An Empirical Investigation of Program Spectra*

Mary Jean Harrold
harrold@cis.ohio-state.edu
Ohio State U.

Gregg Rothermel
grother@cs.orst.edu
Oregon State U.

Rui Wu
rwu@cis.ohio-state.edu
Ohio State U.

Liu Yi
liuyi@cs.orst.edu
Oregon State U.

Abstract

A variety of expensive software maintenance and testing tasks require a comparison of the behaviors of program versions. Program spectra have recently been proposed as a heuristic for use in performing such comparisons. To assess the potential usefulness of spectra in this context, we conducted an experiment that examined the relationship between program spectra and program behavior, and empirically compared several types of spectra. This paper reports the results of that experiment.

1 Introduction

A variety of software testing and maintenance tasks require us to compare the behaviors of multiple program versions. For example, when we modify a program, we use regression testing to compare the behavior of the modified version to the behavior of its previous version, in the hope of detecting faults caused by the modifications. Similarly, when programs exhibit "regression failures" (behavioral failures that did not occur in preceding versions), we compare the behaviors of versions in the hope of pinpointing the cause of those failures. Tasks such as these constitute a significant percentage of the costs of software testing and maintenance; techniques that reduce these costs are valuable.

Path spectra were recently proposed as a heuristic for understanding the magnitude of the behavioral changes between program versions [10].[1] A *path spectrum* is a distribution of paths derived from an execution of a program using program profiling. A *path-spectra-comparison technique* compares path spectra to gain insight into program behavior. Such a technique may aid in addressing testing and maintenance tasks that require such understanding. For example, constructing expected outputs for programs can be costly; the presence of spectra differences may serve as an indicator of cases in which that construction is unnecessary. Alternatively, spectra comparison [10] may help programmers locate points of divergence in computations, that may guide them in fault localization.

For path spectra to be useful in these contexts, however, they must provide meaningful behavior signatures. An assessment of the potential usefulness of path-spectra comparisons requires an understanding of the correlation between spectra differences and program behavior.

Program behavior can be measured in many ways; however, one measure — important to uses of spectra such as those described above — considers whether particular inputs cause a program to fail. Reference [10] hypothesizes a strong correlation between spectra differences and faults, at least in one direction, stating that given a faulty program and corrected version, one would expect differences between spectra on an input that produces the bug in the original program. One goal of this work is to empirically investigate this claim.

If path spectra prove useful then other spectra such as branch spectra or complete-path spectra may also be useful, and may provide a range of techniques, varying in cost and effectiveness, for examining program behavior. Reference [10] conjectures, however, that edge and node spectra will not be as useful as path spectra for distinguishing program behavior. There is some empirical

*This work was supported in part by a grant from Microsoft Inc., by NSF under NYI Award CCR-9696157 to Ohio State University, ESS Award CCR-9707792 to Ohio State University and Oregon State University, and CAREER Award CCR-9703108 to Oregon State University.

[1] The primary use of spectra investigated in [10] addresses the "Year 2000 problem," and involves comparing spectra from *two runs of the same program* on input data that differs only with respect to date. The intuition is that spectra differences may help programmers locate date-dependent computations. The alternative use of spectra that we investigate here, in which spectra are collected from runs of a program and a slightly different version of the program on the same data, is briefly described in [10], but not pursued in depth. The goal of this work is to empirically investigate this alternative suggestion.

Mnemonic	*Name*	*Description*
BHS	Branch-hit	conditional branches that were executed
BCS	Branch-count	number of times each conditional branch was executed
PHS	Path-hit	path (intraprocedural, loop-free) that was executed
PCS	Path-count	number of times each path (intraprocedural, loop-free)was executed
CPS	Complete-path	complete path that was executed
DHS	Data-dependence-hit	definition-use pairs that were executed
DCS	Data-dependence-count	number of times each definition-use pair was executed
OPS	Output	output that was produced
ETS	Execution-trace	execution trace that was produced

Table 1: A catalog of program spectra.

data [1] to support this conjecture: this data indicates that path profiling data may be superior to edge profiling data for certain applications. Another recent study [4], however, suggests the contrary. These studies have not directly investigated program spectra. A second goal of this work is to perform such an investigation.

2 Program Spectra

A program spectrum characterizes, or provides a signature of, a program's behavior [10]. *Path spectra* use path profiling [1, 3] to track the execution of loop-free intraprocedural paths in a program. Path spectra can track the frequency of a path occurrence, or ignore frequency and track whether or not the path occurred. Spectra can also be constructed based on node or edge profiling data.

In addition to path, node, and edge spectra, many additional signatures of program behavior can be treated as spectra. To obtain a broader empirical view of the relationships among spectra, we consider nine distinct types of spectra. We next describe these spectra (Table 1), and use an example to illustrate them. In the following, let P be a program.

Branch spectra. Branch spectra record the set of conditional branches that are exercised as P executes. We treat the entry to a procedure as a branch; under this treatment, for inputs on which programs terminate, branch spectra are functionally equivalent to edge spectra. If, for each conditional branch in P, the spectrum merely indicates whether or not that branch was exercised, the spectrum is a *branch-hit spectrum* (BHS). If, for each conditional branch in P, the spectrum indicates the number of times that branch was executed, the spectrum is a *branch-count spectrum* (BCS).

Path spectra. Path spectra [10] record the set of loop-free intraprocedural paths that are traversed as P executes. If, for each such loop-free path in P, the spectrum merely indicates whether or not that path was executed, the spectrum is a *path-hit spectrum* (PHS). If, for each loop-free intraprocedural path in P, the spectrum indicates the number of times that path was executed, the spectrum is a *path-count spectrum* (PCS).

Complete-path spectrum. A *complete path spectrum* (CPS) records the complete path that is traversed as P executes.

Data-dependence spectra. Data-dependence spectra record the set of definition-use pairs that are exercised as P executes. If, for each definition-use pair in P, the spectrum merely indicates whether or not that definition-use pair was exercised, the spectrum is a *data-dependence-hit spectrum* (DHS). If, for each definition-use pair in P, the spectrum indicates the number of times that definition-use pair was exercised, the spectrum is a *data-dependence-count spectrum* (DCS).

Output spectrum. An *output spectrum* (OPS) records the output produced by P as it executes.

Execution-trace spectrum. An *execution trace spectrum* (ETS) records the sequence of program statements traversed as P executes.

At first glance, CPS and ETS may appear to be the same, because they both record the complete control flow path through P. They differ, however, in that CPS does not record the actual instructions executed along that path, whereas ETS does.

OPS and ETS are of interest in this context because of their relationship to regression testing. One important regression testing activity is the selection of a subset of the test suite that was originally used to test the program for use in testing the modified program; Reference [11] provides details. In brief, given a program P, test suite T for P, and modified version P', we want to identify the tests in T that reveal faults in P'. For tests whose specified behavior has not changed, these "fault-revealing" tests are exactly the tests that produce different output spectra on P and P'. In general, there is no algorithm to precisely identify these tests,

```
program Sums
1 read i
2 sum = 0
3 while i < 10
4   read j
5   sum = sum + j
6   i = i + 1
  endwhile
7 print sum
end Sums
```

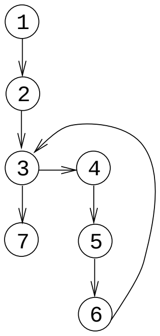

Figure 1: Sums and its control flow graph.

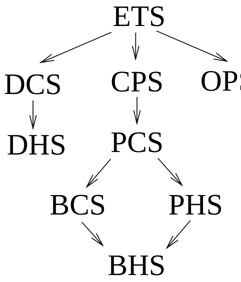

Figure 2: Spectra subsumption hierarchy.

but under certain conditions, the tests that produce different execution trace spectra constitute a conservative (safe) approximation. Several regression test selection techniques [2, 5, 12] exploit this relationship to select safe subsets of T for use in regression testing P'.

To illustrate these spectra, we present program Sums and its control flow graph (Figure 1), and the spectra for Sums on two executions (Table 2): execution 1 uses input 10 and has expected output 0, and execution 2 uses inputs 8, 2, and 4 and has expected output 6. Where applicable, the two types of spectra in each category – hit and count – are shown in columns on the right in the table; where the spectra category does not have these subtypes, "NA" is listed. Consider, for example, the branch spectra for Sums. There are three conditional branches in Sums, and the two executions exercise all of them: the BHS for execution 1 records Y for edges (1,2) and (3,7); the BHS for execution 2 records Y for edges (1,2), (3,4), and (3,7); the BCS for execution 1 records that edges (1,2) and (3,7) were each exercised once; and the BCS for execution 2 records that edges (1,2) and (3,7) were exercised once, and edge (3,4) was exercised twice.

We can analytically compare types of spectra by determining a subsumption relationship that exists between them. Spectra type S_1 *subsumes* spectra type S_2 if and only if, whenever the S_2 spectra for program P, version P', and input i differ, the S_1 spectra for P, P', and i differ. Spectra type S_1 *strictly subsumes* spectra type S_2, denoted $\rightarrow$, if S_1 subsumes S_2, and for some program P, version P', and i, the S_1 spectrum differs but S_2 spectrum does not. Spectra types S_1 and S_2 are *incomparable* if neither $S_1 \rightarrow S_2$ nor $S_2 \rightarrow S_1$. Reps et al. [10] discuss the subsumption relationship that exists between path and edge (and thus branch) spectra; the following theorem establishes the subsumption relationship for all of the spectra we are considering.

Theorem 1: The family of spectra listed in Table 1 is partially ordered by strict subsumption as shown in Figure 2. Furthermore, spectra type S_1 strictly subsumes spectra type S_2 if and only if it is explicitly shown to be so in Figure 2 or follows from the transitivity of the relationship.

Proof:
1. ETS $\rightarrow$ DCS, ETS $\rightarrow$ CPS, ETS $\rightarrow$ OPS .
Given a spectrum $S \in \{\mathrm{DCS}, \mathrm{CPS}, \mathrm{OPS}\}$. Let i be an input that is S-differencing for P and P'. Then, assuming controlled regression testing [11], P' must execute the statement that has changed from P. Thus, i is ETS-differencing, and DCS, CPS, and OPS are subsumed by ETS.
To show that the subsumption is strict, consider a program that has a predicate statement $s : a \leq b$ in P that is changed to $s' : a < b$ in P', and an input i that causes a to be less than b when execution reaches both s and s'. In this case, i is ETS-differencing. However, none of the data-dependencies, the complete path, or the output change for input i, which means that i is not DCS-differencing, CPS-differencing, or OPS-differencing. Thus, equality does not hold, and the subsumption is strict.
2. CPS $\rightarrow$ PCS.
Let i be an input that is PCS-differencing for P and P'. Then there is at least one loop-free path e for P' whose count differs from the count of e in the PCS for P. But this means that there must be a complete path that differs. Thus, i is PCS-differencing and CPS subsumes PCS.
To show that the subsumption is strict, consider the control flow graph of program P, shown on the right in Figure 3. Suppose that, on some input i, the complete path taken through the graph is EabcdfbcdfbceghX. The PCS for P contains paths EabcdfX, EbcdfX, EbcegX, and EhX. Now suppose P' is such that its execution on i produces the complete path EabcdfbcegbcdfhX. Because the complete path is different for P and P' on i, i is CPS-differencing. However, the acyclic paths in P' would be EabcdfX, EbcdfX, EbcegX, and EhX – the same as for P. Thus, equality does not hold, and the subsumption is strict.

		Execution 1 (input is 10)		Execution 2 (input is 8, 2, 4)	
Spectrum	*Profiled entities*	Hit	Count	Hit	Count
Branch	(1,2)	Y	1	Y	1
	(3,4)	N	0	Y	2
	(3,7)	Y	1	Y	1
Path	(1,2,3,7)	Y	1	N	0
	(1,3,7), (1,2,3,4,5,6,7), (1,3,4,5,6,7)	N	0	Y	1
Complete-path	(1,2,3,7)	Y	NA	N	NA
	(1,2,3,(4,5,6,3)^2,7)	N	NA	Y	NA
Data-dependence	(1,(3,7),i), (2,7,sum)	Y	1	N	0
	(1,(3,4),i), (1,6,i), (6,(3,7),i), (2,5,sum), (5,7,sum),(6,6,i), (6,(3,4),i), (5,5,sum)	N	0	Y	1
	(4,5,j)	N	0	Y	2
Output	sum is 0	Y	NA	N	NA
	sum is 6	N	NA	Y	NA
Execution-trace	(S1,S2,S3,S7)	Y	NA	N	NA
	(S1,S2,S3,(S4,S5,S6,S3)^2,S7)	N	NA	Y	NA

Table 2: Spectra for program Sums of Figure 1.

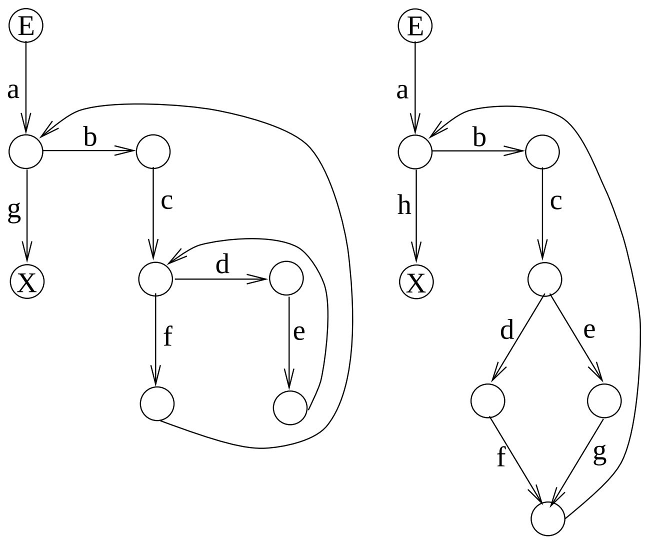

Figure 3: Subgraphs used for proof of Theorem 1.

3. PCS $\rightarrow$ BCS.

Let i be an input that is BCS-differencing for P and P'. Then there is at least one branch e in the BCS for P' whose count differs from the count of e in P's BCS. The paths in PCS are acyclic, so each occurrence of e appears in exactly one path in PCS. Thus, one of the paths containing e has a different count in P and P', and i is PCS-differencing for P and P', and PCS subsumes BCS.

To show that the subsumption is strict, consider the control flow graph on the left in Figure 3. Suppose that, on some input i, the path taken through the graph is EabcdedefbcfgX, and the path taken for P' is EabcdefbcdefgX. Then for both P and P', b is executed twice, g is executed once, d is executed twice, and f is executed twice. For P, however, the paths are EabcdeX, EdeX, EbcfX, EfX, and EgX, whereas for P', the paths are EabcdeX, EfX, EbcdeX, EfX, and EgX. Thus, equality does not hold, and the subsumption is strict.

4. $DCS \rightarrow DHS$, $BCS \rightarrow BHS$ $PCS \rightarrow PHS$.

Let i be an input that is S–hit-differencing for P and P', there exists some entity e (branch, path, data-dependence) that differs in the S-hit spectrum for P and P' on i. Suppose that P's spectrum contains e (branch, path, or data-dependence) that is not contained in P''s spectrum. Then the number of occurrences of e in P's spectrum is at least 1, whereas the number of occurrences of e in P''s spectrum is 0, which means that i is S–count-differencing for P and P'. Thus, S–count subsumes S-hit.

To show that the subsumption is strict, consider an input i that is S–count–differencing, and let P and P' differ only in the number of times that loops are executed. Then, the S–count spectra for P and P' would differ but the S–hit spectra would be the same. Thus, equality does not hold, and the subsumption is strict.

5. PHS $\rightarrow$ BHS.

Let i be an input that is BHS-differencing for P and P'. Suppose that P's BHS contains some branch e that is not in P''s BHS. Then P's PHS must contain at least one path containing e, whereas P''s PHS contains none of the paths containing e. Thus, i is PHS-differencing for P and P', and PHS subsumes BHS.

To show that the subsumption is strict, consider the control flow graph of program P, shown on the left in Figure 3. Suppose that, on some input i, the actual path taken through the graph is EabcfbcfgX. The BHS for P contains branches b, f, and g; the PHS contains paths EabcfX, EbcfX, and EgX. Now suppose that P' is such that it executes the loop only one time. Then the BHS for P' contains b, f, and g – the same as the BHS for P. However, the PHS for P' has EabcfX and EgX. Thus, equality does not hold, and the subsumption is strict.

6. OPS is incomparable with CPS, PCS, PHS, BCS, and BHS.

6a. Consider OPS with CPS. Let P be a program with computation $x = a+b$, and let P' be a modified version in which $x = a + b$ is changed to $x = a - b$, such that neither $x = a+b$ nor $x = a-b$ affects the flow of control in P or P'. In this case i can be OPS-differencing, but it is not CPS-differencing for P, P', and i. Thus, CPS does not subsume OPS, which means that none of CPS, PCS, PHS, BCS, or BHS subsumes OPS.

6b. Now consider BHS and OPS,and a program P in which a section of code is replaced by a semantically equivalent piece of code that has a different control flow structure to get P', let i be an input that produces the same output for P and P' (of course all i should do this) but traverses different branches in P'. In this case, i is BHS-differencing but not OPS-differencing. Thus, BHS does not subsume OPS, which means none of CPS, PCS, PHS, BCS, or BHS subsumes OPS.

By 6a and 6b, OPS is incomparable to CPS, PCS, PHS, BCS, and BHS.

7. DCS and DHS are incomparable with CPS, PCS, PHS, BCS, and BHS.

7a. Consider DHS with CPS. Let P contain a statement $s : a = B + C$, and let P' contain a modified version $s' : a = b+1$. Let i be an input that produces the same complete path in P and P'. Clearly i produces a difference in the definition-use pairs in P and P', and thus, i is DHS-differencing for P, P', and i. However, because the path executed in P and P' for i is the same, i is not CPS-differencing for P, P', and i. Thus, CPS does not subsume DHS, which means that none of CPS, PCS, PHS, BCS, or BHS subsumes DHS or DCS.

7b. Now consider BHS and DCS, and the partial control flow graph for shown in Figure 4. Assume that this graph models both P and P', and that the only difference between the two is that the conditional statement associated with the node that is the target of edge a has been changed. Let i be such that when P is executed with input i, it takes the path EabcdfbceghiX and when P' is executed with input i, it takes the path EabcdfhiX. In this case, i is BHS-differencing for P and P' but not DCS-differencing for P and P'. Thus, DCS does not subsume BHS, which means that neither DCS nor DHS subsumes any of CPS, PCS, PHS, BCS, or BHS.

By 7a and 7b, DCS and DHS are incomparable with

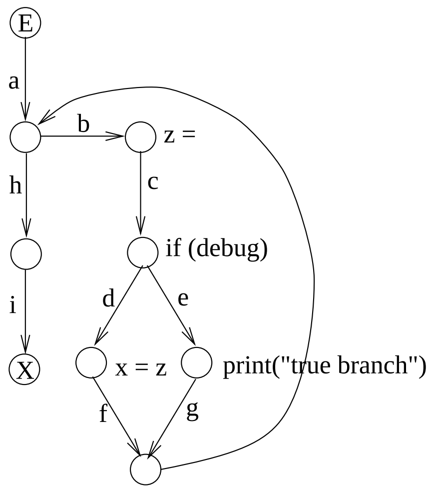

Figure 4: Subgraph used for proof of Theorem 1.

CPS, PCS, PHS, BCS, or BHS.

8. OPS is incomparable with DCS and DHS.

8a. Consider OPS and DCS, and the example given in part 6a above. If i is such that it produces a different output on P and P', then i is OPS-differencing, but not DCS-differencing for P and P'. Thus, neither DCS nor DHS subsumes OPS.

8b. Now consider DHS and OPS, and the example given in 7a above with i such that P and P' produce the same output when executed with i as input. Then i is DCS-differencing, but not OPS-differencing for P, P', and i. Thus, OPS does not subsume DCS, which means that it does not subsume DHS.

By 8a and 8b, OPS is incomparable with both DCS and DHS.

9. BCS and PHS are incomparable.

9a. Consider the partial control flow graphs on the left of Figure 3, and the case where input i causes execution of the path EabcfbcfbcfX when used as input for P but causes execution of the path EabcfbcfX when used as input for P'. The number of times branch b is executed differs in P and P', and thus i is BCS-differencing for P and P'. The paths that are executed in P and P', however, are the same – EabcfX, EbcfX, and EgX in both cases – and thus i is not PHS-differencing for P and P'. Thus PHS does not subsume BCS.

9b. Now consider the partial control flow graph on the right of Figure 3, and i such that when P is executed with i, it takes the path EabcdfbceghX and when P' is executed with i, it takes the path EabcegbcdfhX. Clearly, i is PHS-differencing, but not BCS-differencing, for P and P'. Thus, BCS does not subsume PHS.

From 9a and 9b, BCS and PHS are incomparable.

This completes the proof. □

3 The Experiment

3.1 Objectives

The objectives of our experiment were to empirically investigate the following questions:

1. Given program P, faulty version P', and universe of inputs U for P, what correlation exists between inputs that cause P and P' to produce different spectra. More precisely:
   (a) How often does an input $i \in U$ that causes P' to fail produce different spectra for P and P'?
   (b) How often does an input $i \in U$ that produces different spectra for P and P' cause P' to fail?
2. What are the relationships between the various spectra types, both in terms of their correlation with program-failure behavior, and in terms of their correlation with one another?

3.2 Measures

To quantify 1(a) and 1(b), we utilize two measures. Given program P, faulty version P', and universe of inputs U for P, let $FR(P, P', U)$ be the set of inputs in U that cause P' to fail (i.e., $FR(P, P', U)$ are Fault Revealing for P, P', and U). For each spectra variety S, let $SR(P, P', U)$ be the set of inputs in U that produce spectra on P and P' that differ (i.e., $SR(P, P', U)$ are spectrum S Revealing for P, P', and U).

Degree of Imprecision. For spectra type S, any input i in U that is in $SR(P, P', U)$ but not in $FR(P, P', U)$ exhibits a spectra difference under S that is not correlated with a failure. For such an input i, S-spectra-comparison is "imprecise". The degree of imprecision of S with respect to P, P' and U is given by the equation:

$$\frac{|SR(P, P', U) - FR(P, P', U)|}{|SR(P, P', U)|} * 100 \qquad (1)$$

Degree of Unsafety. For spectra type S, any input i in U that is in $FR(P, P', U)$ but not in $SR(P, P', U)$ exhibits a failure that is not correlated with a spectra difference of type S. For such an i, S-spectra-comparison is "unsafe". The degree of unsafety of S with respect to P, P', and U is given by the equation:

$$\frac{|FR(P, P', U) - SR(P, P', U)|}{|FR(P, P', U)|} * 100 \qquad (2)$$

Note that we can determine the degree of imprecision of S even though S is not safe. In this respect, our use of the term "imprecision" differs from its use in areas such as compiler optimization, where safe analyses are required. In the context of maintenance and testing, however, safe analyses are not always necessary – a technique that identifies a sufficiently large subset of some set of facts can be useful, even though it omits some members of that set of facts. However, we still wish to know, even of unsafe analyses, the degree to which they identify spurious results. Our use of "imprecision" supports this.

Note further that when P is correct for input i, and when P' is intended to have the same output for i as P, then i is in $FR(P, P', U)$ if and only if P and P' produce different output for i, or equivalently, if and only if i is in $OPSR(P, P', U)$.

3.3 Experimental Instrumentation

We used seven C programs as experimental subjects (Table 3). Each program has a variety of versions — each containing one fault-inducing modification — whose control flow graph structure is the same as that of the original program. Each subject program also has a large universe of inputs.[2] By construction, for each of our experimental programs P (with their universe U) and versions P', $FR(P, P', U) = OPSR(P, P', U)$.

Program	Lines of Code	Number of Versions	Input Universe Size	Description
totinfo	431	22	1052	information measure
schedule1	416	7	2650	priority scheduler
schedule2	309	2	2710	priority scheduler
tcas	238	26	1608	altitude separation
printtok1	584	4	4130	lexical analyzer
printtok2	513	7	4115	lexical analyzer
replace	569	20	5542	pattern replacement

Table 3: Subject programs.

We used a variety of tools and techniques to compute and record the various types of spectra. For the output spectra (OPS), we ran P and the P's on the inputs in U. For the execution trace spectra (ETS), we used our test selection tool, `DejaVu` [12], to identify the inputs in U that traverse modified statements in the P's.[3] For the branch-hit spectra (BHS), the branch-count spectra (BCS), the path-hit spectra (PHS), the path-count spectra (PCS) the data-dependence-hit spectra (DHS),

[2] These programs, versions, and inputs were assembled by researchers at Siemens Corporate Research for a study of the fault-detection abilities of control- and data-flow coverage criteria [8].

[3] In general this approach may identify a superset of the inputs that produce different execution traces; however, in practice we can determine when the approach incurs imprecision, and we know that in all the cases examined for our experimentation, the algorithm identified precisely the inputs that produced different execution traces.

the data-dependence-count spectra (DCS), and the complete-path spectra (CPS) we used the various coverage tools from the `Aristotle` analysis system [6] and the `FATE` data-flow testing system [9] to record the entities (i.e., branches, paths, definition-use pairs, complete paths) executed in P and the P's.

3.4 Experimental Design

Variables. The experiment manipulated a single independent variable, namely, the spectra: OPS, ETS, BHS, BCS, CPS, PHS, PCS, DHS, and DCS. On each run (with P, P', and U), we measured a single dependent variable, namely, the set of inputs in U that revealed spectra differences between P and P'. We utilized this data to examine the degrees of imprecision and unsafety of the various spectra, and to compare spectra, using an analysis strategy described later in this section.

Design. This experiment uses a *within-subjects* design: we applied each spectra calculation to each (base program, modified version) pair, for each input in the universe for that base program. We then examined the spectra with respect to the OPS results obtained on that base program, version and universe, to measure the values of dependent variables.

Threats to Validity. The primary threats to validity for this experiment are external: these are conditions that limit our ability to generalize the results of our study to a larger population of subjects. First, the subject programs are not large, and we cannot claim that they represent a random selection over the population of programs as a whole. Second, the faulty program versions all involve simple, one- or two-line faults, manually seeded, with the intent of simulating "real" faults, but with no data to indicate that they represent a random selection over the population of faults as a whole. These threats can be reduced only by repeated application of the experiment on wider classes of subjects.

A second source of threats to validity for this study are internal: these are influences that can affect the dependent variables without the researchers' knowledge. Our greatest concerns here involve instrumentation effects, which can bias our results. To control for such effects, we utilized two types of cross-checks: (1) we obtained certain results independently using different instruments at different sites and examined them for correlation; (2) we examined results for adherence to the analytically determined spectra hierarchy relationship. We did not, however, control for the structure of source programs or for the locality of program changes.

Analysis Strategy. We utilized the data gathered from our experimental runs as follows. First, to examine the correlation between inputs that cause P' to fail and inputs that cause P to produce different spectra than P' (objective 1), for each program P and version P', with respect to universe U, and for each spectra type S, we calculated:

1. The degree of imprecision of S with respect to P, P', and U.
2. The degree of unsafety of S with respect to P, P', and U.

Second, to compare spectra (objective 2), for each program P and version P', with respect to universe U, and for each pair of spectra types S_1 and S_2, we calculated:

1. The number of inputs in U that cause spectra differences of type S_1.
2. The number of inputs in U that cause spectra differences of type S_2.
3. The number of inputs in U that cause spectra differences of type S_1 but not of type S_2.
4. The number of inputs in U that cause spectra differences of type S_2 but not of type S_1.

We also summarized this data over the entire set of (program, modified version) pairs, considering each for the entire universe U of inputs (245,087 inputs). The next section presents and analyzes this data.

4 Data and Analysis

Figure 5 uses boxplots to present the degrees of imprecision and unsafety calculated for each spectra over the 88 different (program, modified program) pairs. (The caption provides an explanation of boxplots.) The data reveals several things relative to our first objective. First, the unsafety data (left) supports the conjecture that we can expect to see spectra differences on inputs that elicit faults. Every variety of spectra demonstrates a median degree of unsafety of 0%. Furthermore, four varieties of spectra (CPS, PCS, BCS, and DCS) demonstrate a 0% degree of unsafety over their entire first, second, and third quartiles; in other words, on over three quarters of the (program, modified program) pairs, these spectra demonstrated differences on *every* input that elicited a fault. For the BHS and PHS spectra, the degrees of unsafety demonstrate a greater interquartile range, reflecting greater diversity in results. Finally, all spectra other than ETS displayed occasional extreme results, represented by the outliers, and further reflected in the fact that their means are noticeably higher than their

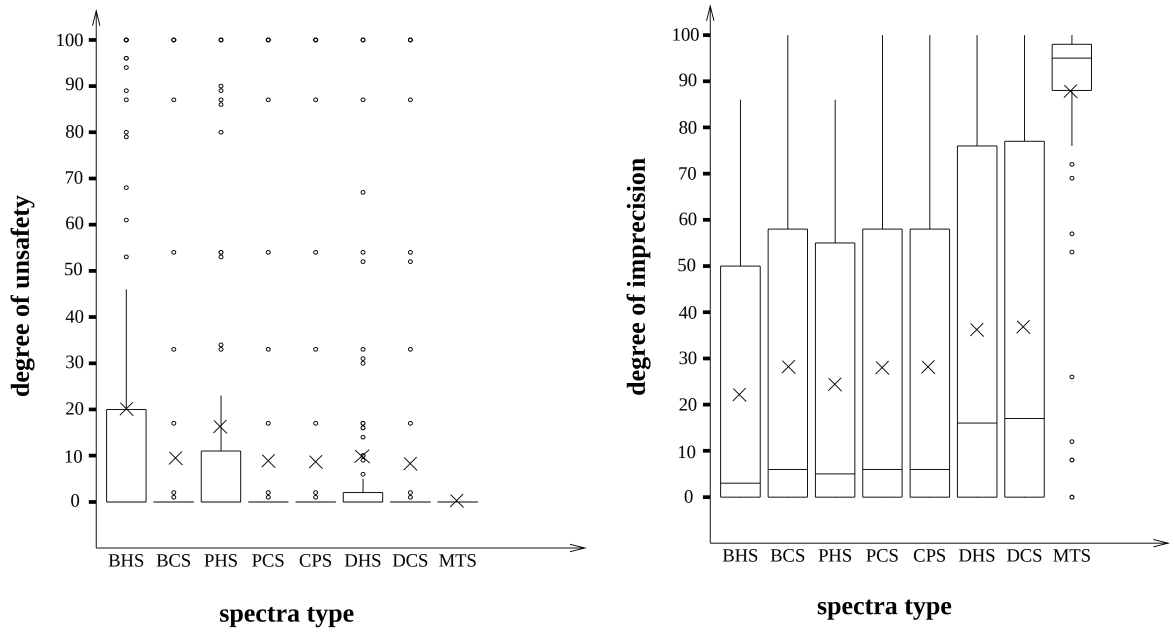

Figure 5: Graphs showing the degrees of unsafety and imprecision of spectra. The vertical axes list degrees of unsafety and imprecision, respectively; the horizontal axes list spectra types. In each boxplot, the dashed line represents the median, and the "X" mark represents the mean, of the degree of imprecision or unsafety that occurred for that spectra. The box indicates the interquartile range – the range in which the middle half of the data falls – and also indicates where that data falls with respect to the median. The whiskers above and/or below boxes indicate the percentages at which data above or below the interquartile range fell; however, data points at a distance of greater than 1.5 times the interquartile range are considered outliers, and represented by small circles.

medians. Significantly, only ETS is safe: in all cases, all inputs that reveal faults also reveal ETS spectra differences, and no other spectra share this trait.

The imprecision data presented in Figure 5 (right) illustrates that not all inputs that elicit spectra differences also produce failures: all varieties of spectra incur imprecision. Significantly, ETS incurs a much higher median degree of imprecision (95%) than the other varieties of spectra, for which median degree of imprecision ranges from 3% to 17%. Thus, the cost of the safety achieved by ETS, in terms of degree of imprecision, is high. However, imprecision results for all spectra other than ETS display a large interquartile range; thus, degree of imprecision varies a great deal more, overall, than degree of unsafety.

One additional significant result is that in terms of both degrees of imprecision and unsafety the CPS, PCS, and BCS spectra display nearly identical behavior. These results do not support the conjecture that path spectra will be more sensitive indicators of different behavior than branch spectra, where behavior is measured in terms of program failure behavior.

Toward our second objective, Table 4 lists data pertaining to the relationship between various spectra. Figure 6 provides a graphical view of the portion of that data that compares the various spectra with OPS, within the context of the entire universe of inputs as applied to all (program, modified program) pairs (see the caption for further description.) The reader may also compare the sizes of shaded areas in Figure 6 to obtain a notion of the relationships among the spectra. In particular, it is easy to see the relationship between ETS, DCS, DHS, and CPS from this figure.

Figure 7 provides a closer view of the relationship between CPS, PCS, PHS, BCS, and BHS that shows, for each of these spectra types, the number of inputs for which spectra differences occurred. Like the imprecision and unsafety results, this figure illustrates the near equivalence, over our programs, modified versions, and input universes, of the CPS, PCS, and BCS spectra, as well as the relative closeness of the PHS and BHS spectra. Furthermore, where analytical results foretell that neither PHS nor BCS subsumes the other, in practice BCS subsumes PHS: BCS spectra differences occurred in 5836 cases in which PHS spectra differences did not occur, but in all cases in which PHS spectra differences occurred, BCS spectra differences also occurred.

These results do not support the conjecture that path spectra will be more sensitive indicators of different behavior than branch spectra. For the 245,087 inputs, PCS is never more sensitive than BCS, and CPS is more sensitive than PCS on only 7 inputs. Essentially,

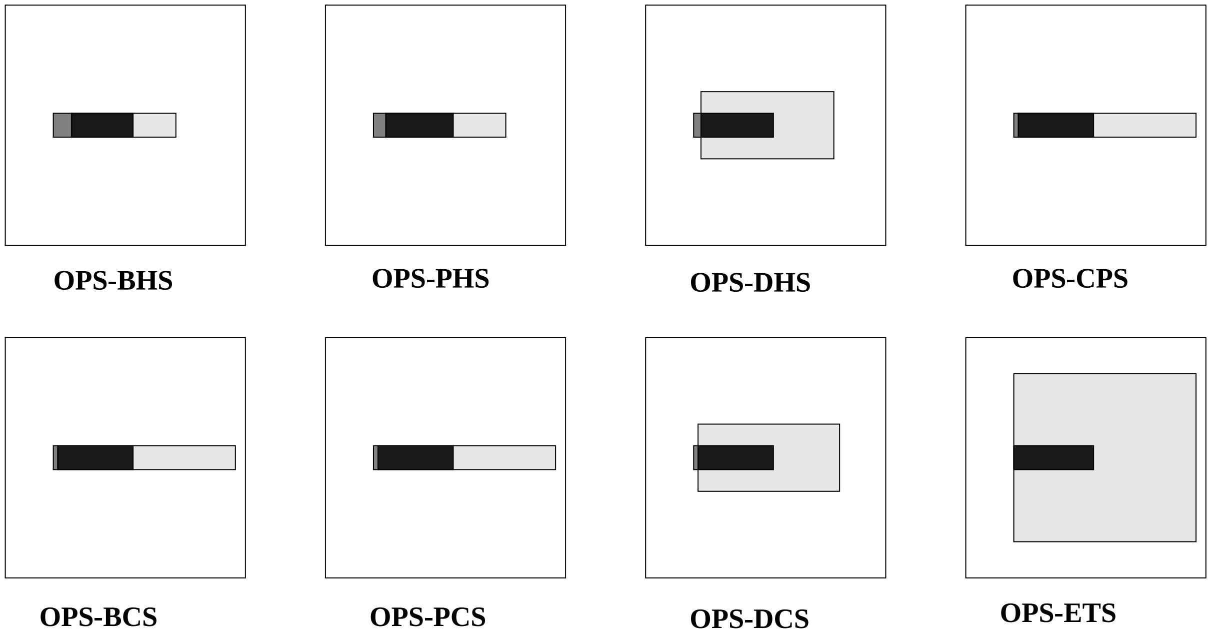

Figure 6: Graphical comparison of OPS with the other spectra. The eight outer squares represent the comparison of the OPS spectra to the other eight spectra, respectively, as labeled. Each such square represents the entire universe of input points over all (program, modified version) pairs. Within the outer squares, the lightly shaded areas indicate the percentages of input points that caused only XS-spectra differences ($XS \neq OPS$), the medium shaded areas represent the percentages of input points that caused only OPS-spectra differences, and the darkly shaded areas represent the percentages of input points under both XS and OPS.

despite the analytical differences between spectra depicted by our subsumption hierarchy, in the cases we studied, the CPS, PCS and BCS spectra collapse into one another. The three spectra have nearly equivalent abilities to distinguish differences in program behaviors.

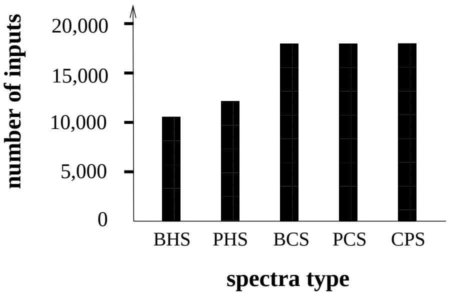

Figure 7: Comparison of CPS, BCS, PCS, PHS, and BHS, showing, for each spectra type (horizontal axis), the number of inputs for which spectra differences occurred (vertical axis).

5 Conclusions

We have described an empirical investigation of the correlation between spectra and program failure behavior, and the relationships between various types of spectra.

We emphasize that our study has considered only one scenario in which the usefulness of program spectra has been postulated: the scenario in which spectra from a program and modified version, run on the same input, are compared. Our results do not provide data applicable to the use of spectra when a single program is run on two slightly different inputs. However, given the degree to which spectra types produced nearly equivalent results for our subjects, further empirical study of the relationship between spectra types in the latter scenario would be appropriate.

We also stress that our study has focused on just one indicator of program behavior – fault-revealingness. This indicator is important, and the fact that spectra do correlate with it – at least in one direction – is significant. Whether spectra will correlate with other measures of behavior, such as measures based on sequences of execution states [10], is a subject for future investigation. Our comparisons of spectra to each other, however, are not restricted to fault-revealing behavior.

Finally, as discussed earlier, there are some threats to validity for this experiment, primarily concerning representativeness of subjects. Additional experimentation with a variety of subjects is necessary to lessen these threats.

Keeping the foregoing in mind, however, our results support several conclusions. First, although the execution trace spectra emerged as the only spectra to neces-

A	B	C	D	E
Spectra ($S1$-$S2$)	Size of $S1(*,*,U)$	Size of $S2(*,*,U)$	$S1(*,*,U)$ - $S2(*,*,U)$	$S2(*,*,U)$ - $S1(*,*,U)$
OPS-BHS	8076	10536	1868	4328
OPS-BCS	8076	17956	469	10349
OPS-PHS	8076	12120	1266	5310
OPS-PCS	8076	17956	469	10349
OPS-DHS	8076	37881	743	30548
OPS-DCS	8076	40372	448	32744
OPS-CPS	8076	17963	469	10356
OPS-ETS	8076	130151	0	122075
BHS-BCS	10536	17956	0	7420
BHS-PHS	10536	12120	0	1584
BHS-PCS	10536	17956	0	7420
BHS-DHS	10536	37881	16	27361
BHS-DCS	10536	40372	12	29848
BHS-CPS	10536	117963	0	7427
BHS-ETS	10536	130151	0	119615
BCS-PHS	17956	12120	5836	0
BCS-PCS	17956	17956	0	0
BCS-DHS	17956	37881	2502	22427
BCS-DCS	17956	40372	12	22428
BCS-CPS	17956	17963	0	7
BCS-ETS	17956	130151	0	112195
PHS-PCS	12120	17956	0	5836
PHS-DHS	12120	37881	29	25790
PHS-DCS	12120	40372	12	28264
PHS-CPS	12120	17963	0	5843
PHS-ETS	12120	130151	0	118031
PCS-DHS	17956	37881	2502	22427
PCS-DCS	17956	40372	12	22428
PCS-CPS	17956	17963	0	7
PCS-ETS	19191	136226	0	117035
DHS-DCS	37881	40372	0	2491
DHS-CPS	37881	17963	22426	2508
DHS-ETS	37881	130151	0	92270
DCS-CPS	40372	17963	22427	18
DCS-ETS	40372	130151	0	89779
CPS-ETS	17963	130151	0	112188

Table 4: Comparison of spectra summarized over all (program, modified version) pairs, considering each for the entire input universe (245,087 inputs). Column **A** lists the spectra compared; Column **B** lists the total number of inputs that cause spectra differences of type $S1$; Column **C** lists the total number of inputs that cause spectra differences of type $S2$; Column **D** lists the total number of inputs that cause spectra differences of type $S1$ but not of type $S2$; Column **E** lists the total number of inputs that cause spectra differences of type $S2$ but not of type $S1$.

sarily exhibit differences for inputs that exhibit faults, certain other types of spectra differences (i.e., CPS, PCS, and BCS) also correlate with high frequency (at least in one direction) with fault occurrences. When failures exist on particular inputs, spectra differences are likely also to exist on those inputs. Moreover, with these spectra, the degree of imprecision – the frequency with which spectra differences exist but faults do not – is much lower than with the execution trace spectra.

Another conclusion involves the near equivalence of the CPS, PCS, and BCS spectra in terms of their ability to distinguish program behaviors. Program instrumentation has a cost, and in practice we must balance that cost against the potential savings, while also considering the criticality of the application. In many situations, it may not be cost-effective to collect the complete traces required for CPS spectra; however, if our results generalize, PCS and BCS spectra possess power almost equivalent to that of CPS, and may be cost-effective alternatives. Furthermore, estimates suggest that the profiling necessary to collect BCS spectra incurs a 16% run-time overhead whereas the profiling necessary to collect PCS spectra incurs a 30% run-time overhead [4]. In the absence of a gain in sensitivity, use of the PCS spectra instead of the BCS spectra would not be worth the added overhead required to collect the more sensitive spectra.

References

[1] G. Ammons, T. Ball, and J. R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *ACM Sigplan Notices*, 32(5):85–96, June 1997.

[2] T. Ball. On the limit of control-flow analysis for regression testing. In *Proc. of the ACM Int'l. Symp. on Softw. Testing and Analysis*, Mar. 1998.

[3] T. Ball and J. R. Larus. Efficient path profiling. In *Proc. of Micro 96*, pages 46–57, Dec. 1996.

[4] T. Ball, P. Mataga, and M. Sagiv. Edge profiling versus path profiling: The showdown. In *Proc. of 25th ACM Symp. on Prin. of Prog. Lang.*, pages 134–148, Jan. 1998.

[5] Y. Chen, D. Rosenblum, and K. Vo. TestTube: A system for selective regression testing. In *Proc. of the 16th Int'l. Conf. on Softw. Eng.*, pages 211–222, May 1994.

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

[7] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation and comparison of program spectra. Technical Report OSU-CISRC-11/97-TR55, The Ohio State University, Nov. 1997.

[8] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. of the 16th Int'l. on Softw. Eng.*, pages 191–200, May 1994.

[9] J. Lloyd and M. Harrold. Implementing an interprocedural dataflow tester using abstract execution. Technical Report 95-111, Clemson University, Clemson, SC, May 1995.

[10] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. *ACM Software Engineering Notes*, 22(6):432–439, Nov. 1997.

[11] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Trans. on Softw. Eng.*, 22(8):529–551, Aug. 1996.

[12] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. on Softw. Eng. and Methodology*, 6(2):173–210, Apr. 1997.