Debugging Plan Failures by Analyzing Execution Traces

Adele E. Howe, Paul R. Cohen

Computer Science Technical Report 92-22

Experimental Knowledge Systems Laboratory
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

Debugging plan failures involves isolating the actions and events that contributed to the failure and explaining how they caused the failure. In this paper, we present an approach to the first step in debugging in which we statistically analyze execution traces of a planner to isolate dependencies between actions and failures. The analysis is based on G-test comparisons of observed and expected frequencies of sequences of actions and failures. We apply the approach to isolate dependencies between failure recovery actions and failures in execution traces of the Phoenix planner. Our analysis of Phoenix suggests that even small changes to the design of a planner result in significant changes in the planner's performance. However, our approach allows us to identify the dependencies between changes in design and changes in performance.

This research was supported by a DARPA-AFOSR contract F49620-89-C-00113, the National Science Foundation under an Issues in Real-Time Computing grant, CDA-8922572, and a grant from the Texas Instruments Corporation. We wish to also thank Eric Hansen and Scott Anderson for their comments on earlier drafts.

1 Introduction

A plan fails when it does not achieve its desired effect. Plan failures are caused by unexpected events at execution time [4], insufficient knowledge of the world [2], actions not producing their intended effect [6], or unanticipated interactions [12]. Debugging failures involves isolating the actions and events that contributed to the failure and explaining how they caused the failure. In this paper, we present an approach to the first step in debugging in which we statistically analyze execution traces of a planner to isolate dependencies between actions and failures.

Other approaches to debugging planners are more knowledge intensive than ours. Sussman's HACKER detects, classifies and repairs bugs in blocks world plans, but it requires considerable knowledge about its domain, which is itself limited[12]. Hammond's CHEF backchains from failure to the states that caused it, applying causal rules that describe the effects of actions[5]. Simmons's GORDIUS debugs faulty plans by regressing desired effects through a causal dependency structure constructed during plan generation from a causal model of the domain[10]. Kambhampati's approach [8], in contrast, is less domain-dependent; his theory of plan modification compares the validation structure (an explanation of correctness generated with the plan) to the planning situation, detects inconsistencies, and uses the validation structure to guide the repair of the plan.

While these systems are comprehensive in scope (i.e., they isolate dependencies, explain and repair the failure), they assume that the planner knows enough to determine what is wrong and fix it. The approach presented in this paper requires little knowledge – either of the domain or of the planner – to identify dependencies between actions and failures. Complementary to the more knowledge intensive approaches, ours is most appropriate when a rich domain model is not available or when the existing model may be incorrect or buggy, as when the system is under development.

2 Isolating Failure Dependencies

Previous experiments and analyses of failure recovery in the Phoenix system showed that changing the set of recovery actions changed the type and frequency of failures [7]. In these experiments, we discovered that seemingly minor changes to the design of Phoenix's failure recovery component, such as adding two new recovery actions with limited applicability, changed system performance much more than expected. For example, although we changed how the system responded to only three of eleven failure situations, performance in the other eight situations degraded, and the number and variety of failures increased. In the current analysis, we are trying to understand how failure recovery actions might have effects beyond their expected scope:

1. Does a failure depend on the recovery action that preceded it?

2. Does a failure depend on the failure that preceded it?

3. a: Does the effect of a recovery action on the next failure depend on the failure to which the recovery action was applied (b: conversely, does the effect of a failure on the next failure depend on intervening actions)?

To address these three questions, we will describe failures in Phoenix, present some statistical techniques, and describe the results of applying the techniques to execution traces from Phoenix.

2.1 Failures in the Phoenix System

Our research on debugging failures was conducted on the Phoenix system[3]: a simulator of forest fire fighting in Yellowstone National Park and an agent architecture. A single agent, the fireboss, coordinates the efforts of field agents who build fireline to contain the spread of the fire. Fire spread is influenced by weather and terrain, but even when these factors remain constant, the spread is unpredictable. Plan failures are a natural result of this unpredictability of the environment, but they may also result from flaws in Phoenix's plans.

A failure is detected when a plan cannot execute to completion. Failures may be detected during plan generation or execution, and are classified into 11 domain-specific types. Failure classification is a common part of failure detection; many systems monitor for specific failure events such as on-coming vehicles or changes in vehicle paths [9], and unexpanded actions or unsupported preconditions in a planner that interleaves planning and execution [1]. Failures in Phoenix are classified by how they are detected and what they failed to do. For example, a *cant-calculate-projection* failure (abbreviated *prj*) is detected during plan generation, whereas a *violation-insufficient-time* failure (abbreviated *vit*) is detected through execution monitoring when a plan will take longer to complete than it has been allotted.

To repair a failure, the planner applies one of a set of actions: usually six, but in one version of the system, eight. Most of the actions can be applied to any failure, but the scope and nature of their repairs varies. For example, *replan-parent* (abbreviated *rp*) is applicable to any failure and recomputes the plan from the last major decision point; while *add-another-bulldozer* (abbreviated *ab*) repairs only *vit* failures by adding a few actions to the failed part of the plan.

2.2 Techniques for Isolating Dependencies

We answer the three earlier questions about the effects of failure recovery on failures by statistically analyzing execution traces. The strategy is to view execution traces as transitions between failure types and recovery actions and analyze these transitions for dependencies. The analysis is a two step process: Combinations of failures and actions are first tested for whether they are more or less likely to be followed by each of the failures. Then the significant combinations are compared to remove combinations subsumed by more general ones.

Assume we observe the following trace:

$$F_f^1 \rightarrow A_a \rightarrow A_b \rightarrow A_c \rightarrow F_g^2 \rightarrow A_b \rightarrow F_f^3 \rightarrow A_c \rightarrow A_b \rightarrow F_g^4$$

where F's are failure types and A's are actions.[1] The subscripts indicate individuals from a set, so F_f means failure of type f; the superscripts indicate temporal order. It appears from this short trace that failure F_g is always preceded by failure F_f and that the intervening actions do not matter. If F_f does indeed cause F_g directly or indirectly, then we would expect to see F_g following F_f more often than F_g follows other failures; conversely, if the two failures are independent, F_g should be no more likely after F_f than after any other failure. Without many more examples of F_f and F_g, we cannot reliably conclude that F_f and F_g are related.

2.2.1 G Test Filtering to Identify Patterns

An execution trace is an alternating sequence of actions and failures. Each failure is followed by an action to repair the failure, but another failure soon follows. A well-designed failure recovery component, like the Hippocratic physician, should do no harm; the cure for one failure should not increase the likelihood of later failures. Fortunately, we can detect this and related situations if they arise. All we require is a test for statistical dependence in categorical data. We can isolate a type of failure recovery action, call it A_a, from the set of actions A to see whether a failure F_f from the set of failures F is especially likely to follow A_a. We count the instances of F_f that follow instances of A_a, and also the instances of F_f that follow instances of all actions other than A_a (abbreviated $A_{\overline{a}}$). We also count instances of failures other than F_f (abbreviated $F_{\overline{f}}$) that follow A_a and that follow $A_{\overline{a}}$. These four frequencies are arranged in a contingency table, like the one in table 1. In this case we see a strong dependence or association between A_a and F_f: 20 cases of F_f follow A_a and only 10 failures other than F_f follow A_a. But while action A_a leads most frequently to failure F_f, actions other than A_a lead to F_f relatively infrequently (30 instances in 130). A G-test[2] will detect this dependence between A_a and F_f; in this case, $G = 25.710, p < .001$, which means that the contingency table in figure 1 is extremely unlikely to have arisen by chance if A_a and F_f are independent.

But table 1 might not tell the whole story. Perhaps not A_a but something that preceded A_a is responsible for the increased incidence of failure F_f. What immediately preceded A_a was, of course, a failure (recall that A_a is a failure recovery action); so perhaps the numbers in table 1 represent a dependence between successive failures, and A_a just happened to get caught in the middle. A third possibility is that both are true:

[1] In our analysis, the actions are recovery actions, but they could as well be any planning decisions, actions or percepts thought to influence failures. The analytical techniques do not require single intervening actions, but could support multiple actions and could certainly support planning actions instead of recovery actions.

[2] We explain the G-test in the appendix using data from a later example.

	F_f	$F_{\bar{f}}$
A_a	20	10
$A_{\bar{a}}$	30	100

Table 1: A contingency table of frequencies of failure F_f and all other failures, following action A_a and all other actions.

successive failures are not independent *and* the intervening action increases the likelihood of the second failure.

Three kinds of contingency tables differentiate these three possibilities. We illustrated an *action-failure* ($A - F$) table in table 1. We construct one such table for each action-failure pair that arises in our execution traces. When we see an instance of A_a in the execution trace, we look at the next failure; if it is F_f, we add one to the first quadrant of the table. Otherwise, we add one to the second quadrant. When we see an instance of an action other than A_a (i.e., $A_{\bar{a}}$), if the following failure is F_f we add one to the third quadrant, and if the following failure is $F_{\bar{f}}$ we add one to the fourth quadrant. We build *failure-failure* ($F - F$) tables similarly, with one table for each pair of successive failures. Lastly, we build *failure-action-failure* ($FA - F$) tables by treating the first failure and the intervening action as a unit, and looking for dependencies between the second failure and each such unit.

With these tables and the G-test, we can answer the three questions we asked earlier. To answer the first question, whether a particular action A_a influences the incidence of a particular following failure, F_f^2, we simply construct an $A - F$ contingency table like the one in table 1 and run a G-test. Similarly, we answer the second question, whether a particular failure F_g^1 affects the incidence of a particular next failure F_f^2, by constructing an $F - F$ contingency table and running a G-test.

Answering the third question is slightly more involved, so it will help to review the question:

Question 3a Is the dependency between action A_a and the following failure F_f^2 significant in the context of all or just some of the previous failures $F^1 \in F$?

Question 3b Is the dependency between failure F_g^1 and failure F_f^2 significant for all or just some of the intervening actions $a \in A$?

Imagine the answer to question 3a is that a dependency holds between action A_a and F_f^2 irrespective of the first failure F^1. In this case, the $A_a - F_f^2$ interaction *subsumes* all significant $F_x^1 A_a - F_f^2$ interactions for all first failures F_x^1. The first failure is irrelevant to the dependency between A_a and F_f^2. Similarly, if the dependency between F_g^1 and F_f^2 holds regardless of the intervening action, then the $F_g^1 - F_f^2$ interaction subsumes all significant $F_g^1 A - F_f^2$ interactions.

	F^2_g	$F^2_{\bar{g}}$
$F^1_h A_a$	20	5
$F^1_i A_a$	2	5
$F^1_j A_a$	1	3
$F^1_k A_a$	2	4
	25	17

Table 2: Example contingency table in which $A - F$ dependency is due to a particular $FA - F$ dependency

Another answer to question 3a is that a dependency holds between A_a and F^2_f for some first failures but not others. In this case, we should not claim an overall $A_a - F^2_f$ dependency, despite finding it in an $A - F$ contingency table. The reason is that the effect we observe in the $A - F$ table could be due entirely to a strong $F^1A - F^2$ dependency. This situation is shown in table 2. We see that a particular failure, F^2_g, is relatively likely to follow the sequence, $F^1_h A_a$, but is relatively unlikely to follow any of the sequences $F^1_i A_a$, $F^1_j A_a$ or $F^1_k A_a$. Yet because the $F^1_h A_a - F^2_g$ dependency is so strong, we are quite likely to find a general $A_a - F^2_g$ dependency, also. If we claim on this basis that F^2_g is especially likely to follow A_a, then we are ignoring the other three rows in table 2. Running a G-test on table 2 tells us whether the second failure depends on the preceding $F^1_x A_a$ sequence. If it does, then we must report particular $FA - F$ dependencies in the table, not a general $A - F$ dependency. If the G test is insignificant, it means that the $A - F$ dependency subsumes all the $FA - F$ dependencies represented in the table. Therefore, when we detect a significant $A - F$ dependency, we always construct another contingency table, like the one just described, to test whether the $A - F$ effect subsumes corresponding $FA - F$ effects.

We follow the same strategy to answer question 3b but we construct a slightly different contingency table. Assume we find a significant dependency between F^1_f and F^2_g. If this effect subsumes the $F^1_f A_x - F^2_g$ effects for all intervening actions A_x, then a G-test on the $F^1_f A_i - F^2_g$ table should not be significant. If the test is significant it means one or more particular sequences (e.g., $F^1_f A_c$) influence the incidence of F^2_g, and we cannot cite a general $F^1_f - F^2_g$ effect.

Another way to run these tests is suggested by the column marginal frequencies in table 2. These are the observed frequencies, summed over all actions, of the incidence of F^2_g and $F^2_{\bar{g}}$ following A_a. They can be compared with the expected ratio of F^2_g and $F^2_{\bar{g}}$ observed in the entire population of execution traces. A G-test comparing these observed and expected frequencies will tell us whether the $A - F^2$ dependency in table 2 is significant. (It is closely related to the $A - F$ test, discussed earlier, but not identical.) Then we can ask how much each of the individual FAF rows contributes to this *pooled* effect. Two outcomes are particularly interesting: the pooled effect $(F^1 - F^2)$ is significant but the individual rows ($F^1A - F^2$) are less significant; and, both the pooled effect and the individual effects are significant. In the first case, the pooled effect subsumes the individual effects; in the second, the individual effects may conflict, resulting in a diluted pooled effect. Because the test and its interpretation are somewhat involved, the next subsection will work through examples of each case.

The techniques we have described are fully automated. A program runs G-tests on all combinations of failures and actions: $F^1 - F^2$, $A - F^2$, and $F^1A - F^2$. For every significant pair of $FA - F$ and $F - F$ combinations (e.g., $F^1_f A_a - F^2_g$ and $F^1_f - F^2_g$), the program executes a G-test on a $FA - F$ contingency table with F^1_f held constant and A varied. If the G-test on the $FA - F$ table shows the pooled effect to be more significant than the individual effects, then each $F^1_f A_x - F^2_g$ effect is removed from the set of effects; otherwise the individual effects are removed. (We explain what we mean by "more significant" in the next section.) Similarly, for every significant pair $F^1_f A_a - F^2_g$ and $A_a - F^2_g$, a G-test is run on an $FA - F$ table with F^1 varied and with A_a held constant; if the pooled effect is more significant, then each $F^1_f A_a - F^2_g$ effect is removed; otherwise, the $A_a - F^2_g$ effect is removed.

2.2.2 Examples of Comparing Effects

To answer questions 3a and 3b, we compare the expected and observed frequencies of the pooled effect ($F - F$ or $A - F$) to the individual effects ($FA - F$). The two interesting results are that the pooled effect subsumes the individual effects, or that the pooled effect is diluted and the individual effects are required. To explain how these tests are performed, we will work through two examples, one for each of the possible interesting results of the test, on data from the recovery experiments. In these examples, the subscripts will refer to specific recovery actions and failures in Phoenix.

Subsumed Effects Imagine that G-tests showed significant effects for $F^1_{prj} - F^2_{prj}$ and $F^1_{prj}A_{rp} - F^2_{prj}$. Is F^1_{prj} responsible for both effects, making A_{rp} irrelevant? Figure 1 illustrates the steps needed to answer the question.[3]

First we determine the frequencies of F^2_{prj} and $F^1_{\overline{prj}}$ for each of the actions used to repair F^1_{prj} (shown in the table A in figure 1). We sum over the actions to get the column margins of 116 and 126. Second, we construct a contingency table (table B) for these sums and the frequencies of F^2_{prj} and $F^2_{\overline{prj}}$ observed in the population. A G-test on this table yields a significant result, $G_P = 72.8, p < .001$, suggesting that $F^1_{prj} - F^2_{prj}$ is a significant effect. G_P denotes this pooled effect. Then, we construct contingency tables for each of the row distributions in table A, comparing the observed frequencies of F^2_{prj} and $F^2_{\overline{prj}}$, given each of the actions, to the expected population frequencies. Summing the individual G values yields a total G (called G_T). G_T is also the sum of the pooled effect

[3] The one step not shown in the figure is the computation of the G-test itself. The equation for the G-test and a description of its computation for this example are presented in the appendix.

A. Observed Frequencies

	F^2_{prj}	$F^2_{\overline{prj}}$
$F^1_{prj}A_{rv}$	12	19
$F^1_{prj}A_{ra}$	7	8
$F^1_{prj}A_{sa}$	18	14
$F^1_{prj}A_{rp}$	79	85
Pooled	116	126

B. Contingency Table for Pooled

	F^2_{prj}	$F^2_{\overline{prj}}$
F^1_{prj}	116	126
expected	508	163

$G_P = 72.8, p < .001$

C. Contingency Tables for Each Action

	F^2_{prj}	$F^2_{\overline{prj}}$	G
$F^1_{prj}A_{rv}$	12	19	3.91
expected	52	25	
$F^1_{prj}A_{ra}$	7	8	4.09
expected	31	10	
$F^1_{prj}A_{sa}$	18	14	16.56
expected	79	18	
$F^1_{prj}A_{rp}$	79	85	50.22
expected	346	110	
G_T			74.78

Figure 1: G-test comparing observed frequency of F^2_{prj} following F^1_{prj} to expected, for pooled and action specific dependencies

(G_P) plus any variance in the general direction or trend of the pooled effect introduced by differences in the individual samples (called G_H for heterogeneity) or:

$$G_T = G_P + G_H$$

Because $G_T = 74.78$ and $G_P = 72.8$, $G_H = 1.98$, which is not significant. Thus, we gain no more information knowing all the $F^1_{prj}A_x - F^2_{prj}$ effects than we have knowing only the pooled effect, $F^1_{prj} - F^2_{prj}$. Consequently, the pooled effect subsumes the $F^1A - F^2$ effects.

Diluted Effects If both G_P and G_H are both significant, however, then the $F^1A - F^2$ effects may conflict, resulting in a diluted pooled effect. In this case, we are inclined to disregard the pooled effect and credit the $F^1A - F^2$ effects for the dependencies. Table 3 shows just such a situation. Applying the G-test, we find effects for $F^1_{ccp}A_{rp} - F^2_{ccp}$, $F^1_{ner}A_{rp} - F^2_{ccp}$, $F^1_{prj}A_{rp} - F^2_{ccp}$, and $A_{rp} - F^2_{ccp}$. But two of the individual observed ratios (those involving *prj* and *ner*) are significantly lower than expected, that is, they tend to encounter *ccp* less often relative to the population distribution of F_{ccp} and $F_{\overline{ccp}}$, and one, *ccp*, is greater than expected. As in the previous example, we compute G_P for the sum of the frequencies of F^2_{ccp} and $F^2_{\overline{ccp}}$ following A_{rp} and all F^1_x relative to the population proportions. G_P quantifies the strength of the similarity across all individuals (each F^1 in which A_{rp} is applied). G_T is the extent to which the individual rows match the population proportions plus the extent to which they differ from each other. When we remove G_P from G_T, we are left with the extent to which the individual rows differ, G_H, which for this table is a significant difference. So we gain considerable information by knowing all the $F^1_xA_{rp} - F^2_{ccp}$ effects over knowing simply the $A_{rp} - F^2_{ccp}$ effect.

F^1A_{rp}	F^2_{ccp}	$F^2_{\overline{ccp}}$	G	P
$F^1_{vit}A_{rp}$	53	151	0.690	.406
$F^1_{cfp}A_{rp}$	3	12	0.105	.746
$F^1_{ccp}A_{rp}$	25	23	18.294	.000
$F^1_{ner}A_{rp}$	11	116	19.141	.000
$F^1_{bdu}A_{rp}$	3	3	1.981	.159
$F^1_{ccv}A_{rp}$	2	7	0.008	.928
$F^1_{prj}A_{rp}$	16	148	20.739	.000
G_T			60.958	.000
pooled	113	460	4.704	.030
G_H			56.254	.000

Table 3: Example of diluted effects: Homogeneity Test for A_{rp} and F^2_{ccp}

2.3 Results

The analysis requires frequency counts for all combinations of failures and their immediate predecessors. We culled these data from the set of experiments in which we evaluated failure recovery[7]. For that set, we ran three experiments of 100 trials each to evaluate three different designs for failure recovery: a basic design, the basic design augmented by an analytically derived control strategy, and the basic design augmented by the control strategy plus two new recovery actions. In each experiment, we collected data on failures and their repairs. For each trial of the experiments, we temporally ordered the data into execution traces and grouped the data into *[*F^1, A, F^2*]* triples where F^1 is repaired by A and F^2 is the following failure. The grouping resulted in about 950 triples for each experiment. From these, we generated contingency tables.

Because the experiment conditions differed between the three experiments, we analyzed each data set separately. The first and second data sets included instances of ten failure types and six recovery actions; the third included eleven failure types and eight recovery actions. Table 4 summarizes the number of effects found in each of the experiments as each test was applied. The failure types and actions in the three experiments produce 2697 possible different combinations, of which 632 were observed in the data. Because the G-test cannot be applied to contingency tables with empty cells (i.e., combinations without examples in the data), only 632 combinations were tested. Of these 632 combinations, G-test filtering identified 190 as significant: 42 $A - F^2$, 58 $F^1 - F^2$ and 90 $F^1A - F^2$ effects. Testing for subsumed and diluted effects (called G_H in the table) dropped the total number to 101.

The analysis above addresses dependencies *within* an experiment. Yet, part of our motivation was to understand why the pattern of failure changed across our three experiments. Because the incidence of failure changes, we expected the failure dependencies

		Exp. 1	Exp. 2	Exp. 3	Totals
$A - F^2$	combinations	60	60	88	208
	non-empty	42	38	48	128
	G-Test	7	15	20	42
	G_H	4	8	15	27
$F^1 - F^2$	combinations	100	100	121	321
	non-empty	58	57	58	123
	G-Test	15	21	22	58
	G_H	13	19	15	47
$F^1A - F^2$	combinations	600	600	968	2168
	non-empty	125	97	109	331
	G-Test	21	29	40	90
	G_H	7	4	16	27
Totals	combinations	760	760	1177	2697
	non-empty	225	192	215	632
	G-Test	43	65	85	190
	G_H	24	31	46	101

Table 4: Summary of effects found by G-test and G_H tests on the data sets

to change also. To test this intuition, we compared the sets of dependencies discovered by the analysis across combinations of the experiments, both for which dependencies are found and for the directionality of the dependency (i.e., the dependency reflects an increase or decrease in the incidence of failure). The results are summarized in Table 5. Experiment three is a variant of experiment two, which is a variant of experiment one. Thus, we would expect adjacent experiments (i.e., one and two, and two and three) to be the most similar in terms of their shared dependencies; and non-adjacent experiments (i.e., one and three, and one and two and three) to be least similar, that is, to share few dependencies. In fact, the data exhibits just this phenomenon: experiments one and two had the most dependencies in common; experiments one and three and the full set (one, two and three) had the fewest in common. Some of the implications of the limited overlap across all three experiments will be discussed in the last section.

2.4 Discussion: Shortcomings and Extensions

We have analyzed the failure recovery performance data for dependencies between failures, recovery actions and the failures that immediately follow repairs. Given our original goal, debugging planning failures, this analysis seems limited. In fact, debugging failures in Phoenix requires considerably more work. In this section, we will address some potential shortcomings of this approach and describe how we are extending this analysis.

	Experiment Combinations			
	1&2	2&3	1&3	1,2&3
$A - F^2$	1	1	0	0
$F^1 - F^2$	9	5	2	2
$F^1 A - F^2$	1	1	0	0
Totals	11	7	2	2

Table 5: Number of dependencies shared by combinations of experiments

The algorithm for discovering the dependencies is combinatorial: as the number of factors increases and the number of values for the factors increases, so does the number of potential dependencies and the number of tests to be done. However, this complexity is mitigated by the sparseness of observed transitions in the data (e.g., not all actions are applied successfully to all failures) and by the automation of the analysis. Because transitions are only interesting if they are observed, the only transitions even considered are those that appeared in the data; for the failure recovery data, this filtering reduced the set by three quarters. Additionally, because the analysis is simple and fully automated, the calculations are fast and can be run in a batch. Realistically, the computation time required for this analysis was far shorter than that required to generate the data in the first place (less than five minutes for the analysis, but about a week for the data collection).

A more critical shortcoming of the analysis is that it considered only temporally adjacent failures and actions, and only one at a time. The combinatorial nature of the analysis precludes considering every possible ordering of precursors. However, we don't know how many dependencies we are missing by limiting our attention to $[FA-F]$ triples. We strongly suspect that the incidence of dependencies over longer chains is small. We are working on a new experiment design to selectively eliminate recovery actions from the available set to test whether each causes or avoids particular failures. For example, if we remove an action from the set, say *rp*, and the frequency of *vit* relative to other failures decreases, then we could see whether dependencies in $F^1_x A_{rp} - F^2_{vit}$ triples explain all the surplus F_{vit} failures when A_{rp} is in the set, or whether A_{rp} affects F_{vit} over longer chains.

Our results tell us roughly why changing the design of failure recovery changes the incidence of failure, but do not tell us *how* these effects came about. The analysis described here is the first step of a four step procedure called *failure recovery analysis*: First, we search failure recovery data for statistically significant interactions between recovery efforts and failures. Second, we map these effects to structures in the planner's knowledge base known to be vulnerable to failure: the structures may be sensitive to changes in the environment or intolerant of inadequacies in plan knowledge. Third, from the interactions and vulnerable plan structures, we generate explanations of why the failures occur. Finally, we use the explanations to separate the occasional, acceptable failures from chronic, unacceptable failures, and recommend redesigns of the planner and recovery component.

3 How Uncovering Contributors to Failure Can Lead to Better Planning

Analyses of dependencies can contribute in several ways to our understanding of planner performance. As described, they can identify contributors to failure and assist in the testing and debugging of planners with incomplete or incorrect domain models. Additionally, these tests provide a measure of similarity between test situations. The overlap between adjacent experiments was moderate, but negligible across the entire set. The more the environment and agent changes, the more we expect observed effects to change; thus, we can use dependencies as a kind of similarity measure across planners and environments. If we defined domain-independent categories of failures, we could compare the behavior of different planners on the basis of their failure dependencies.

The most compelling result of our analysis is not any single dependency, but rather, how those effects changed as the experimental situation changed. We changed the selection strategy used by the failure recovery component and the number of dependencies increased; we added two new types of actions, actions that could be applied to only three failure types, and the number of dependencies increased further. Seven of the dependencies introduced in the third experiment are directly attributable to the two new actions. In addition, we found 32 new dependencies and saw 24 dependencies disappear; these must be due at least in part to interactions between the existing architecture and the two new actions. The interactions are either cases in which the new action caused certain types of failures, or in which the new action precluded application of other actions that contributed to particular failures; thus, the new actions can be either causing or inhibiting effects.

The lesson from our analysis is that while design changes rarely have isolated effects, we don't have to give up hope of analyzing the effects. We can track the effects: We make minor changes and havoc ensues, but now we have a way to measure the havoc. Phoenix is an example of a system that can interleave plans in arbitrary ways, as dictated by situation. Debugging its failures by "watching the system" or by predicting all possible execution traces is simply not feasible, but running Phoenix many times and analyzing the data is feasible. If we intend to build large planning systems or if we intend to reuse plans in complex but apparently similar situations, then we should start exploring techniques for debugging failures based on execution traces. The techniques described in this paper isolate indirect, difficult-to-identify effects of design changes and so can assist in debugging new planner designs.

A. G-test Computation

The G-test is similar in form and application to the more familiar chi-square test. The advantage of the G-test over the chi-square test is that it is additive: the test can be applied to subsamples and summed to get the same value as if it had been applied to the whole sample. Our analysis relies on this characteristic to detect subsumed and diluted effects.

The G-test as computed in our analyses is called an *interaction* or *heterogeneity G-test*; it is a form of the G-test used to test goodness of fit to an expected ratio. To explain its computation, we will work through the derivation of the pooled effect given in figure 1A. We wish to know whether the data matches the frequencies of the population: does F^2_{prj} follow F^1_{prj} at the same frequency, relative to other failures F^2, that it follows all failures F^1? The frequencies for F^2_{prj} and $F^2_{\overline{prj}}$ following F^1_{prj} for each A_x are given in table 1. The frequencies for the population are obtained by counting all occurrences of F^2_{prj} and $F_{\overline{prj}}$ for all possible precursors, which yields a population frequency of 215 F^2_{prj} and 726 $F^2_{\overline{prj}}$.

The G-test equation (taken from Sokal and Rohlf [11, page 722]) for comparing the observed with the expected frequencies is:

$$G = 2\left[\sum^{b} f_1 \ln\left(\frac{\sum^b f_1}{\hat{p}}\right) + \sum^{b} f_2 \ln\left(\frac{\sum^b f_2}{\hat{q}}\right) - n \ln n\right]$$

where b is the number of rows in the contingency table, $\sum^b f_1$ is the sum of the first column in the contingency table, $\sum^b f_2$ is the sum of the second column, $\hat{p}$ is the expected fraction of the total for the first column, $\hat{q}$ is the expected fraction of the total for the second column, and $n = \sum^b f_1 + \sum^b f_2$ or the total observations. The intuition behind the test is that we compare the observed frequencies, $\sum^b f_1$, to the expected frequencies, $\frac{\sum^b f_1}{\hat{p}}$, disregarding the overlap $n \ln n$. So for table 1, $b = 4$, $\sum^b f_1 = 116$, $\sum^b f_2 = 126$, $\hat{p} = \frac{215}{215+726}$, $\hat{q} = \frac{726}{215+726}$, and $n = 242$ or:

$$G = 2\left[116 \ln\left(\frac{116}{\frac{215}{215+726}}\right) + 126 \ln\left(\frac{126}{\frac{726}{215+726}}\right) - 242 \ln 242\right] = 72.8$$

To determine whether this value of G is significant, we compare the result to a χ^2 distribution with 1 degree of freedom (number of ratios, two, minus one) and find that it is significant at $p < .001$.

Similarly, we can compute the results for the each row in the table in a similar fashion by comparing them to the same population frequencies. The results of each of these calculations are the G statistics listed in table C in figure 1.

References

[1] Jose A. Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, Minnesota, 1988. American Association for Artificial Intelligence.

[2] R.T. Chien and S. Weissman. Planning and execution in incompletely specified environments. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 160–174, Tiblisi, Georgia, USSR, 1975.

[3] Paul R. Cohen, Michael Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3), Fall 1989.

[4] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[5] K.J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failure. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986. American Association for Artificial Intelligence.

[6] Philip J. Hayes. A representation for robot plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tiblisi, Georgia, USSR, 1975. International Joint Council on Artificial Intelligence.

[7] Adele E. Howe and Paul R. Cohen. Failure recovery: A model and experiments. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 801–808, Anaheim, CA, July 1991.

[8] Subbarao Kambhampati. A theory of plan modification. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pages 176–182, Boston, MA, 1990. American Association for Artificial Intelligence.

[9] James C. Sanborn and James A. Hendler. Dynamic reaction: Controlling behavior in dynamic domains. *International Journal of Artificial Intelligence in Engineering*, 3(2), April 1988.

[10] Reid G. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, Minnesota, 1988.

[11] Robert R. Sokal and F. James Rohlf. *Biometry: The Principles and Practice of Statistics in Biological Research.* W.H. Freeman and Co., New York, second edition, 1981.

[12] Gerald A. Sussman. A computational model of skill acquisition. Technical Report Memo no. AI-TR-297, MIT AI Lab, 1973.