for observed confounders means that relevant characteristics of the units are taken into account when computing causal effects. In practice, a regression model is most often used to estimate the causal effect of the treatment. The observed confounders in the observational data are incorporated into the regression model.

B. Fault Localization using Causal Inference

Our basic model [4] for the causal effect of a particular statement upon program failures is a linear regression model of the form

$$Y = \alpha + \tau T + \beta X + \varepsilon \tag{2}$$

In this model Y is the outcome variable, T is the treatment variable, X is a confounder, τ and β are coefficients, α is the intercept, and ε is an error term. The coefficient τ of the treatment variable T is the causal effect of the treatment, and the fitted value $\hat{\tau}$ for τ is the estimated treatment effect. We now describe the specific nature of each of these variables.

When applying our causal model to a statement s, *coverage* (execution) of s by a test is the "treatment". We wish to estimate its causal effect on the outcome *program failure*. A test that does not cover s is a member of the control group (for s). The treatment variable T is a binary indicator that is 1 for a test just in case it covers s.

If we could randomly generate a mixture of passing and failing tests belonging to the treatment group (tests that cover s) and the control group (tests that don't cover s), we could replace the two expected values in Equation 1 with the proportions of failing tests in the treatment group and the control group, respectively, in order to estimate the causal effect of s on failures. However, *we assume that we must make due with an existing test suite that was not generated in this way.* Therefore it is necessary to control for confounding, if possible.

```
    void upgrade_process_prio(prio, ratio){
                . . .
201     if (prio >= MAXPRIO)
202        return;
                . . .
207     if (count > 1) /* off by one */ {
208        n = (int) (count*ratio + 1);
209        proc = find_nth(src_queue, n);
210        if (proc) {
211           src_queue = del_ele(src_queue, proc);
                . . .
           }
216     }
217 }
```

Figure 1. A code snippet from the Siemens program *Schedule* version 4

The confounding variable X in our causal model is a binary coverage indicator for the forward control dependence predecessor of a statement. Informally, statement s' is the *forward control dependence predecessor* of statement s, denoted *CDP*(s), if s is control dependent on s' and s does not dominate s'. Statement s is control dependent on s' if s' represents a branch predicate whose execution directly controls whether s is executed. The variable X in our model takes on the value 1 for a test just in case it covers *CDP*(s). The justification for using the foward control dependence predecessor, which is presented in our previous paper [4], is based on Pearl's Structural Causal Model and his "Back-Door Theorem" [20].

To illustrate the use of our causal model, we shall use the example code snippet in Figure 1. Figure 1 is a faulty function in version 4 of the Siemens-suite [9] program *Schedule*. *Schedule* is a priority scheduler program. The program sometimes fails when the fault at line 207 is executed. The program has a total of 2650 tests out of which 294 fail.

Table I shows a summary of the execution data gathered for statement 207. The last three columns of each row contain the values of binary variables T, X, and Y for which the value 1 indicates, respectively, that statement 207 was covered, that its control dependence predecessor was covered, and that the program failed. For each configuration (row) of T, X, and Y values in the table, the leftmost column indicates the number of tests that induced that configuration. For example, the first row of Table I means that, out of 2650 tests, 294 failing tests covered both statement 207 and its control dependence predecessor. Table I shows that for statement 207, there were 1775 units in the treatment group and 875 units in the control group. Fitting the regression model in Equation (2) to the data in Table I, the causal effect of statement 207 on failures is the coefficient of the treatment variable, which is 0.17.

Table I
SUMMARY OF EXECUTION DATA GATHERED FOR STATEMENT 207

Number of Tests	Statement Coverage (T)	CDP (X)	Outcome (Y)
294	1	1	1
1481	1	1	0
40	0	1	0
835	0	0	0

Table II
SUMMARY OF EXECUTION DATA GATHERED FOR STATEMENT 211

Number of Tests	Statement Coverage (T)	CDP (X)	Outcome (Y)
193	1	1	1
729	1	1	0
1	0	1	1
14	0	1	0
1613	0	0	0

III. AUGMENTING OUR CAUSAL MODEL WITH MATCHING AND IMPUTATION

In our previous work [4], we presented empirical evidence that the causal model represented by Equation 2 improves the effectiveness of fault localization, when the estimated treatment effect $\hat{\tau}$ for a statement is used as its suspiciousness score. However, for some statements the causal effect estimates can be unreliable, because the model does not take into account the test-suite's composition. As mentioned in the Introduction, a (complete) lack-of-overlap problem occurs for a statement s if there are no tests that cover *CDP*(s) but don't cover s. A lack-of-balance problem occurs for s if the control flow paths induced by the tests that cover s (the treatment group) differ too much from those induced by the tests that don't cover s (the control group). In this section we present our approach to computing more reliable causal estimates by addressing the lack-of-balance and lack-of-overlap problems.

A. Matching Test Cases to Overcome Lack-of-Balance

Our technique for addressing the lack-of-balance problem relies upon the classical causal inference technique called matching. Before presenting our technique, we first illustrate the lack-of-balance problem, using the example code snippet in Figure 1. Tables I and II summarize execution data gathered for statements 207 and 211, respectively, of Figure 1. There is a lack of balance problem with the tests for each of these statements. There are 1775 tests that cover statement 207 (the treatment group) and 875 tests that don't cover it (the control group). Each of the tests that covers statement 207 also covers its CDP. However, only 40 of the tests that don't cover statement 207 do cover its CDP. The control flow paths induced by the remaining 835 tests branch away from statement 207 before reaching its CDP. Similarly, only 15 of the 1628 tests that don't cover statement 211 cover its CDP.

Matching [18] is a technique that was developed to bring some of the benefits of randomized experiments to observational studies. The purpose of matching is to reorganize the data in such a way that relative balance between the treatment group and the control group is achieved with respect to observed confounders. Different kinds of matching techniques have been developed, including exact matching, nearest-neighbor matching, and propensity-score matching. We use a form of *exact matching* in this paper.

To address the lack-of-balance problem for a statement s, our technique excludes tests that do not cover s and also do not cover *CDP*(s) from the set of tests used to fit the causal inference model. That is, the only tests from the control group that are used in fitting the model are those that cover *CDP*(s). This ensures that the treatment group and the modified control group are balanced with respect to the value of the coverage indicator for *CDP*(s), which will be 1 for both groups. Thus the control flow paths of the tests used to fit the model all reach *CDP*(s) before either branching to s or branching away from s. For example, matching on the CDP of statement 211 of *Schedule* means that 1613 tests will be excluded from the set used to fit the causal model for statement 211.

To illustrate the importance of matching on the CDP, we consider estimating the causal effect of statement 211 on failures by fitting Equation (2) to the matched data from Table II (i.e., with the 1613 tests discarded). The causal effect estimate for statement 211 is 0.14. If we do not take into account the CDP of statement 211, so Equation (3) is fit without discarding any test cases, the (biased) causal-effect estimate (δ) for statement 211 becomes 0.21.

$$Y = \alpha + \delta T + \varepsilon \tag{3}$$

That is, if we do not include a coverage indicator for its CDP in our model, statement 211 will be ranked higher than the faulty statement 207, which has a causal-effect estimate of 0.17. However, by matching on the CDP our technique is able to more accurately estimate the causal effect of statement 211 on failure.

B. Addressing Lack-of-Overlap with Causal Effect Imputation

Before we present our approach to the lack-of-overlap problem, we first illustrate the problem using Table I. The coverage data in Table I can be divided into two subgroups: 1815 units for which the coverage indicator X for the CDP of statement 207 has the value of one and 835 units for which X has the value zero. The 1815 units with $X = 1$ can be divided into a treatment group (1775 units) and a control group (40 units). However, the subgroup with $X = 0$ has 835 units in the control group but no treatment units. Therefore there is a lack-of-overlap in the subgroup with

Algorithm 1 : Pseudocode for Causal-Effect Imputation

```
1 impute_causal_effect(Statement:S){
2    effect = -10;
3    matched_data = get_matched_data(S);
4    if(lack-of-overlap(matched_data) == FALSE){
5         if(S == FUNCTION_ENTRY){
6             fit Equation (3) to matched_data;
7             effect = δ;
8         }else{
9             fit Equation (2) to matched_data;
10            effect = τ;
11        }
12        return effect;
13    }
14    effect = impute_causal_effect(CDP(S));
15    return effect;
16 }
```

units that have $X = 0$. Statements 207 and 211 are said to exhibit a partial lack-of-overlap and the problem is resolved using matching.

However, there are cases where after matching (i.e., coverage data has been balanced) a statement exhibits a complete lack-of-overlap. That is, the tests that reach the CDP of a statement only cover the statement (treatment group) or do not cover the statement (control group). Our technique addresses a statement s with a complete lack-of-overlap problem by imputing to s the causal effect of an *ancestor* of s that does not have a lack-of-overlap problem. We call this *causal-effect imputation*. Our technique uses a dynamic form of the *control dependence graph (CDG)* [10] during imputation. Informally, the CDG of a program is a graph whose vertices correspond to statements and whose edges correspond to control dependences.[3]

Algorithm 1 shows our causal-effect imputation algorithm. The algorithm takes as input a statement S and returns the imputed causal effect. To impute causal effects, the algorithm first retrieves the matched data for S at line 3. At line 4, the algorithm checks whether the matched data has a lack-of-overlap problem. If so, the algorithm is called recursively with *CDP*(S) as the argument. The algorithm terminates if an ancestor of S in the CDG is found that does not have a lack-of-overlap problem or if the FUNCTION_ENTRY vertex in the CDG is reached. The FUNCTION_ENTRY vertex is used to represent entry to a function.

IV. Empirical Studies

To assess the importance of balance and overlap among the tests used to estimate a statement's causal effect on program failures, and to evaluate the effectiveness of our techniques for handling these properties and improving fault localization, we implemented our technique, and performed a number of empirical studies using several subject programs. This section overviews the subject programs we used for our studies, discusses our implementation, describes the effectiveness metric we used for the evaluation, and presents the studies.

A. Subject Programs

We used seven programs from the Unix suite (Cal, Col, Comm, Look, Spline, Tr, and Uniq), the seven programs in the Siemens suite (Print-tokens, Print-tokens2, Replace, Schedule, Schedule2, Tcas, and Tot-info), Sed, and Space as subject programs for our studies.[4]

Table III shows the characteristics of the subjects. For each subject, the first (Program), second (#Vers / uVers), third (#LOC), fourth (#Tests), fifth (#Nodes), sixth (CLOO%), and seventh (Description) columns show the name of the program, total number of versions (Vers) and number of versions used (uVers), the number of lines of source code, the number of tests, the average number of vertices in the dynamic control dependence graph, the average percentage of vertices with complete lack-of-overlap problems, and a description of the program, respectively. The Unix suite has a total of 117 program versions, of which we used 113. The Siemens suite has a total of 132 program versions, of which we used 128. For the Sed and Space programs we used a total of 40 versions. Overall we omitted 16 faulty versions because there were no syntactic differences between the C files of the correct version and the faulty version of the program or because none of the tests failed when executed on the faulty version of the program. Thus, we used 16 programs with a total of 281 faulty versions.

B. Implementation

We used the CIL framework [19], which supports the analysis of ANSI C programs, to analyze the subject programs. We implemented algorithms to instrument the programs and extract dynamic control-flow graphs and control-dependence graphs. (These graphs represent only statements and dependences that were actually executed.) We implemented the algorithms in the *Objective Caml* language, because it is required to interface with CIL. We implemented our causal-inference algorithms and a test-selection algorithm in *R* [21], which is a statistical computation system with its own language and runtime environment. The test-selection algorithm was used to select tests that reach each statement's control dependence predecessor.

Our implementation instruments each program version so that at runtime, it will gather the coverage data

[3]Informally, a vertex u is control-dependent on vertex v if in the control flow graph (CFG) of the program v has two outgoing edges e_1 and e_2 and the execution of e_1 causes u to be executed and the execution e_2 causes u not to be executed. The CFG of a program is a graph whose vertices correspond to statements in a program and edges corresponds flow of control in the program.

[4]We obtained the Unix suite from Eric Wong of University of Texas at Dallas and the Space and Sed programs from the Software-artifact Infrastructure Repository [9].

Table III
SUBJECTS USED FOR EMPIRICAL STUDIES.

Program	#Vers / uVers	#LOC	#Tests	#Nodes	CLOO (%)	Description
Cal	20 / 19	202	162	131.5	60.3	calendar printer
Col	30 / 29	102	156	240.0	66.1	filter-line reverser
Comm	12 / 10	167	186	116.1	37.3	file comparer
Look	14 / 14	170	193	140.9	37.8	word finder
Spline	13 / 13	338	700	247.0	43.6	curve interpolator
Tr	11 / 11	137	870	150.2	60.6	character translator
Uniq	17 / 17	143	431	131.3	38.2	duplicate line remover
Print-tokens	7 / 5	472	4130	423.6	35.6	lexical analyzer
Print-tokens2	10 / 10	399	4115	340.5	44.8	lexical analyzer
Replace	32 / 31	512	5542	415.9	38.6	pattern replacement
Schedule	9 / 9	292	2710	216.3	43.3	priority scheduler
Schedule2	10 / 9	301	2650	225.9	37.2	priority scheduler
Tcas	41 / 41	141	1608	136.7	21.1	altitude separation
Tot-info	23 / 23	440	1052	249.0	48.9	information measure
Sed	10 / 10	14K	363	4151.0	58.8	stream editing utility
Space	38 / 30	6K	157	3851.9	51.1	ADL interpreter

needed to construct its dynamic control-flow graph and control-dependence graph. Our implementation uses the dynamic control-flow graph to compute the dynamic control-dependence graph and then extracts the control-dependence predecessor of each statement from the control-dependence graph. Our implementation also computes a statement-coverage matrix from the coverage data gathered for each version. We also computed fault matrices that indicate, for each faulty version, which tests pass and which tests fail. We performed our studies on Mac OS X version 10.5.

C. Effectiveness Metric

To compare the effectiveness of fault-localization techniques that are based on a measure of statement suspiciousness, we employed a metric that has been used in many previous studies [3], [7], [15], [22]. This metric, which we shall denote by *Cost*, characterizes the cost of applying a particular suspiciousness measure to a faulty program as the percentage of statements that a developer must examine before encountering the first faulty statement, assuming the statements are examined in non-increasing order of their suspiciousness scores. If there are ties in the suspiciousness scores of statements, then we assume that the developer has to examine all the tied statements. For example, if there are five statements in the program and they have the same suspiciousness scores then the *Cost* is 100% because all the statements may be examined to find the fault.

To compare the effectiveness of two fault-localization techniques A and B with respect to a particular program version P_i, we use the *Cost* of applying one of them, say B, to P_i as a baseline and subtract the *Cost* of applying A to P_i. A positive result means that A performed better than B on P_i and a negative result means B performed better than A. The difference corresponds to the magnitude of improvement. For example, if the *Cost* of A is 30% and the *Cost* of B is 40%, then the improvement of A over B is 10%, which means that developers would examine 10% fewer statements if they used A.

We display the results of applying A and B to all subject program versions using a graph like the one shown in Figure 2. There is a vertical bar for each program version for which A and B have different *Cost* values. The length of the bar corresponds to the difference in *Cost* values, which is the magnitude of improvement. The horizontal axis represents the cost of the baseline technique, say B. Bars above the horizontal axis correspond to versions for which A performed better than B and bars below the horizontal axis represent versions for A performed worse than B.

D. Study 1: Lack of Overlap

The goal of this study is to determine the percentage of vertices in the subject program's control dependence graphs that had a complete lack-of-overlap problem as described in Section III. To do this we gathered the coverage data for each vertex and used our technique to balance the coverage data and also to determine whether a vertex had a lack-of-overlap problem.

Table III shows the results of the study in the sixth column (CLOO). As the table shows, a considerable number of vertices had a complete lack-of-overlap problem after the coverage data had been balanced. This implies that causal-effect estimates cannot be computed for such statements. For example, the Replace versions have on average of 415.9 vertices in their control dependence graphs, of which 38.6% had a lack-of-overlap problem (even though Replace has 5542 tests). The Print-tokens versions have on average 423.6 vertices, of which 35.6% had a complete lack-of-overlap problem.

The results show that having a large number of tests does not necessarily imply that a test suite is suitable for causal analysis.

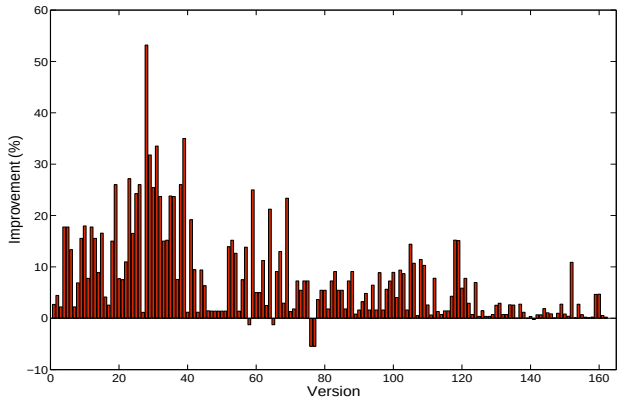

Figure 2. Comparison of *Causal-Effect-MI* with *Biased-Effect*.

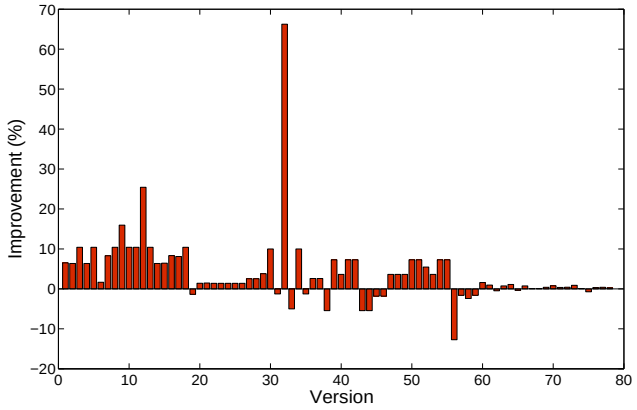

Figure 3. Comparison of *Causal-Effect-MI* with *Causal-Effect-1*.

E. Comparison Studies

The remaining six studies (Studies 2-7) compare our causal-inference techniques to each other and to alternative techniques, with respect to effectiveness. They address the importance of using matching on *CDPs* to address lack-of-balance and of using causal-effect imputation to address lack-of-overlap, as described in in Section III.

Study 2: The goal of this study is to assess the importance of matching tests to be used to estimate the failure-causing effect of a statement s, based on coverage of the control-dependence predecessor of s. To do this, for each statement s, we used only tests that covered *CDP*(s) to fit the causal-inference model shown in Equation 2. If this set of tests did not include both tests that cover s and tests that did not cover s, we used causal-effect imputation. We denote this approach by *Causal-Effect-MI*. We compared it to the naive causal-effect estimator shown in Equation 3, which does not involve the CDP and employs all of the tests in the test suite. We denote the latter estimator by *Biased-Effect*.

Figure 2 shows the results of the study. We used the *Biased-Effect* results as a baseline and subtracted the *Cost* of *Causal-Effect-MI* from the *Cost* of *Biased-Effect*. Figure 2 shows that *Causal-Effect-MI* performed better than *Biased-Effect* on 157 faulty versions but performed worse on 5 versions. The two approaches performed equally well for 119 versions. This study indicates that using causal inference with matching of tests based on coverage of the control-dependence predecessor, and using causal-effect imputation when matching is not possible, is more effective at localizing faults than a naive causal effect estimator used without matching.

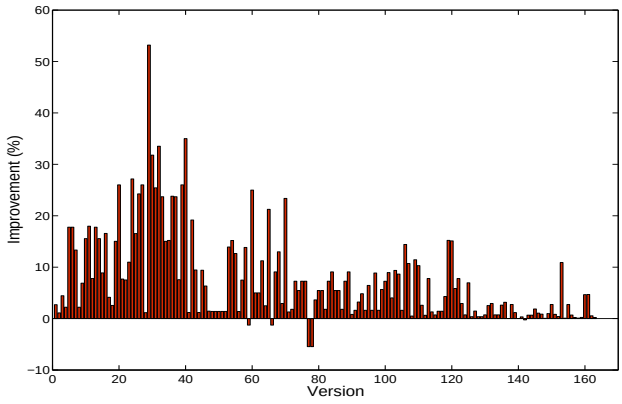

Figure 4. Comparison of *Causal-Effect-MI* with *Tarantula metric*.

Study 3: The goal of this study is to compare the effectiveness of *Causal-Effect-MI* to that of the causal-inference model of Equation 2 (which includes a coverage indicator for the CDP) used without either matching or causal-effect imputation. We denote the latter approach, which we used in our previous work [4], by *Causal-Effect-1*.

Figure 3 shows the results of the study, with *Causal-Effect-1* as the baseline. Figure 3 shows that *Causal-Effect-MI* performed better than *Causal-Effect-1* on 62 versions and performed worse on 16 versions. The two techniques performed identically on 203 versions. The reason for the better performance of *Causal-Effect-MI* is that *Causal-Effect-1* does not address the lack-of-overlap and lack-of-balance problems, whereas *Causal-Effect-MI* does so through matching.

Study 4: The goal of this study is to compare the fault-localization effectiveness of *Causal-Effect-MI* to that of the TARANTULA metric [15]. The *Cost* values for TARANTULA formed the baseline.

Figure 4 shows that *Causal-Effect-MI* performed better than the TARANTULA metric on 158 versions, performed worse on 5 versions, and performed identically on 78 versions. The results for Tarantula are very similar to those for *Biased-Effect*, although *Biased-Effect* uses data from both the treatment and control units (tests) whereas TARANTULA uses data only from the treatment units.

Study 5: The goal of this study is to compare the fault-localization effectiveness of *Causal-Effect-MI* to that of the Ochiai metric [1]. In our previous work [4], *Causal-Effect-*

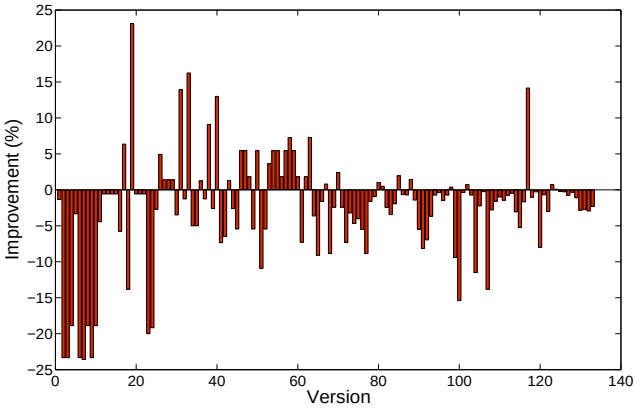

Figure 5. Comparison of *Causal-Effect-MI* with *Ochiai metric*.

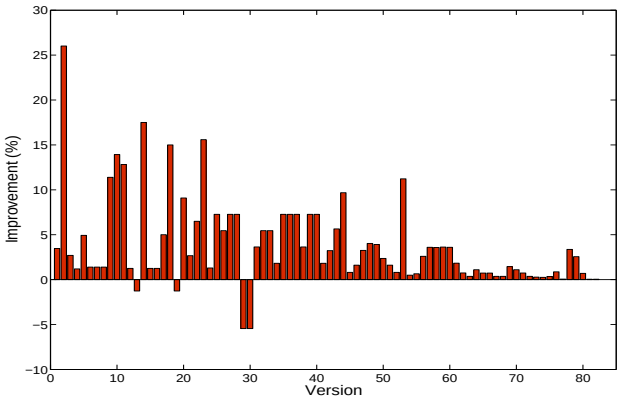

Figure 6. Comparison of *Causal-Ochiai-MI* with *Ochiai metric*.

1 did not perform well compared to the Ochiai metric. We repeated that comparison, using all 281 faulty program versions. *Causal-Effect-1* performed better than the Ochiai metric on 26 versions, performed worse on 118 versions, and performed equally well on 137 versions. As discussed in our previous work [4], these results occur because the Ochiai metric includes a product of a precision term and a recall term, whereas *Causal-Effect-1* is similar to a precision measure used by itself. The recall term in the Ochiai metric assigns more weight to statements that are covered by more failing tests.

Figure 5 shows the results of comparing *Causal-Effect-MI* to the Ochiai metric, with the latter as baseline. The Figure shows that *Causal-Effect-MI* performed better that the Ochiai metric on 37 versions and performed worse 96 versions. The two techniques performed identically on 133 versions. Although, against the Ochiai metric, *Causal-Effect-MI* performed somewhat better than *Causal-Effect-1*, it still did not perform as well as the Ochiai metric.

Study 6: The goal of this study is to evaluate the result of integrating *Causal-Effect-MI* with the Ochiai metric by replacing the latter's precision term, which is an estimate

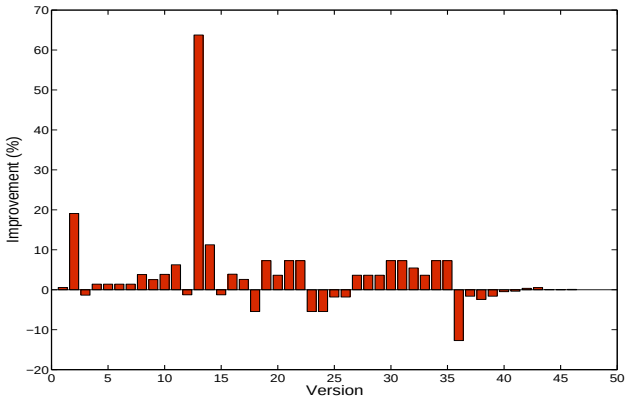

Figure 7. Comparison of *Causal-Ochiai-MI* with *Causal Ochiai*.

of the conditional probability Pr(*failure* | s covered), with *Causal-Effect-MI*. We denote this hybrid by *Causal-Ochiai-MI*.

In our previous work [4], we similarly integrated *Causal-Effect-1* with the Ochiai metric and named the result *Causal-Ochiai*. In this study, we first applied *Causal-Ochiai* to all the 281 faulty versions and compared the results to those of the the basic Ochiai metric. The Causal-Ochiai metric performed better than the Ochiai metric on 71 versions, performed worse on three versions, and performed identically on 207 versions. We then compared *Causal-Ochiai-MI* to the Ochiai metric, using the Ochiai metric as the baseline. Figure 6 shows that *Causal-Ochiai-MI* performed better than the Ochiai metric on 78 versions and performed worse on 4 versions. The two techniques performed identically on 199 versions. The result suggests that there is a substantial benefit to combining a recall measure with a causal-effect estimate obtained using matching and (if necessary) causal-effect-imputation.

Study 7: The goal of this final study is to compare the fault-localization effectiveness of *Causal-Ochiai-MI* to that of *Causal-Ochiai*, with *Causal-Ochiai* as the baseline. Figure 7 shows that, over a total of 281 faulty versions, *Causal-Ochiai-MI* performed better than *Causal-Ochiai* on 32 versions and performed worse on 14 versions. The two techniques performed identically on 235 versions. The results confirm that the use of matching and imputation in *Causal-Ochiai-MI* improve upon *Causal-Ochiai*.

F. Threats to Validity

The empirical studies are subject to three validity threats: internal, external, and construct. Internal validity concerns factors that might influence dependent variables without the knowledge of the researcher. There is the possibility of errors in the implementation of the algorithms we used in our studies. To address potential errors in our implementation, we randomly selected a subset of the subjects and checked

manually the results of the studies. Threats to external validity occur when the results of a study cannot be generalized. In this work such threats are greatly mitigated by the use of established causal inference theory and methodology. However, more experiments on additional subjects will be needed to fully address this threat. Threats to construct validity concern the appropriateness of study measurements, and they pertain to the *Score* metric we used to measure the effectiveness of fault localization. This metric has the advantage of having been used in a number of other fault-localization studies, but it has weaknesses. For example a developer is likely stop examining program statements in non-increasing order of their suspiciousness scores if he or she does not find a fault relatively soon. Alternatively, a developer might use suspiciousness scores to set debugging breakpoints. We intend to evaluate our techniques using other effectiveness measures in the future.

V. Related Work

Many fault-localization techniques use statistical analysis and program coverage to identify those entities that are the most suspicious of being faulty. One category of statistical-based techniques is the associative techniques (e.g., [15], [16], [17]). These techniques implicitly assume that the program entity most correlated or associated with program failure is most likely to be the cause of the failure. However, as we discussed in the paper and demonstrated with empirical studies, association does not imply causality, and suspicious entities are often not related to the fault. Our technique, in contrast, uses the theoretically-grounded causal-inference methodology to find the cause of the program failures. Our empirical studies demonstrate the effectiveness of our causal-inference technique over the associative techniques.

Another category of statistical-based techniques are, like ours, causal techniques. These techniques are based either on experimentation or on observational data. One technique [8] attempts to find the cause of program failure by repeatedly deleting a statement from a program and re-executing the program after the statement has been deleted. Another technique [26] Delta Debugging, attempts to find the cause of program failure by switching program states and re-executing the program. Both of these experimental-causal techniques attempt to locate the faults by performing experiments on programs, but performing experiments on programs is difficult and expensive. For example, Delta Debugging must ensure the consistency of memory changes, both techniques require repeated execution of the program, which can be time consuming, and they require the presence of an oracle to determine the status (success or failure) of a program each time it is re-executed Our approach differs from the experimental-causal approach because it is based on observational data e.g., coverage information) that is often available and that is used for the analysis. Our technique is executed once on tests whose status (passing or failing) is already determined, and thus, does not require repeated execution of the program with different memory states or oracles for each of these executions. Thus, our technique can be more efficient than these experimental-based approaches.

Another category of associative-based fault-localization techniques use slicing (e.g., [23], [27]). to compute the set of statements that potentially affect the values of a given program point (e.g., program output). However, these techniques do not find the cause of the failure. Instead, they compute the set of statements that are potentially correlated with the fault. Additionally, these techniques provide no guidance on how the statements in the slice should be examined. Thus, it is difficult to compare these techniques directly with our new technique. However, we have demonstrated with empirical studies that association does not imply causality, and thus, these techniques cannot be as effective as causal techniques in identifying faults.

A number of papers report the relationship between test-suite composition and fault localization [2], [5], [14], [25]. Some of these techniques generate tests using heuristics for effective fault localization, whereas others reduce the size of an existing test suite while trying to maintain the efficacy of the test suite for effective fault localization. However, the resulting tests might not be appropriate for all statements in the program because the techniques are based on heuristics. In contrast, our technique is theoretically motivated and as such provides guarantees for fault-localization effectiveness.

Finally, this work improves upon our previous work [4] on finding the cause of program failures from observational data. In that work, we did not address the lack-of-balance or lack-of-overlap problems. Our current technique addresses the two problems by using tests that reach the control-dependence predecessor of a statement and causal-effect imputation to significantly improve the accuracy of causal-effect estimation. Thus, this new technique results in improved fault-localization results.

VI. Conclusion and Future Work

In this paper, we have shown empirically the importance of test-suite composition for effective fault localization from a causal perspective. We have presented evidence that for accurately estimating the causal effect of a statement on program failures, it is desirable to employ a set of tests that all reach the control dependence predecessor of the statement; moreover, each branch out of the CDP should be taken by some tests. Such a matched test set avoids the problem lack-of-balance with respect to coverage of the CDP, which could bias the causal estimate. We also presented a causal-effect-imputation technique for addressing lack-of-overlap after matching.

There is clearly a limit as to how accurate causal effect estimates based only on coverage information can be. For example, statements in the same control dependent region will tend to have the same effect estimates. In the future, we

will also investigate combining control and data dependences with variable values in our causal model to further improve the effectiveness of fault localization.

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A. J. van Gemund. On the Accuracy of Spectrum-Based Fault Localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques*, pages 89–98, September 2007.

[2] S. Artzi, J. Dolby, F. Tip, and M. Pistoia. Directed Test Generation for Effective Fault Localization. In *Proceedings of the International Symposium for Software Testing and Analysis*, July 2010.

[3] G. K. Baah, A. Podgurski, and M. J. Harrold. The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis. In *Proceedings of International Symposium for Software Testing and Analysis*, July 2008.

[4] G. K. Baah, A. Podgurski, and M. J. Harrold. Causal Inference for Statistical Fault Localization. In *Proceedings of the International Symposium on Software Testing and Analysis*, July 2010.

[5] B. Baudry, F. Fleurey, and Y. L. Traon. Improving test suites for efficient fault localization. In *Proceedings of International Conference on Software Engineering*, May 2006.

[6] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan. Identifying Bug Signatures Using Discriminative Graph Mining. In *Proceedings of the International Symposium on Software Testing and Analysis*, July 2009.

[7] H. Cleve and A. Zeller. Locating Causes of Program Failures. In *Proceedings of the International Symposium on the Foundations of Software Engineering*, pages 342–351, May 2005.

[8] R. A. DeMillo, H. Pan, and E. H. Spafford. Critical Slicing for Software Fault Localization. In *ISSTA '96: Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis*, pages 121–134, 1996.

[9] H. Do, S. Elbaum, and G. Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[10] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, July 1987.

[11] A. Gelman and J. Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2007.

[12] J. J. Heckman. Microdata, Heterogeneity and the Evaluation of Public Policy. *Nobel Lectures*, Economics 1996-2000:255–322, 2000.

[13] P. W. Holland. Statistics and Causal Inference. *Journal of American Statistical Association*, 81:945–970, 1986.

[14] B. Jiang, Z. Zhang, T. H. Tse, and T. Y. Chen. How Well Do Test Case Prioritization Techniques Support Statistical Fault Localization. In *Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC)*, pages 99–106, 2009.

[15] J. Jones, M. J. Harrold, and J. Stasko. Visualization of Test Information to Assist Fault Localization. In *Proceedings of the International Conference on Software Engineering*, pages 467–477, May 2002.

[16] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable Statistical Bug Isolation. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 15–26, June 2005.

[17] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. SOBER: Statistical Model-based Bug Localization. In *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 286–295, September 2005.

[18] S. L. Morgan and C. Winship. *Counterfactuals and Causal Inference: Methods and Principles of Social Research*. Cambridge University Press, 2007.

[19] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In *Proceedings of the International Conference on Compiler Construction*, pages 213–228, April 2002.

[20] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, San Francisco, CA, USA, 2000.

[21] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[22] M. Renieris and S. Reiss. Fault Localization With Nearest Neighbor Queries. In *International Conference on Automated Software Engineering*, pages 30–39, November 2003.

[23] M. Weiser. Program Slicing. In *Proceedings of the International Conference on Software Engineering*, pages 439–449, March 1981.

[24] W. E. Wong, V. Debroy, and B. Choi. A Family of Code Coverage-Based Heuristics for Effective Fault Localization. *Journal of Systems and Software*, 83(2):188–208, 2010.

[25] Y. Yu, J. A. Jones, and M. J. Harrold. An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization. In *Proceedings of the 30th International Conference on Software Engineering*, pages 201–210, 2008.

[26] A. Zeller. Isolating Cause-Effect Chains from Computer Programs. In *Proceedings ACM SIGSOFT 10th International Symposium on the Foundations of Software Engineering*, November 2002.

[27] X. Zhang, N. Gupta, and R. Gupta. Pruning Dynamic Slices With Confidence. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 169–180, June 2006.