A MIXED-INITIATIVE PLANNING APPROACH TO EXPLORATORY DATA ANALYSIS

A Dissertation Presented

by

ROBERT A. ST. AMANT

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial
fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1996

Department of Computer Science

ACKNOWLEDGMENTS

I have become indebted to many people in many different ways on the long road to a Ph.D.

Bill LaRoe, Mike Webb, Charles Chapoton, and Joe O'Rourke helped me to get started in graduate school, and I wouldn't have been able to take the first step without their encouragement and support.

Much of the breadth of my dissertation is due to the interest and encouragement of the members of my thesis committee. Mike Sutherland acted as my mentor in the sometimes complex world of statistics. Arny Rosenberg set high standards for intellectual rigor, skill in making complex ideas accessible, and interest in empirical computer science. Victor Lesser's eye for detail is wonderfully balanced by a view of the broad horizons of AI research. The depth of his familiarity with AI made our conversations both exciting and challenging.

Paul Cohen, who chaired my committee and advised me through five years of research, shares credit for most of the ideas in this dissertation. When I first came to UMass, Paul explained that entering graduate school is in many ways like entering into an apprenticeship. At the time I didn't realize that this entails serious responsibilities for *both* parties. Now, looking back, it's clear that Paul has been an exceptional mentor. Whatever skills I have in identifying an interesting problem, carrying out a research plan, recognizing and conveying the significance of a result—in short, being a scientist—have arisen from his instruction and guidance.

I've also been aided by two generations of high-caliber graduate students in the Experimental Knowledge Systems Laboratory. The old guard, including Adele Howe, Scott Anderson, Cindy Loiselle, and Paul Silvey, introduced me to the hectic, interactive research life of a grad student. Current members of the EKSL, Marc Atkin,

Dawn Gregory, and Tim Oates, have continued to provide a creative, supportive working environment. Dave Hart and Peggy Weston, who manage the lab, have helped in keeping my thoughts and schedule organized.

Like many computer science dissertations, my 200-odd pages reflect a large software system, AIDE. It would not have been possible to build AIDE without Norm Carver, whose work supported initial prototypes of AIDE. I also relied heavily on the system design skills of David Westbrook and the programming expertise of Matt Schmill. For the evaluation of AIDE I am grateful to Marc Atkin, Lisa Ballesteros, Alan Garvey, Dawn Gregory, David Jensen, Tim Oates, Matt Schmill, Tom Wagner, and David Westbrook, for their willingness in acting as subjects and in advising me how to improve the system.

And because balance (though some would say "juggling") is an essential part of a graduate student's life, I have to thank those who made my time outside the research lab worthwhile. These include my volleyball-playing friends, my beer-brewing/beer-drinking friends, and especially my foreign-travel friends, Bill and Rowena LaRoe and Mike and Jackie Webb. Special thanks go to Martin and Ellen Herbordt, who shared almost every aspect of my graduate student career. They are great friends I can commiserate with during the bad times, celebrate with during the good times, and relax with during the times in between.

Finally, I have been lucky enough to reach this point accompanied by my wonderful wife, Luellen Brochu. We've gone through a great deal together. Describing the ways in which Luellen has supported my efforts and enriched my life would make this section twice as long as it is now.

Thank you all.

ABSTRACT

A MIXED-INITIATIVE PLANNING APPROACH TO EXPLORATORY DATA ANALYSIS

SEPTEMBER 1996

ROBERT A. ST. AMANT

B.S., JOHNS HOPKINS UNIVERSITY

PH.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Paul R. Cohen

Exploratory data analysis (EDA) has come to play an increasingly important role in statistical analysis. Modern computer-based statistics packages contain a rich set of operations, suitable for almost any EDA application. One can fit lines and higher order functions to relationships, identify and describe clusters, transform and reduce data to meet the specific requirements of a domain, among many other possibilities, in seeking to understand patterns in data.

Unfortunately, EDA can be difficult. Conventional statistics packages offer the user hundreds of operations, which must often be combined in lengthy sequences to produce useful results. In addition, the application of these operations often depends on the user's knowledge of what the data mean. In other words, EDA is too large a problem for a human analyst to solve alone, but complete automation of the process is not feasible either because domain-specific knowledge is required.

This dissertation describes an assistant for intelligent data exploration called AIDE. AIDE is mixed-initiative, autonomously pursuing its own goals, but always allowing the user to review and possibly override its decisions. AIDE's design as a knowledge-based planning system allows it to detect and evaluate suggestive features

in the data, identify appropriate strategies for extracting the patterns, apply the strategies incrementally, and combine the results in a coherent whole.

An experimental evaluation compared the performance of human subjects analyzing data with and without AIDE's assistance. Although the subjects worked with AIDE for only a couple of hours, each, it clearly influenced the efficiency and coherence of their explorations. Analysis of the experimental results turned up suggestive evidence that AIDE facilitates data analysis primarily by helping users navigate through the space of relations among variables.

This research provides a novel look at automated support for data analysis. Conventional systems tend to take over the task completely, or rely on the user for every step of the analysis. AIDE's mixed-initiative planning approach provides an alternative in which control changes hands flexibly between the user and the system. This arrangement capitalizes on the strengths of both: the system takes over low-level search and statistical computations, while the user remains responsible for strategic, knowledgeable guidance of the process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

> We have plenty of information technology—what is perhaps needed now is more intelligence technology, to help us make sense of the growing volume of information stored in the form of statistical data, documents, messages, and so on. For example, not many people know that the infamous hole in the ozone layer remained undetected for seven years as a result of infoglut. The hole had in fact been identified by a US weather satellite in 1979, but nobody realised this at the time because the information was buried—along with 3 million other unread tapes—in the archives of the National Records Centre in Washington DC. It was only when British scientists were analysing the data much later in 1986 that the hole in the ozone was first "discovered". — Tom Forester [Forester, 1989, p. 23]

Statistical data analysis plays an important role in empirical scientific research. Faced with questions about complex behavior—of physical systems, biological processes, ecological relationships, even computer systems—our most effective approach is often to observe the phenomena that interest us and try to understand what the data tell us.

Data analysis relies heavily on understanding the underlying processes that generate the data. Sometimes, however, our background knowledge, our theories and models, cannot account for patterns we encounter. Some patterns may be only slight deviations from what we expect to find, but others may be novel and completely unforeseen. In Forester's example, scientists did not initially set out to decide whether a hole exists in the ozone layer. Rather, in this case as in others, suggestive patterns in data can take an analysis in new directions, leading to significant results unrelated to the initial goals of the analysis. This kind of opportunism is characteristic of data exploration.

Exploring data is not easy. The difficulties have led some to describe aspects of the work as data dredging, data mining, data archaeology, even data torturing.[1] Some patterns may be so deeply hidden in the data that they can only be teased out through great effort. Other patterns may be easier to detect but hard to characterize. Unexpected interactions between patterns may be difficult to account for. Data exploration requires skill and often a good deal of insight to produce useful results.

Exploring data is as important as it is difficult. While all results may not be as dramatic or far-reaching as the ozone example, exploration has led to significant (and often unexpected) findings in satellite data analysis [Mallows, 1979], air pollution analysis [Cleveland *et al.*, 1974], medical imaging [Diaconis, 1985], artificial intelligence [Cohen, 1995], and many other scientific domains [Tanur *et al.*, 1972].

The statistical subfield of *exploratory data analysis*, or EDA, was established in the late 1960s by statisticians seeking to understand and extend techniques for describing data.[2] Proponents of EDA include John Tukey, Frederick Mosteller, I. J. Good, and many others. Good views the goal of EDA as facilitating the generation of hypotheses to explain patterns in data [Good, 1983]. In contrast to the view that data analysis should be mainly concerned with hypothesis testing, EDA involves examining a dataset with as few assumptions as is practical, letting interpretation be guided by the data, rather than by preconceptions about its structure. Exploration of a dataset is thus initially guided by general principles rather than specific questions. In his seminal textbook, *Exploratory Data Analysis*, Tukey writes,

> A basic problem about any body of data is to make it more easily and effectively handleable by minds—our minds, her mind, his mind. To this general end:
>
> - anything that makes a simpler description possible makes the description more easily handleable.

[1] Taking an optimistic view, Savage comments, "If you torture the data long enough, eventually it will confess." [Good, 1983, p. 284]

[2] Paul Velleman holds that EDA's philosophical roots go back to Francis Bacon's Novum Organum (1620), while Edward Tufte traces some of its methods to the charts of William Playfair in the late 1700s [Tufte, 1983].

- anything that looks below the previously described surface makes the description more effective.

So we shall always be glad (a) to simplify description and (b) to describe one layer deeper [Tukey, 1977, p.v].

EDA works by simplifying descriptions so that they can be easily handled, and extending descriptions through an incremental deepening process. Drawing on the work of Tukey and others [Velleman and Hoaglin, 1981; Hoaglin *et al.*, 1983], we have developed a categorization of EDA procedures into four classes: description, simplification, deepening, and extension.

Descriptive procedures produce summaries of data. These may be simple summary statistics, such as means and medians. They may be more comprehensive, taking the form of a schematic box plot or a regression line. Description is the generation of results that can be directly interpreted by the user.

A log transform that straightens a curved relationship is an example of *simplification.* Irregularities such as outliers are more easily detected in linear relationships than in nonlinear ones; a change in density can often enhance patterns in data; many statistical operations, even ones as simple as Pearson's correlation coefficient, rely on linearity. A straightening procedure simplifies in that it enhances our observation, manipulation, and evaluation—in a word, our description—of the data.

EDA *deepens* descriptions in detail, accuracy, and precision, in the way a microscope enhances observation by trading a global perspective for local detail. An example is the common practice of examining the residuals of a linear fit. The examination itself is carried out by simplification and description procedures. In this case it can bring to light structure in the data not captured by the line, such as clustering, unequal variance in residuals, or local deviations from linearity.

One further aspect of the process is *extension*, which includes building simple models of data. With descriptive, simplifying, and deepening capabilities, EDA attacks data within a limited scope, building descriptions of single variables, bivariate

relationships, tables, partitions and clusters. Local descriptions can have non-local, often wide-spread implications, however; these are examined by extension procedures. When clusters are observed in the values of a variable x, for example, and similar clusters are observed in y, then an extension procedure prompts the examination of the relationship (x, y).

EDA can be viewed as the intelligent application of these procedures. Descriptions of data are established where applicable, supported by simplification procedures, which facilitate description by transforming data into more appropriate forms, deepening procedures, which refine descriptions to increasing levels of detail, and extension procedures, which incrementally widen the scope of the descriptions. The entire process is sensitive both to patterns in the data and to knowledge about what these patterns may mean.

1.1 The Problem

Early EDA techniques involved pencil-and-paper analysis; rather than poring over columns of numbers, the analyst drew scatter plots and numerical and graphical summaries to gain closer contact with the data. Data analysts bring much more to the table than knowledge about statistical procedures and tests. They are also aware of the meaning of variables, patterns that should be present in the data and those that are novel or surprising, and how to follow suggestive clues to identify and describe these patterns. Interactive handling of the data, even through laborious manual techniques, makes good use of these abilities.

Modern systems for EDA have greatly reduced the manual effort of data exploration. A single mouse click or typed command will generate a histogram for a batch of variable values or a scatter plot and regression line for a relationship. These operations and an enormous array of others are implemented in general statistical computing environments. Further, some tasks are completely automated, relying on machine learning or statistical modeling techniques, so that the analyst only views

a set of results, rather than watching over the entire process. By taking over the more mechanical tasks, a good statistical environment lets the analyst concentrate on selecting the data to examine and deciding how to describe its structure.

Imagine that we are given an unfamiliar dataset and are asked, "What can you tell me about the data?" We may have some notions about what the variable names mean—perhaps it is a census dataset—and how they might be be related to one another, but nothing that we could immediately formalize as a hypothesis test. What can we do? We can explore. For a single variable *density* we might construct a histogram, to get a better idea of its distribution—where its central mass lies, whether it is homogeneous or perhaps falls into clusters, if there are unusual observations or unexpected relationships between the observations. In looking at a relationship between two variables *density* and *area*, we might decide that it is approximately linear, or perhaps that its functional form is given by a particular form, such as $density = k/area$. Both observation of the data and knowledge of context influence our decision. If we fit a function to the data and then examine the residuals, we might find patterns in the data that can be traced to the effects of a third variable, *income*. There may be clusters in the residuals of (*density*, *area*), for example, that correspond to specific groups of values in *income*. Knowledge of context again comes into play. Eventually, step by step, we build a better understanding of the data.

Unfortunately EDA is much more difficult than this description may lead us to believe. Even if we are adept at building descriptions, it can be difficult to decide which relationships, in a dataset of twenty or fifty or two hundred variables, are worth examining in the first place. The descriptions themselves may be complex, incorporating free parameters or involving lengthy transformations of the original data. EDA is an enormous problem, both in the amount of data to be explored and in the complexity of the techniques needed to perform the exploration. This dissertation concentrates on two aspects of the problem.

The first aspect involves the limitations of human cognition. Problems often require that enormous amounts of data be processed, far more than can be managed with direct human supervision. Forester's ozone case is one example; other domains that process massive datasets include astronomy, medical imaging, earth sciences, and molecular biology, to name just a few. Realizing that a pattern in one relationship interacts with a different pattern in one of hundreds of other relationships is often beyond our ability. Exacerbating this problem is that many modern analysis techniques *generate* data, rather than reduce it [Pregibon, 1991] (e.g., Monte Carlo sampling, cross-validation, the bootstrap, and the jackknife). Human analysts can be overwhelmed simply by the amount of data to be explored.

We might think that this problem can be solved by additional computing power. A statistical system, perhaps using AI techniques, might be able to handle the huge amounts of data required, generating descriptions and finding subtle relationships between distantly related variables. Unfortunately, this leads to the second aspect of the EDA problem: such a system has built-in limitations in its lack of contextual knowledge. Subject-matter knowledge is essential in interpreting the importance of results. An automated data-mining program might browse through weather satellite data, finding statistically significant relationships between variables measuring temperature, wind patterns, and so forth, but how is it to know that discovering a hole in the ozone layer has greater implications than, say, discovering that local temperatures change with the seasons?

Work in data exploration tends to address one of these problems without considering the other. Each new version of an interactive statistical package gives the user a bigger, more powerful set of tools—but the user must still directly supervise even the simplest tasks, rather than let them be handled by the system. In contrast, new systems developed in machine learning, knowledge discovery in databases, statistics, and other areas try to solve exploratory problems without involving the user at all.

The former approach regards direct human control of the problem-solving process as sacrosanct, while the latter attempts to automate everything that can be automated, letting the user handle whatever remains [Rouse *et al.*, 1987]. Neither approach is an adequate solution.

1.2 A Proposed Solution

These two aspects of the EDA problem form the central motivation for my research. The goal is to build an automated assistant that helps human analysts explore data. I have chosen the term "assistant", rather than "tool" or "interface", with a specific distinction in mind. Two properties let us call a system an assistant rather than a sophisticated tool or interface. First, an assistant is at least partially autonomous. We can give an assistant general instructions and let it make its own decisions about how to carry them out. Second, an assistant responds to guidance as it works. An automated system will inevitably make mistakes from time to time, so its reasoning process (past decisions as well as current ones) must be available to the user for approval or modification. A responsiveness to the guidance provided by human knowledge of context has been termed "accommodation" [Lubinsky and Pregibon, 1988]. An accommodating system takes advantage of human knowledge to augment its own necessarily limited view of the world. The combination of autonomy and accommodation lets the human data analyst shift some of the routine or search-intensive aspects of exploration to an automated system, without giving up the ability to review and guide the entire process.

I have designed and implemented AIDE, an Assistant for Intelligent Data Exploration, to play this role. AIDE is a knowledge-based planning system that incrementally explores a dataset, guided by user directives and its own evaluation of suggestive indications in the data. Its plan library contains a varied set of strategies for generating and interpreting indications in data, building appropriate descriptions of data, and combining results in a coherent whole. The system is mixed-initiative,

autonomously pursuing high- and low-level goals while still allowing the user to inform or override its decisions.

1.3 Contributions

AIDE's design is founded on a strong, heretofore unrecognized relationship between data exploration and planning. Researchers in machine learning [Biswas *et al.*, 1991; Fisher and Langley, 1986], statistical expert systems [Oldford and Peters, 1986; Thisted, 1986; Lubinsky and Pregibon, 1988], knowledge discovery in databases [Zytkow and Zembowicz, 1993], and other areas have taken advantage of AI techniques by viewing data analysis as a search problem. We can gain further advantage by casting EDA as a planning problem. EDA techniques rely on abstraction, hierarchical problem decomposition, and procedural knowledge: these properties make it amenable to planning.

AIDE is a partial hierarchical planner, a specialized type of planner that can represent sophisticated types of control knowledge. Control knowledge is necessary in the representation of *statistical strategies,* or descriptions of the actions and decisions involved in applying statistical tools to a problem [Hand, 1986]. Representation of statistical strategies has posed a difficult problem for the builders of statistical expert systems. Planning is a novel and general solution.

AIDE's most important contribution is to show that a mixed-initiative approach to data exploration can be effective. AIDE is guided not only by its own evaluation of patterns in the data, but also by user directives. When the system sees a clear path for its actions, it can proceed without substantive guidance from the user; nevertheless the user can take over at any point to direct the exploration toward appropriate context-dependent goals. AIDE's planning representation, in addition to being appropriate for representing statistical strategies, turns out to be well-suited in many ways to mixed-initiative interaction with users as well.

Finally, research with AIDE makes a methodological contribution: evaluation of statistical systems that address tasks like exploration has been relatively limited. Because of the complexity of the exploration task, and because of the need for human involvement, developers of such systems generally concentrate on explaining their design decisions and showing how the system treats specific examples [Pregibon, 1991; Kloesgen, 1992; Brachman *et al.*, 1992; Goldstein and Roth, 1994; Roth *et al.*, 1994]. The effectiveness of these systems is not in question, but our research tries to supply a better means of demonstrating it. Research with AIDE takes on the empirical task of (a) showing that its approach is feasible and (b) determining how and why it is effective. While we have not always been completely successful, our experiments are a step in the right direction.

1.4 A Road Map

In this introduction we have briefly discussed what EDA is, why it is difficult, and why it is important. We have also described how people currently go about exploring data, and proposed a way to improve this process. The remainder of this dissertation will go into greater detail on these points.

We can most easily illustrate EDA techniques through examples. Chapter 2 describes datasets from different domains. We explore each dataset, pointing out patterns of interest, while explaining common exploratory techniques and strategies. This chapter gives a brief, hands-on introduction to EDA techniques and how they are applied.

In Chapter 3 we discuss several systems designed to solve problems that involve exploration. AIDE's design reflects many of the considerations that early researchers have found to be important. In addition, AIDE's design is based on research in planning, statistical analysis, and human-computer interaction. Related work in these other areas is discussed in the appropriate context, in Chapters 5, 6, and 7.

A basic grasp of EDA techniques lets us move to the next issue, which involves bringing an automated system into the process. In Chapter 4 we characterize EDA as a search problem, defining a set of primitive operations that combine to implement a wide range of EDA techniques. Chapter 5 then explains why planning is appropriate for EDA. We focus on partial hierarchical planning, which is well-suited to representing procedural knowledge about EDA. We describe how exploration is carried out by planning operations and discuss the planning algorithm, the plan language, and related planners that strongly influenced the development of the AIDE planner.

Chapter 6 describes AIDE's plans. While much of AIDE's intelligent behavior can be attributed to the planning representation itself, AIDE's performance depends strongly on the specific statistical strategies implemented in the representation. We discuss strategies for fitting lines, straightening curves, clustering, classification, as well as a few simple modeling techniques. We also discuss statistical expert systems and machine learning systems that have addressed problems related to exploration.

In Chapter 7 we bring the user back into the picture, discussing AIDE as an interactive system. We describe the look-and-feel of the system and how it interacts with the user in a detailed example. We give an overview of work in mixed-initiative planning and human-computer interaction, areas that have contributed many of the concepts underlying AIDE's design.

Chapter 8 describes our evaluation of AIDE's performance. The basic issue is simple: users should be able to perform EDA better with AIDE's help than without it. We describe an experiment design that gives us a way of comparing performance under these two conditions. Our experimental results tell us that AIDE does indeed improve exploration; further analysis points out where and why AIDE gives an advantage.

Chapter 9 summarizes the work, pulling all the threads—statistical strategies, planning, and human-computer interaction—together. This chapter ends with a discussion of future work: how the user interface might be improved, how new

strategies might be learned automatically, and potential challenges in extending AIDE to other statistical domains.

Each of the chapters above, except for the conclusion, ends with a brief summary of one or two paragraphs. To gain an overview of this work, one might browse through just these concluding summaries:

CHAPTER 2

EXPLORING DATA

This chapter explores two different datasets to give the flavor of EDA. The first dataset comes from an experiment with PHOENIX, a simulation in which agents interact with their environment in complex ways. The second dataset, NETWORTH, was artificially generated as the source data for an EDA assignment in a statistical methods course. All of the results reported in this chapter can be produced through interaction with AIDE, though most of them were originally generated through a conventional statistical interface. AIDE has been applied to many datasets in addition to these; a partial listing is given in Appendix A.

2.1 PHOENIX

The first dataset is taken from an experiment with PHOENIX, a simulation of forest fires and fire-fighting agents in Yellowstone National Park [Cohen *et al.*, 1989]. The experiment involved setting a fire at a fixed location and specified time, and observing the behavior of the fireboss (the planner) and the bulldozers (the agents that put out the fire). Variability between trials is due to randomly changing wind speed and direction, non-uniform terrain and elevation, and the varying amounts of time agents take in executing primitive tasks. In this experiment forty variables were collected over the course of some 340 PHOENIX trials, including measurements of the wind speed, the outcome (success or failure), the type of plan used, and the number of times the system needed to replan. We became interested in the relationship between the time it takes the planner to put out a fire (Duration) and the amount of fireline built during the trial (FirelineBuilt). Figure 2.1 shows a scatter plot of these two variables.

Clearly the relationship can be partitioned into two subsets: a vertical set of points at zero on the Duration axis and a separate, approximately linear set of points. An examination of other variables shows that the vertical points correspond to trials in which the outcome was `Failure`—PHOENIX was unable to put out the fire. Figure 2.2 shows a row-line plot of Duration values (i.e., all Duration values plotted in sorted order) for failed trials. Duration values are relatively continuous except for three flat areas at 120, 150, and 200. A moment of thought gives the reason for this behavior. When PHOENIX fails, it may in some cases stop the simulation; in other cases, however, PHOENIX may continue to fight a losing battle indefinitely. The experimenters have decided that when a trial runs for 120, 150, or 200 time units it is effectively a failure. These values were instituted as cut-off points. Why the experimenters used three separate cutoff values is a mystery.

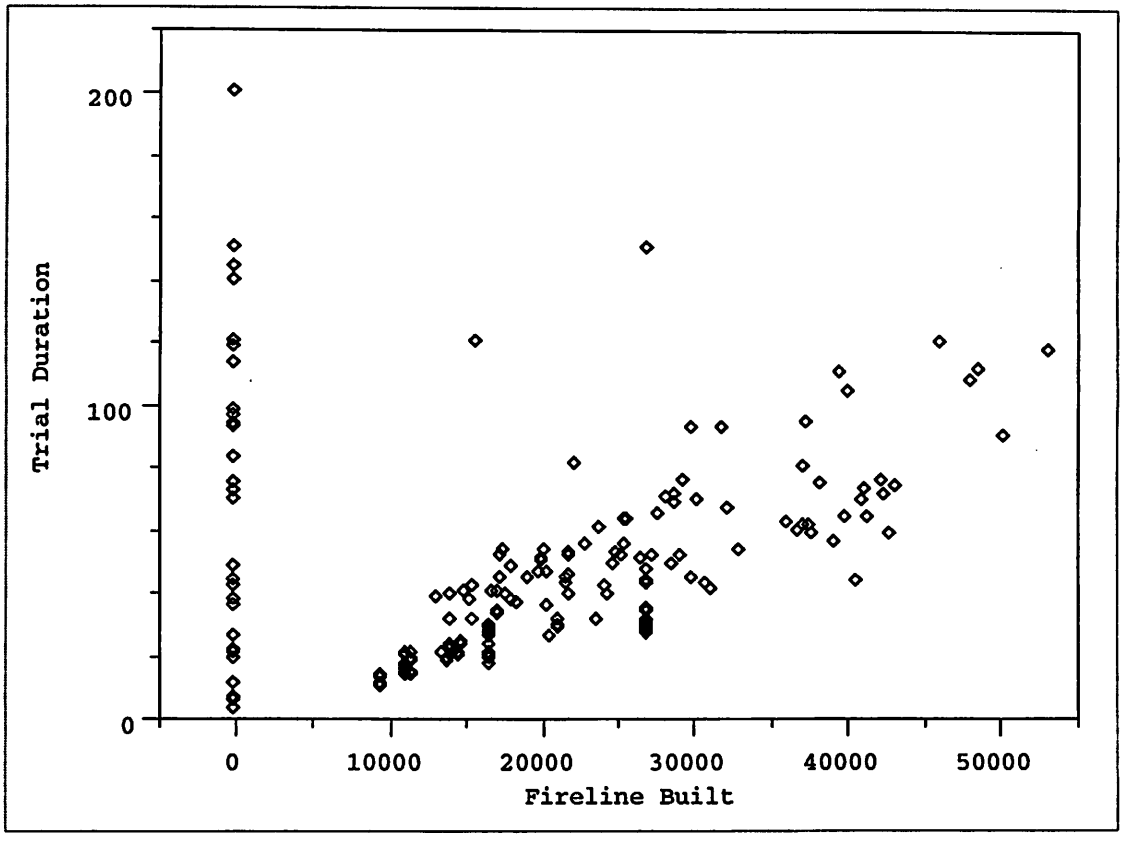


Figure 2.1. FirelineBuilt versus Duration

The correlation of points in the Success partition is positive, as expected from the scatter plot and from our intuitive understanding of the relationship between measures of duration and the expenditure of resources. Two outliers may be important, but are set aside for now.

The "approximately linear" pattern in the Success partition can be more precisely described by a least-squares or resistant line, as shown in Figure 2.3. The residuals of the fit—the degree to which the data are *not* explained by the description—are then generated by subtracting the actual value of Duration, for each value of FirelineBuilt, from the value predicted by the regression line. There are no indications of further structure, such as curvature, that would render the description incorrect, and thus we tentatively accept the linear description.

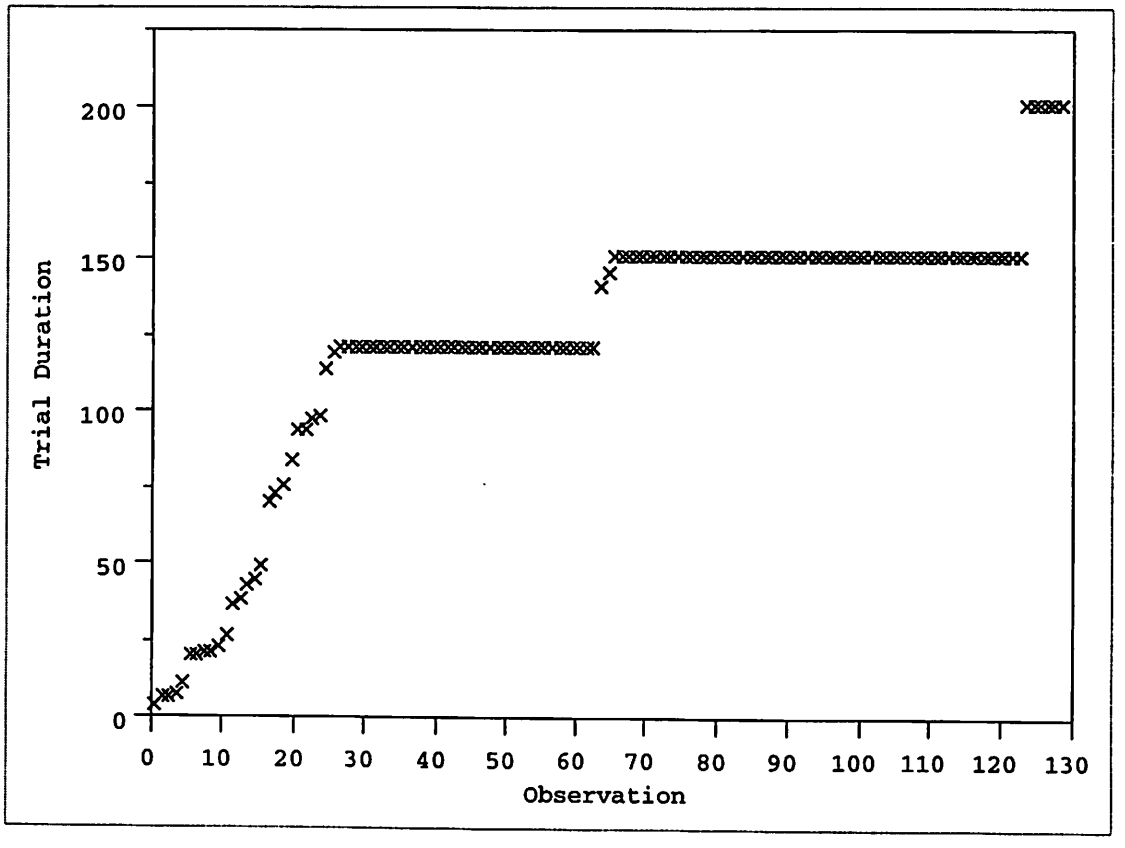


Figure 2.2. Sorted Duration values for failed trials

A closer look at the Success partition shows small, vertical clusters in the lower range of FirelineBuilt. Figure 2.4 highlights these points. One of the outliers, mentioned earlier, belongs to a vertical cluster. In a histogram or kernel density estimate of FirelineBuilt, these would appear as small peaks. A better view of the general pattern of clustering is given by each cluster's central location, its median FirelineBuilt and Duration value. Once the clusters have been reduced to a set of representative values, they can be described in turn. These points also follow a linear pattern, with a slope slightly less than that of the line fitting the entire partition.

Because not all points fall into the vertical clusters, it becomes appropriate to generate a binary variable to encode this difference in behavior. A comparison of this variable with other relevant possibilities (i.e., other discrete experiment variables) shows that the clustered data correspond to trials in which the planner did not need

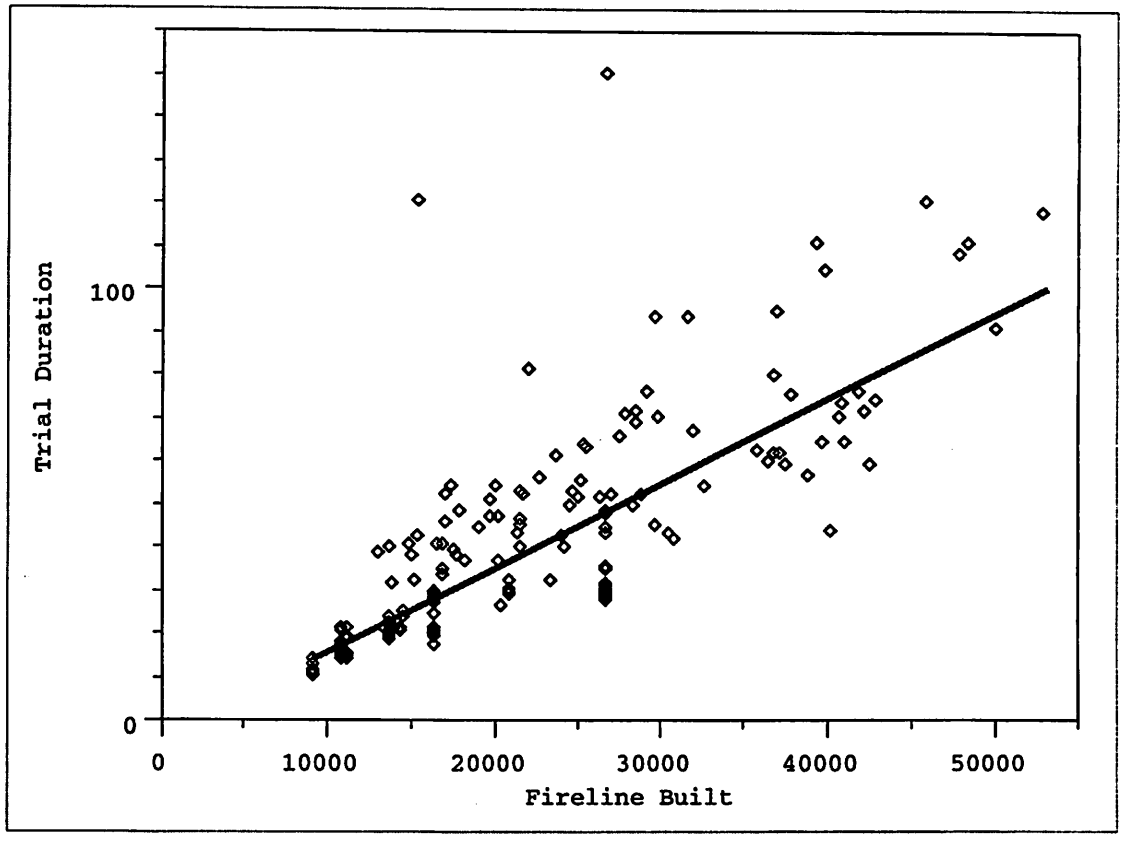


Figure 2.3. Linear fit of FirelineBuilt versus Duration, successful trials

to replan. This is shown in in the contingency table in Table 2.1: observations fall into clusters when # Replans = 0, and usually not otherwise.

A more detailed way to characterize the clusters involves assigning unique identifiers to points in different clusters. Another search through relevant variables produces WindSpeed and PlanType, which together provide moderately good prediction of cluster membership. That is, each cluster corresponds to a different combination of WindSpeed and PlanType values. The experiment varied WindSpeed over three

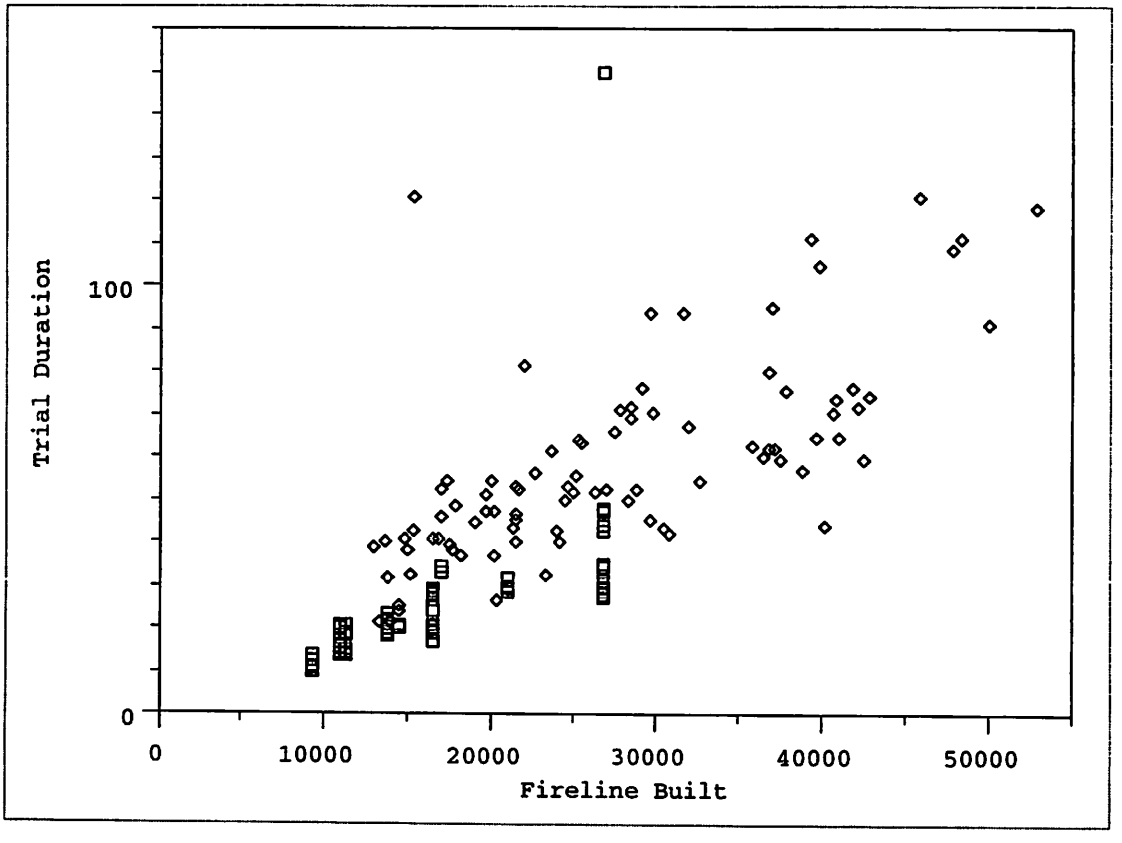


Figure 2.4. Clusters in FirelineBuilt versus Duration, successful trials

Table 2.1. Clustered and non-clustered FirelineBuilt for successful trials

# Replans =	0	1	2	3	5
clustered FirelineBuilt	121	3	0	0	0
non-clustered FirelineBuilt	4	63	20	2	2

distinct values (3, 6, and 9 mph), and there were also three different types of plans applied. Figure 2.5 plots the clustered points, using a different marker for each wind speed. The scale of the axes has been changed to give a more detailed view of the clusters. The triangles are for WindSpeed = 3, X's for WindSpeed = 6, and Z's for WindSpeed = 9. The higher wind speeds correspond to trials that take longer to complete and entail building more fire line. Comparable descriptions of the effectiveness of plans in different conditions can be derived through an analysis of the clusters by PlanType.

Finally, one way to summarize these relationships, given what we have discovered, is shown in Table 2.2. Each cell contains the median Duration value for a different combination of PlanType and WindSpeed values. Interactions can be identified by observing that the row values change at different rates in each column, as do column

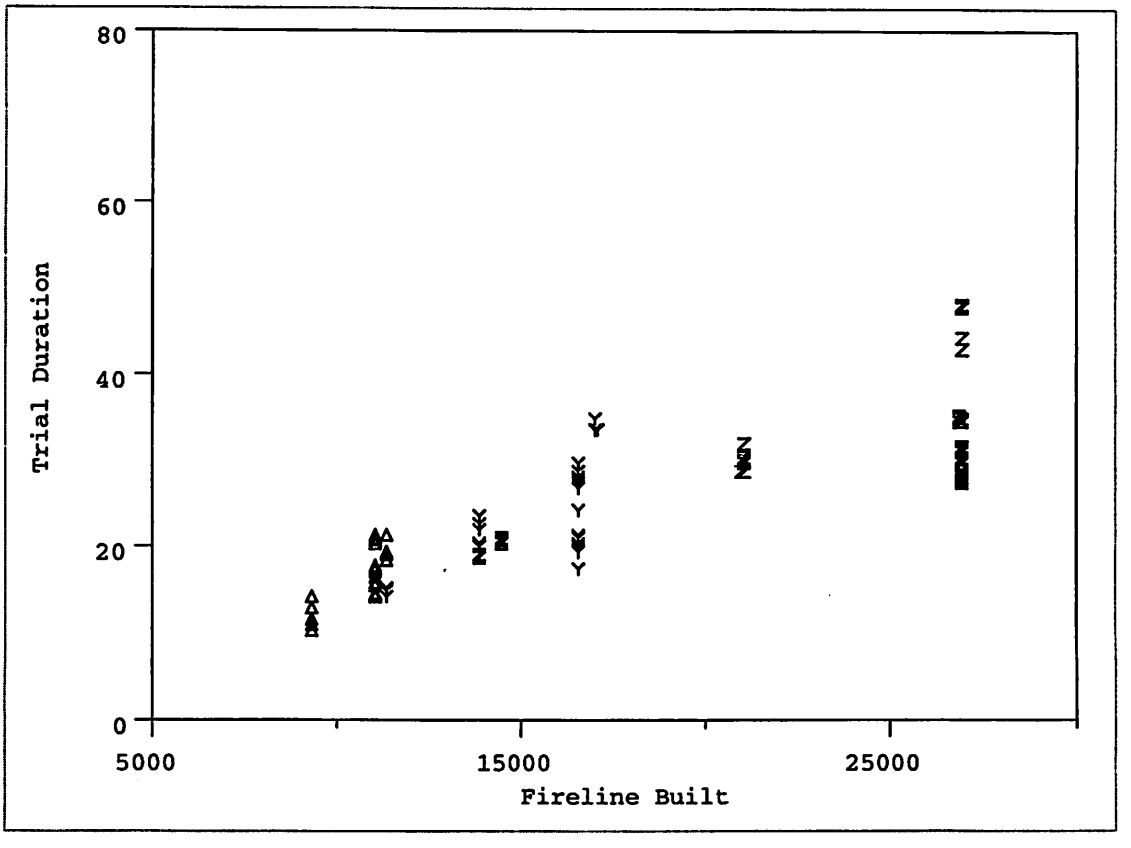


Figure 2.5. Clusters in FirelineBuilt versus Duration, by WindSpeed

values across rows. Possibilities for proceeding include extracting the rows or columns and overlaying their line plots, or applying median-polishing techniques [Hoaglin *et al.*, 1983]. However, we will stop here.

At this point we have quite a refined view of the behavior of the planner as wind speed and plan type change. The final table, Table 2.2, represents the behavior of five different dataset variables:

- WindSpeed (columns);
- PlanType (rows);
- Duration (cells);
- Outcome (table contains only successful trials);
- Replans (table contains only trials that succeeded on the *first* try).

Additionally, there is an implicit dependence on the variable FirelineBuilt; if not for the gap in its distribution, along with the vertical clustering, we might never have pursued this line of exploration.

The results of the exploration include the partition of the relationship, observations about cutoff values in failed trials, observations about the behavior of vertical clusters, and descriptions of all the relationships these patterns have with other variables. These results can be combined in an informal model of the behavior of PHOENIX. Our findings let us draw direct connections between variables such as

Table 2.2. Median Duration by WindSpeed and PlanType for clustered data

WindSpeed =	Low	Medium	High
PlanType = X	11.0	15.5	18.8
PlanType = Y	14.5	26.7	22.1
PlanType = Z	19.5	29.9	29.7

Outcome and FirelineBuilt, links that can be annotated with appropriate descriptions. As we build a model, our context knowledge often suggests that we examine specific relationships and possibilities for describing them. A more detailed account of the application of these procedures to the PHOENIX data is given in *Empirical Methods in Artificial Intelligence* [Cohen, 1995].

2.2 NETWORTH

Dawn Gregory developed the NETWORTH dataset as a homework problem for a course on empirical methods. The advantage of using artificial data is that exploratory findings can be tested against a "true" model.

The data generator works in a straightforward manner. A set of variables is defined, each variable classified as exogenous or endogenous. The values of an exogenous variable are determined by sampling from a specified probability distribution. For example, V_x might be defined as an exogenous variable with a uniform distribution of $U(500)$. The values of an endogenous variables are determined by a functional combination of exogenous and other endogenous variable values, plus a noise function. For example, V_n could be defined as $25V_x + N(0,1)$, where V_x is defined as before and $N(0,1)$ is a standard normal distribution. Variables can be categorical and can rely on conditional tests in computing their values. The variables and their definitions constitute a model. One way to look at a model is as a graph in which nodes correspond to variables and groups of arcs correspond to functional relationships between variables. A single observation is generated by sampling from the distributions of the exogenous variables, and "pushing" these values through the graph of the model, until all values have been generated for all variables. A dataset is generated by repeatedly generating new observations.

The dataset we consider here is the NETWORTH dataset, the direct relationships of which are shown in Figure 2.6. The input to the data generator for this dataset is

given at the end of this chapter in Table 2.3. The patterns that appear in the data are comparable to those in the PHOENIX dataset.

In this section our exploration will take a different tack. In addition to the operations discussed so far, we will apply modeling heuristics to build a model directly from the data, rather than only through the results of the exploration. The descriptive EDA techniques that have served well up to this point have their limitations. With exploratory descriptions we can answer questions such as, "Do low values of variable x correspond to low values of variable y?" and, "Can we reasonably group these observations?" But we eventually will want to ask questions about predictiveness and causation: "If x changes, what will happen to y?" or "What can I do to make the values of y increase?" Modeling techniques can help answer these questions.

Pearl's IC (Inductive Causation) algorithm [Pearl and Verma, 1991] is typical of many causal modeling techniques in the literature. IC begins by assuming that all variables are directly related. From a complete graph of variables, IC iteratively

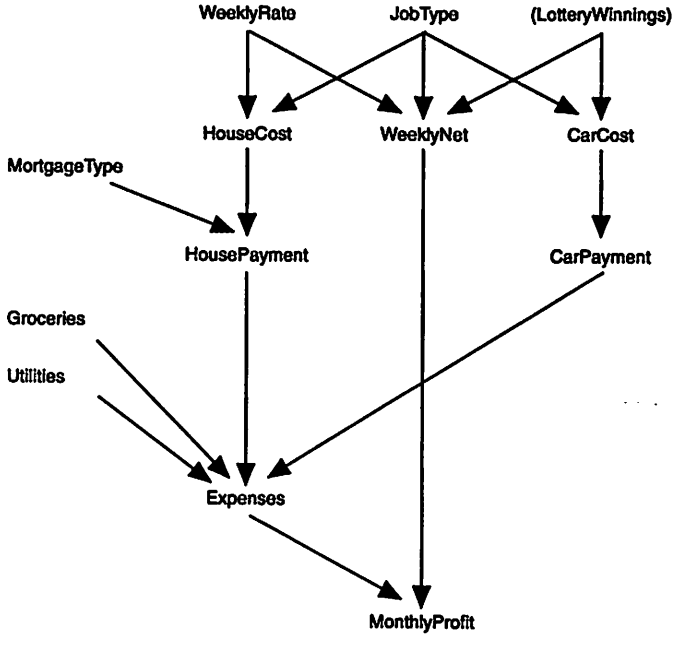


Figure 2.6. Causal relationships in the NETWORTH dataset

removes arcs until only "true," direct relationships remain. The key decision for IC is determining whether variables *a* and *b* are conditionally independent, i.e., whether changes to *a* can affect *b* if we hold some set of other variables constant. IC uses a simple partial correlation test for conditional independence.[1]

By setting IC's various parameters to different levels, we can generate different causal models of the NETWORTH data. For one set of parameter values, the model in Figure 2.7 is generated. IC identifies four relationships, but is unable to determine directionality. While the results of the modeling procedure are correct, as far as they go, they are also disappointing. With different parameter settings, IC can at best identify about half of the direct relationships, but to do this it also identifies many incorrectly. Why are the remaining relationships so difficult to find? As with many modeling algorithms (e.g. standard regression, classification, and clustering

[1]Causal modeling algorithms and heuristics are described in more detail in Section 6.5.1.

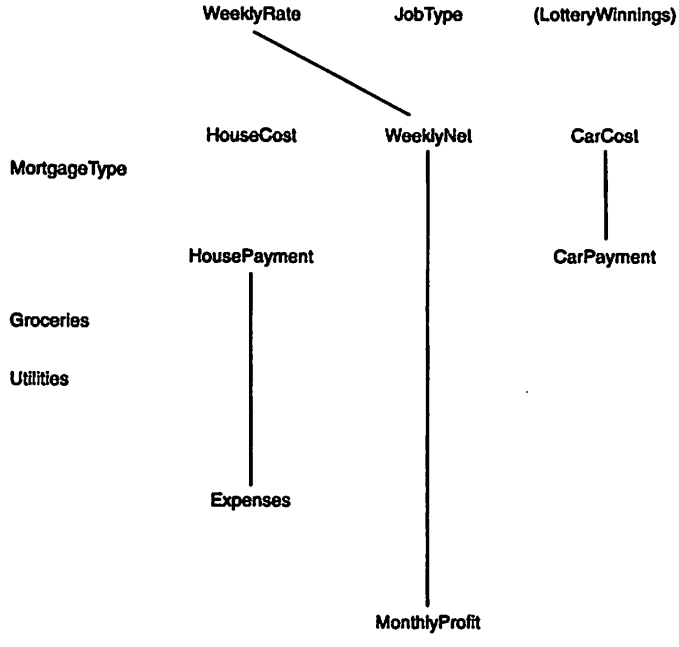


Figure 2.7. NETWORTH causal relationships determined by IC

algorithms), IC relies on assumptions about the types of patterns present in the data; specifically, IC's heuristics are designed to detect conditional independence between variables that are linearly related. One could argue that the modeling algorithm, as implemented, is inadequate, but one could as easily hold that it was improperly applied, that the fault lay in its application to a dataset for which it was not designed.

Exploration techniques can address this problem. Let's take an example, one suggested by our notions about plausible patterns in the data as well as by IC's model—the relationship between the variables WeeklyNet and WeeklyRate.[2] These variables are concerned in some way with income, though it is not immediately clear what they represent. If our intuitions are correct, though, there should be a high, positive correlation between the two factors. The scatter plot in Figure 2.8 shows that this is correct.

The scatter plot also shows that the relationship is approximately linear. The rightmost group of points, though far from the central mass of the data, appears to lie along the line that characterizes the rest of the relationship. A reasonable description of the relationship might then be given by a least-squares line, as in Figure 2.9. The line fits the data very well, with an R^2 of 0.974. However, the residuals in Figure 2.10, show two distinct patterns, or indications. First, note that the residuals do not seem to be evenly distributed around zero on the x-axis, but rather seem to be sloping downward toward the right. Perhaps we should revisit our choice of a least-squares line to consider other possibilities, such as a resistant line, that might be less sensitive to the lack of smoothness in the distribution of the data.

Second, there are clear outliers in the upper part of the plot. In the linear fit of Figure 2.9, these points belong in a group at roughly WeeklyRate = 1000. These observations correspond to people for whom the value of WeeklyNet is proportionally higher than the value of WeeklyRate. Separating these points out and searching for

[2]Thanks to David Jensen for this example, generated during a practice session with AIDE.

other variables in the dataset to account for the pattern shows these to be observations for which JobType is Consultant—in other words, consultants take home more of their weekly income.

This latter result gives us a direct relationship between JobType and WeeklyNet, which can be added to initial causal model generated by IC. The modeling algorithm can then (potentially) use this new information to extend its results further. Even if many of the functional and structural relationships between the variables in NETWORTH turn out to be too subtle for the conditional independence heuristics to capture, exploration can uncover and describe these relationships. This is one example of a common kind of interaction between exploration and modeling. Decisions about which modeling algorithm is most appropriate, about which data are relevant and how the data should be formatted, and about how results should be interpreted generally

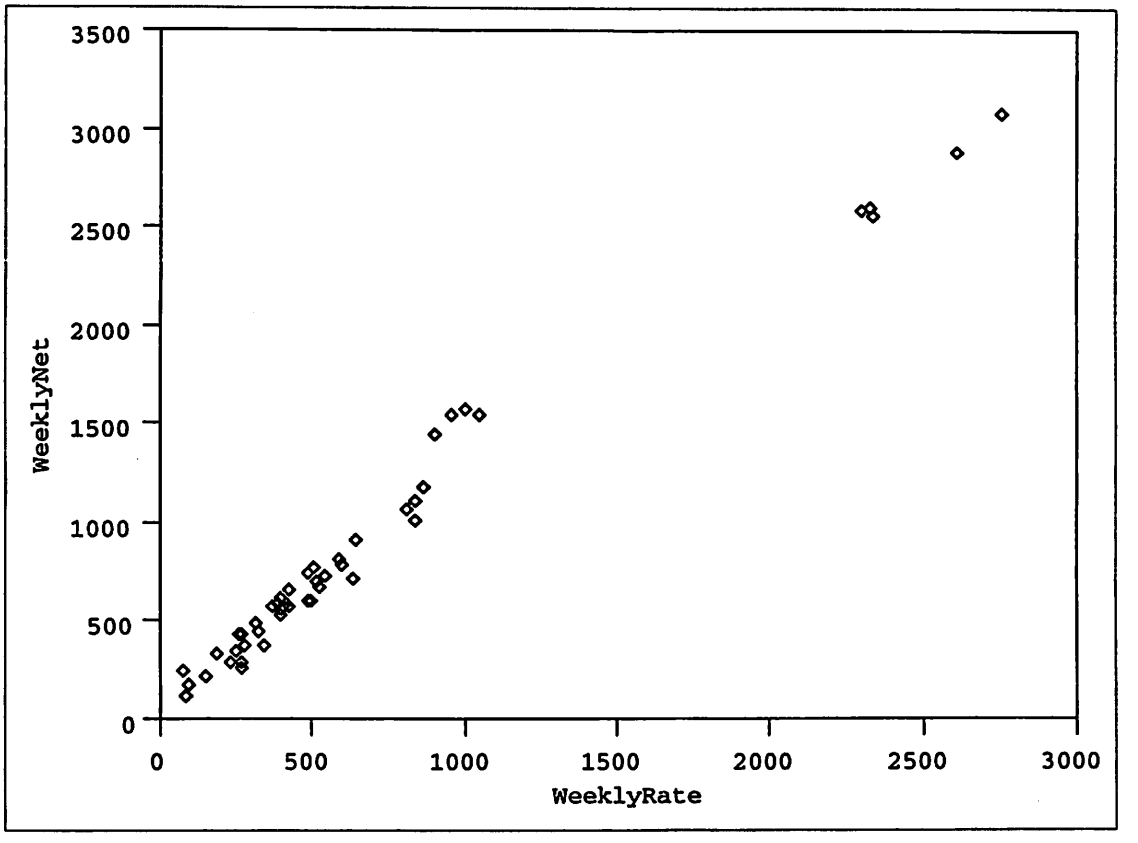


Figure 2.8. WeeklyNet versus WeeklyRate

lie outside the purview of modeling; however, exploration can take responsibility for these activities [Hoaglin *et al.*, 1983].

2.3 Summary

The two datasets explored in this chapter, PHOENIX and NETWORTH, help to illustrate the variety of EDA techniques and the flexible, opportunistic ways in which they are commonly applied. These techniques encompass fitting lines and higher order functions, detecting clusters and identifying variables predictive of clustering, reducing data to enhance patterns, and so forth. Individual results, usually interesting in their own right, can often also act as signposts pointing the way to further results.

EDA techniques are driven both by patterns in the data and by our contextual knowledge. Note the use of *indications* in the analysis. Indications are suggestive

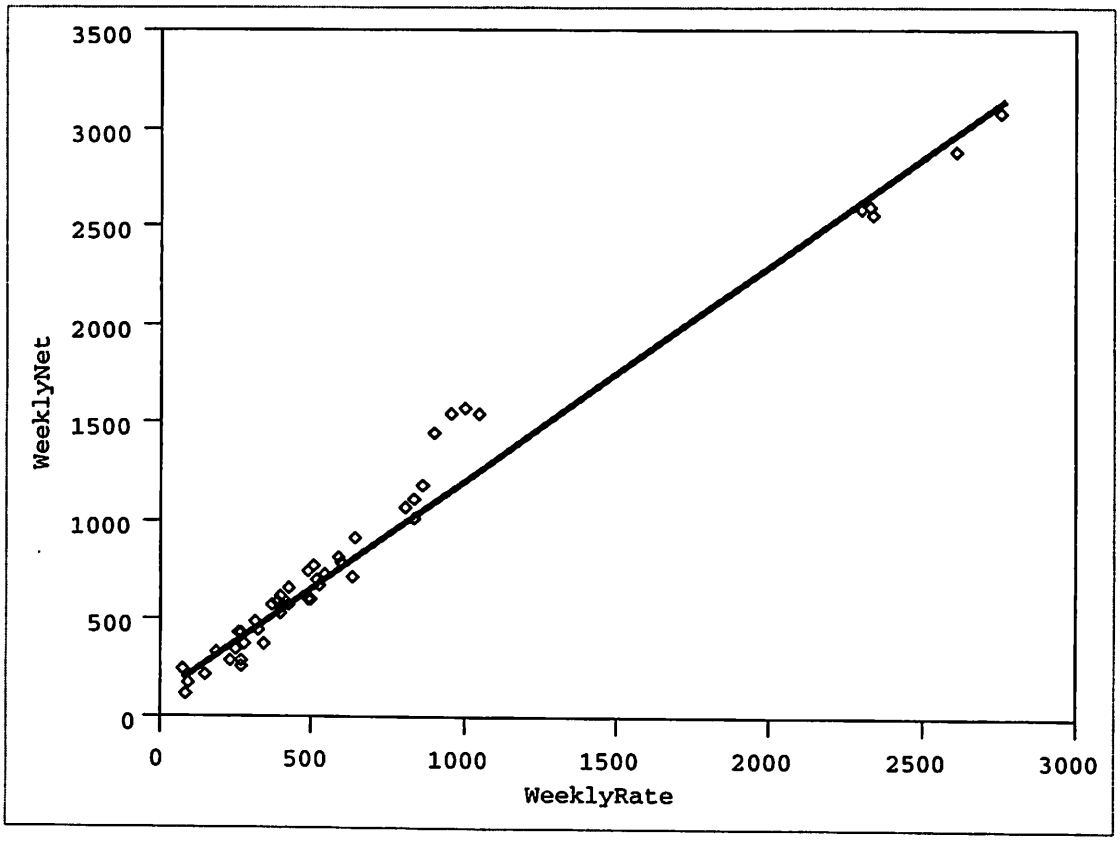


Figure 2.9. Linear fit of WeeklyNet versus WeeklyRate

characteristics of the data, most often involving evaluation of a statistic or descriptive structure [Mosteller and Tukey, 1977]. In the PHOENIX dataset, a gap in the distribution of FirelineBuilt indicates that a partition is an appropriate description; clusters in Fireline Built indicate that another factor may influence its behavior. In general, indications help us move from simple, surface descriptions to more focused descriptions, as we gradually extract more detail from the data.

Contextual knowledge also contributes to the process. Our interest in the relationship (FirelineBuilt, Duration) was prompted by our expectation that patterns in the relationship between measures of duration and the expenditure of resources might prove enlightening. Our interpretation of outliers in the residuals of the functional relationship between WeeklyNet and WeeklyRate was aided by knowledge that JobType can be a relevant factor.

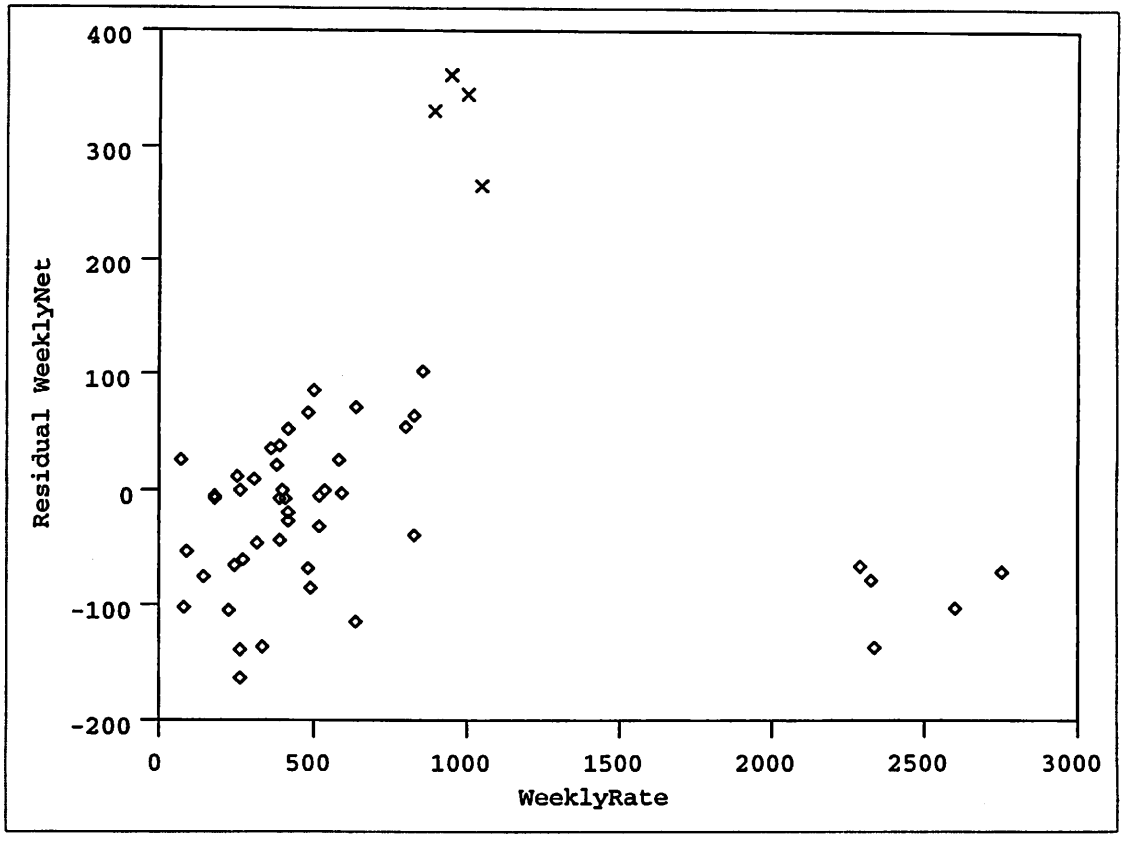


Figure 2.10. WeeklyNet versus WeeklyRate residuals

Thus exploration is partly driven by indications, in bottom-up fashion, and partly by user knowledge and with sensitivity to context, in top-down fashion. These two aspects break the responsibility for EDA into two natural parts. AIDE is responsible for detecting and following up on indications in the data, while the user is responsible for high level strategic guidance of the process. The separation of responsibilities is not strict; AIDE can be trusted with some strategic decision-making, and often the user must work at the detailed level to produce the best results. In general, however, the strategic/tactical distinction describes the respective roles of the user and the system.

Table 2.3. Functional relationships in NETWORTH

Variable	Computation	
LotteryWinnings	Pr = 90/100	→ 0
	Pr = 5/100	→ 20
	Pr = 4/100	→ 100
	Pr = 1/100	→ 1000
JobType	Pr = 5/100	→ `Executive`
	Pr = 15/100	→ `Consultant`
	Pr = 20/100	→ `Engineer`
	Pr = 60/100	→ `Worker`
WeeklyRate	Job = `Executive`	→ 2500
	Job = `1Consultant`	→ 1500
	Job = `Engineer`	→ 500 + $U(50)$
	Job = `Worker`	→ 500
WeeklyNet	Job = `Executive`	→ Lottery + (Rate)(0.9)
	Job = `Consultant`	→ Lottery + (Rate)(0.65)
	Job = `Engineer`	→ Lottery + (Rate)(0.75)
	Job = `Worker`	→ Lottery + (Rate)(0.75)
HouseCost	Job = `Executive`	→ $(\text{Rate}/2)^2$
	Job = `Consultant`	→ $(\text{Rate}/2)^{1.5}$
	Job = `Engineer`	→ 200000
	Job = `Worker`	→ 150 Rate
MortgageTerm	Pr = 2/10	→ `short`
	Pr = 3/10	→ `medium`
	Pr = 5/10	→ `long`
HousePayment	Mortgage = `short`	→ (HouseCost)(1.7/180)
	Mortgage = `medium`	→ (HouseCost)(2.0/240)
	Mortgage = `long`	→ (HouseCost)(2.5/360)
CarCost	Job = `Executive`	→ 50000 + $U(50000)$
	Job = `Consultant`	→ 25000 + $U(15000)$
	Job = `Engineer`	→ 16000
	Job = `Worker` and Pr = 1/4	→ 0
	Job = `Worker` and Pr = 3/4	→ 3000 + $U(7000)$
CarPayment	Pr = 1/3	→ CarCost/24 + CarCost/100
	Pr = 1/3	→ CarCost/48 + CarCost/100
	Pr = 1/3	→ CarCost/60 + CarCost/100
Utilities	50 + U(150)	
Groceries	150 + U(250)	
Expenses	HousePayment + CarPayment + Utilities + Groceries	
MonthlyProfit	(4)(WeeklyNet) - Expenses	

CHAPTER 3

RELATED WORK

AIDE incorporates research from several disparate fields: in statistics, the representation of expert statistical knowledge for EDA; in artificial intelligence, partial hierarchical planning, mixed-initiative planning, and areas of machine learning; in human-computer interaction, collaborative systems for data exploration. Because of this, and because of the generality of the task addressed by AIDE, it is difficult to draw a unified picture of related work.

We can nevertheless identify several systems that cross the boundaries between research areas, systems that have in many ways influenced AIDE's design and development. They are distinguished by their emphasis on the following points:

- Exploration is a coherent process of related actions, in which results are produced incrementally and often opportunistically.
- Some exploratory activities can be automated.
- Evaluation of exploratory findings must involve human judgment.

The systems also attempt to give users a general set of statistical or analytical tools, using only weak domain knowledge. These characteristics rule out autonomous machine learning systems, user-driven statistical programming environments, domain-specific visualization systems, and most of the tightly targeted systems common in the knowledge discovery in databases literature.

Much of our discussion of related work will be deferred until later chapters, where relevant details can be fit into the appropriate context.

3.1 REX

REX, the Regression EXpert system, advises a user in regression analysis problems [Gale, 1986]. It generates a regression for a dataset—something almost any statistical package can do—but then extends the analysis by testing assumptions and suggesting solutions when assumptions are violated. REX performs its analysis interactively, producing graphical displays and textual explanations, giving justifications for its actions, and halting the analysis when a decision based on external context is required. REX was designed as a front end to a statistics system, playing the role of intermediary between the user and an underlying package of statistical functions.

The contribution of REX lies in its implementation of an explicit statistical strategy. Such representations are almost completely lacking in the statistical literature [Lubinsky and Pregibon, 1988]. The strategy is encoded as a hierarchy of frames, with relevant rules associated with each frame—in effect, as a static decision tree. As REX executes each step in its analysis, it tests the data to ensure that assumptions are met and that results are consistent with the assumptions. These tests guide REX along a path down the tree. If the rules indicate a problem, REX generates a set of possible ways to deal with it, and asks the user to choose between them. When the data are clean, interaction is minimal; otherwise the user is queried to address each problem. Without guidance, REX does not continue.

In a retrospective of the work [Pregibon, 1991], Daryl Pregibon made these observations: First, REX's interaction with the user was flexible, depending on the data. If there were no problems with the data (e.g., no outliers, no curvature, no skew, approximate linearity) REX could proceed without help from the user. Only on reaching a problem would REX stop to ask for help. This stands in strong contrast to the usual menu-based or programming interaction users ordinarily face.

Second, users differed in their acceptance of REX's analysis. Non-statisticians were entirely willing to let REX proceed on its own, accepting its answers, even learn-

ing from its explanations. Statisticians, on the other hand, sometimes disagreed with REX's decisions, and some actually disliked the system. While REX's interactions with the user depended on the data, it always made the same decisions at each point. REX was thus not responsive to any special knowledge the user might have that could guide the analysis in one direction or another.

Third, REX was not successful as an environment in which statisticians might develop their own strategies. Formulating strategies is a difficult task, whether or not it is supported by an appropriate computational environment. We can partially attribute REX's problems in this area to its rule-based design—it is often difficult to represent procedural knowledge in rule form.

REX was an ambitious system. It was built not only to automate difficult statistical procedures, but also to provide a kind of playground for statisticians, an environment in which they could explore their own analysis strategies and perhaps develop new ones. REX was significant as an exploratory prototype, whose design decisions could be evaluated for their use in future systems.

3.2 TESS

TESS, the Tree-based Environment for Statistical Strategy, is a successor to REX and the related Student system [Gale, 1986]. TESS produces hierarchies of data descriptions in a semi-automated way [Lubinsky and Pregibon, 1988]. TESS views human knowledge of context as essential to the task. Thus rather than incorporating a limited, machine-based representation of context into its processing, it "accommodates context" by providing for close user interaction in deciding which paths should be explored. From the sets of data descriptions TESS generates, the user can select the ones most appropriate for the analysis.

TESS casts the description of data as a search problem, where description is a process of generating an explanatory hierarchical decomposition of data. A single description of a dataset is a path through the resulting forest of trees. Descriptions

are evaluated on two counts: accuracy and parsimony. The accuracy measure in TESS is based on Mallows's theory of data description [Mallows, 1983], in which the accuracy of a description of a dataset y is measured by counting the number of datasets that are similar to y that produce an identical description. The parsimony measure is directly obtained from the hierarchical representation of descriptions. The most accurate description of a bivariate relationship, for example, is a list of all its data points. This, the root node in a tree of descriptions of the dataset, is naturally the least parsimonious description possible. The relationship might be further described as bivariate normal in one subtree, its variables individually described in further detail in related subtrees, residuals from a linear fit described in still another subtree, and so forth. At the leaves we find reductions to single statistics, which give the most parsimonious but also the least accurate descriptions of the relationship. Good descriptions trade off accuracy and parsimony, in the way estimators trade off variance and bias.

TESS's expertise lies in the handling of data manipulation operations in a regression setting; problems of variable selection, which is strongly affected by context-sensitive goals, remain the user's responsibility. Nevertheless in TESS we see the first elements of a mixed-initiative approach to data analysis. To produce its descriptions, TESS uses a pre-determined set of search control mechanisms, which can be selected by the user. For example, the user can say, "Find the best description using a search with a maximum depth of six," or "Find the ten best descriptions possible within the next five minutes." The general idea is that TESS's search will result in a set of high accuracy, high parsimony descriptions, which are given to the user. The user can select among these intermediate results and cause the system to extend the search from that point. In other words, the user, relying on external knowledge of context, can tell TESS to refocus its attention on different areas of the search space. Control is thus shared, in a limited but important way, between the user and the system.

3.3 IDES

IDES, the Interactive Data Exploration System, is one component of SAGETOOLS [Goldstein and Roth, 1994; Roth *et al.*, 1994]. SAGETOOLS treats data exploration as a problem to be solved by heuristic knowledge, and relies heavily on human judgment in its processing. The system grew out of research in automatic presentation systems, which are intended to relieve users of the need for graphical design and display knowledge. Exploring a dataset in the SAGETOOLS environment, the user can concentrate on relationships and patterns in the data, rather than graphical presentation details.

SAGETOOLS breaks data exploration techniques into three categories: data visualization techniques, which include designing and generating effective graphical displays; data manipulation techniques, which include selecting data, focusing on particular attributes, and grouping observations; and data analysis techniques, which are the standard tools of statistical testing and analysis. These categories are interdependent and somewhat overlapping.

IDES is concerned with data manipulation. Its implementation concentrates on data in the form of objects, such as cars or houses, with attributes, such as asking price, age, size, and so forth. In IDES, data manipulation techniques fall into three general types. The first controls the scope of the data, restricting or expanding the amount of data one wishes to view. One might change the scope in a housing dataset by specifying, for example, that only `house` objects with `asking-price` greater than $100,000 should be considered. One might also merge subsets to expand the scope. The second type of technique involves choosing the level of detail of the data, or changing the granularity of the data. This is managed by grouping and reducing numerical-valued properties of a set of objects. For example, we might group houses in our dataset by `neighborhood`, aggregate their `asking-price` values, and reduce them to derive a mean or median `asking-price` for houses in each neighborhood.

The third technique involves selecting the focus of attention. Selecting the focus, in the simplest case, involves choosing the attributes of the dataset one wishes to manipulate with other operations. For example, one might focus on the attributes `asking-price` and `neighborhood` to gain some understanding of the relationship between them. Deriving new attributes from old ones is also considered a form of attention focusing.

IDES provides a powerful interactive environment for data exploration. In contrast to REX and TESS, IDES concentrates on data presentation, rather than statistical analysis. The result is an environment that supports flexible data manipulation with immediate, sophisticated graphical feedback. While IDES displays far less autonomy than REX and TESS, it can be effective in solving many kinds of data exploration problems.

3.4 IMACS

IMACS, the Interactive Market Analysis and Classification System, is aimed at the task of "data archaeology" [Brachman *et al.*, 1992]. Data archaeology is distinct from data mining, in which an autonomous statistical or machine learning algorithm searches a large database for implicit patterns. Data archaeology recognizes that results do not emerge in a single pass over the data, but rather evolve in an iterative process that requires constant human interaction. IMACS aims to improve current data analysis technology by increasing the flexibility of data representation, improving complex and error-prone access methods, supporting iterative exploration, and managing work over time. The prototypical IMACS application is a large commercial database, such as a department store's customer information database.

Representational flexibility is provided by the formal knowledge representation language Classic [Brachman *et al.*, 1990]. Classic contains three kinds of objects: individuals, which represent objects in the domain of interest; concepts, which are potentially complex descriptions of individuals; and roles, which are two-place predicates

relating individuals. The representation gives users more flexibility than provided by a relational database representation. Because datasets must be mapped into the Classic representation to be accessible to IMACS, however, a significant part of an IMACS application is the definition and generation of a domain knowledge base.

IMACS gives the user a variety of ways to access and manipulate data. A query language allows direct, SQL-style queries, augmented to take advantage of the knowledge representation features of Classic. The user may also create templates, or forms, to represent common queries, which can then be filled in appropriately for different situations. The user can also interact directly with a graphical or tabular display of the data.

IMACS supports a particular type of exploration, that of segmenting data into interesting subsets. In commercial databases one can often find patterns hidden in large, pre-determined categories; for example, the set of all customers includes those who buy expensive items only during sales. Its access and manipulation methods are geared toward identifying interesting patterns in subsets, aiding segmentation, and defining Classic concepts to record significant new groupings of objects.

The contribution of IMACS is the recognition that data archaeology (and by extension data exploration) is an iterative process that requires human guidance, and that AI knowledge representation techniques can significantly extend the conventional data analysis environment. IMACS (like IDES) is somewhat limited in its view of data as relationships between objects with attributes as well as in its view of exploration as segmenting data. IMACS's strength lies in its conceptual framework rather than its internal processing; again like IDES, IMACS generally displays little autonomy.

IMACS is a representative example of several systems for knowledge discovery in databases, including IDEA [Kellogg and Livezey, 1992], The Knowledge Discovery Workbench [Matheus *et al.*, 1993], and RECON [Simoudis *et al.*, 1994]. These systems are comparable to IMACS in their goals, scope, and limitations, and we will not discuss them further.

3.5 Explora

The Explora system [Kloesgen, 1992; Kloesgen, 1993] searches for statistical patterns in numerical data. Explora provides a large, fixed set of pattern templates, which the user fills in to create hypotheses. The system attempts to verify these hypotheses by evaluating their strength or coverage in the data.

Explora concentrates largely on finding subsets of a dataset for which specific distributional properties hold. For example, a pattern that Explora might judge to be interesting is "a subset of variable x in which the mean is significantly smaller or larger than in the entire population." The user can specify constraints on the variables considered and the range of data examined.

Given these user constraints, Explora generates a set of significant patterns, which it evaluates with various heuristics and statistical tests. Rules for reduction, generalization, and selection then process the patterns to generate results that are presented to the user. The user may at this point revise the current search constraints and activate further exploration by the system.

Explora has many of the characteristics we require of a system for exploration. It provides a reasonable set of pattern templates, a variety of search techniques, and mechanisms for the user to influence its processing. It supports the incremental, opportunistic EDA style. As with IMACS and IDES, Explora adopts a question-answer style of user interaction, requiring a good deal of user input concerning the types of patterns, statistical tests, and search strategies it should consider in its processing.

3.6 Other Systems

Exploration is addressed in one form or another by a great many systems in machine learning, scientific discovery, and knowledge discovery in databases. Few address the issues we have tackled in AIDE, however.

Machine learning systems are autonomous by nature. They often perform tasks similar to that of AIDE, but without accounting for the knowledge-rich, opportunistic

aspects of the process. Conceptual clustering systems (e.g., COBWEB [Fisher, 1987], CLASSIT [Gennari *et al.*, 1989], ITERATE [Biswas *et al.*, 1991]) attempt to extract plausible groups or hierarchies of groups from data. Classification systems (e.g., ID3 and C4.5 [Quinlan, 1993]) try to predict classes of one variable based on the values of other variables. Constructive induction systems [Fawcett, 1993] generate new variables, which are viewed as concepts, to better explain patterns in data. Traditional scientific discovery systems (e.g., BACON [Langley *et al.*, 1987], ABACUS [Falkenhainer and Michalski, 1986], E* [Schaffer, 1990]) attempt to generate theories, or functional relationships, from data. All these systems are limited both in the types of patterns they can identify and the kinds of input data they can process. For example, E*, the most successful function-finding system of its kind, can discover only six classes of functional relationships. Its output is simply an equation, dispensing with observations of outlying values, skewed variables, and other potentially significant properties of the data. Modern classification algorithms are sophisticated and robust, but it is not clear how they would fare given, say, a simple regression problem. Constructive induction and clustering systems deal mainly with categorical variables, which imposes strong constraints on the types of structure that can be found [Fisher and Langley, 1986].

Matheus et al. [Matheus *et al.*, 1993], in a comprehensive review of techniques for knowledge discovery in databases, raise a philosophical issue: they assert that developers of KDD systems face an inevitable tradeoff between autonomy and versatility. In their eyes, the ideal system would be able to handle all of the data access tasks, the selection of analysis procedures, and the evaluation of results entirely autonomously, while still being applicable across many domains. Most existing systems tend to fall near one endpoint or the other in this tradeoff. Unfortunately, this perspective misses the point that human involvement is an essential part of the exploratory process. A completely autonomous system can have little notion of the significance of its

findings—but this is exactly the kind of knowledge that informs the selection of data, analysis methods, and evaluation techniques.

Machine learning and KDD systems are very good at performing the specific tasks for which they have been designed; AIDE can be viewed, from one perspective, as an attempt to incorporate their techniques in a more comprehensive framework for exploration. Due to time and effort constraints, AIDE can only implement a representative sampling of the various possibilities. It is enough nevertheless to suggest the promise of the approach.

3.7 Summary

The systems discussed in this chapter, REX, TESS, IDES, IMACS, and Explora, share a view of data exploration as a coherent, incremental process, amenable only to partial automation. Brachman's description of IMACS as a system for *data archaeology* (in contrast to data mining) is insightful: results do not emerge in a single pass over the data, but rather evolve in an iterative process that requires constant human interaction.

These systems are similar to AIDE in several ways. They are aimed at difficult data analysis tasks related to exploration; they concentrate on giving a human analyst appropriate tools, rather than providing black box solutions; judgments about the significance or interestingness of results is left in the hands of the analyst. Some systems (IDES and IMACS) even use a representation of primitive operations that is functionally equivalent to AIDE's. Nevertheless, except for TESS, none of the systems concentrates on maintaining a flexible balance between autonomy and accommodation. As discussed in the next chapters, this balance is central to AIDE's processing.

CHAPTER 4

FORMULATING THE EDA PROBLEM

EDA can be difficult. For example, as part of the evaluation of AIDE, datasets were generated to test the system's performance. Checking whether the generated patterns could be reconstructed in practice took several hours, even with the knowledge of which patterns to expect. EDA techniques are not difficult to apply; however, there are a great many of them, they are heavily parameterized, and they can often be reasonably applied to any subset of the data. From a problem-solving standpoint, EDA is difficult because it involves search.

Consider the analysis of the PHOENIX data in Section 2.1. To derive the summary of median Duration by WindSpeed and PlanType, we needed to make these decisions:

- Select relationship (FirelineBuilt, Duration).
- Observe indication of gap in FirelineBuilt distribution.
- Search dataset; find variable Outcome.
- Partition (FirelineBuilt, Duration) by Outcome.
- Select partition of successes.
- Select description operation: linear fit.
- Select linear fit operation: resistant fit.
- Explore resistant fit...
- *Backtrack.*
- Observe vertical clusters.
- Search dataset; find variable # Replans.

- Partition (FirelineBuilt, Duration) [success partition] by # Replans.
- *Backtrack.*
- Search dataset; find variables WindSpeed and PlanType.
- Partition (FirelineBuilt, Duration) [success partition] by WindSpeed and PlanType.
- Select statistic to describe clusters: median.
- (Apply statistic.)
- ...

This sequence can be viewed as a path through a relatively small decision tree, going about ten levels deep. In many situations EDA decision trees will be much deeper, and very branchy. Now imagine a user generating this analysis with the help of a statistical user interface for data exploration. Even the simplest packages provide on the order of fifty operations, while the more sophisticated packages provide hundreds. This informal characterization gives us a search space containing on the order of 10^{20} states that we need to consider in constructing one partial result, the table of medians.

The search space for the exploration of the entire dataset is enormous. Fortunately, reformulation can reduce the search space. Our reformulation will be based on a new set of operations discussed in the next section.

4.1 EDA Primitives

We define three types of basic operations: reduction, transformation, and decomposition. These operations are second-order functions that take functions and data as input. They constitute a language for manipulating exploratory structures, in the same sense that the relational algebra constitutes a language for manipulation of relations in a database [Stemple and Sheard, 1989].

We can think of these operations as analogous to functions that operate on matrices. The determinant of a matrix generates a scalar value from an $m \times m$ matrix:

$$\det \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}.$$

The multiplication of an $n \times m$ matrix by a scalar generates another $n \times m$ matrix:

$$a \cdot \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} ax_{11} & ax_{12} \\ ax_{21} & ax_{22} \end{bmatrix}.$$

The "splitting" of a $n \times m$ matrix can result in several smaller matrices that if merged would contain all the original data:

$$\text{split} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} \text{and} \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix},$$

or

$$\text{split} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \text{and} \begin{bmatrix} x_{12} & x_{13} \\ x_{22} & x_{23} \end{bmatrix}.$$

These operations are analogous to reductions, transformations, and decompositions, respectively, in the way that they alter the structure of the input data to generate a result. A reduction generates a scalar value from a variable or multivariate relationship (the determinant); a transformation generates a variable or relationship of the same structural form as its input (scalar/matrix multiplication); a decomposition generates a new set of variables or relationships, which contain subsets of the original data (matrix splitting).

These operations act on datasets represented as relations, a common approach in the literature of statistical software [Chambers, 1977]. Variables correspond to relation attributes, while observations correspond to relation tuples. Bivariate and multivariate relationships are relations with two or more attributes. A variable may be considered either an attribute of a relation or a relation consisting of a single attribute. In contrast to the usual relational database conventions, the domain of an attribute may contain other relations as well as atomic values.

Reductions map relations to atomic elements. A reduction takes a function and a relation as input. Computing the mean of a sequence of numbers is an example of a reduction of a relation whose single attribute has a numeric domain. Letter values (i.e., medians, fourths, eighths, etc. [Tukey, 1977]) are reductions, as is the correlation between two sequences of numbers. All statistical summaries of this type fall into the class of reduction operations. In this dissertation we use the notation `(reduce (function mean) ?sequence ?output)` where `?sequence` is a batch of numbers and `(function mean)` is the function called on the data.

A *transformation* operation maps a function over the tuples of a relation. A transformation takes a function f and a relation R as input. The operation generates a new relation S, such that for each tuple r_i over the attributes in R, S contains a corresponding tuple $s_i = f(r_i)$. Taking logs of a batch of numbers is a simple example of transformation: `(transform (function log) ?sequence)`.

A *decomposition* operation breaks a dataset down into smaller datasets. A decomposition takes a relationship R and a mapping function M as input. The mapping function is applied to individual tuples and returns a subset of $1, \ldots, k$ for each tuple. This set of integers can be viewed as a set of assignments of the tuple to a newly generated relation. The decomposition of R by M generates a new relation S, of cardinality k, with a single attribute whose values are new relations themselves. The contents of the ith relation in S are those tuples r_j in R such that $M(r_j) = i$. Decompositions may partition a dataset or generate overlapping subsets. Separating a dataset into clusters is a decomposition, as is isolating outliers in a relationship. For decompositions we write `(decompose (function mapping) ?dataset ?output)`. Successive decompositions of a dataset leads to an implicit tree of increasingly refined subsets of the data.

This conceptual breakdown of EDA operations is comparable that of IDES [Goldstein and Roth, 1994], IMACS [Brachman *et al.*, 1992], and related systems, but differs in the details. Controlling the scope of data in IDES corresponds to decomposing

a dataset and selecting one subset in AIDE. Choosing the level of detail in IDES means decomposing a dataset, transforming each partition by applying a reduction, and potentially further transforming the result. Selecting the focus of attention in IDES is a higher level activity in AIDE. The mapping of IMACS concepts to AIDE representational constructs is similarly direct.

The effectiveness of the IDES and IMACS representation depends strongly on their view of data as objects with attributes. AIDE takes a less restrictive (i.e. conceptually weaker) view, which gives more flexibility, especially for bottom-up processing. For example, in IDES's object-based representation it is easy to say, "Treat houses with the property Cost $> x$ as a separate group; call them 'expensive houses'." This is possible to do in AIDE, but without built-in support for the representation of typed objects. On the other hand, in AIDE we can say, "Partition observations into ten equally-spaced bins," and then manipulate the resulting relation of subsets. In IDES's object-based representation we would create ten new objects representing groups of observations, perhaps calling them "Bin-1 objects" through "Bin-10 objects," plus another object grouping the ten new objects, and then we would manipulate this aggregate object. The object-based approach can grow cumbersome when data manipulation results in groups of observations and values that have no intuitive interpretation as objects. For our purposes, AIDE's more flexible representation is preferable.

4.2 Applying the Primitives

Combinations of these three types of operations capture most, if not all, common exploratory procedures. More importantly, by representing procedures in terms of these types of operations, shared structure in the procedures becomes apparent.

Consider a simple procedure for building a histogram, for a discrete-valued variable x: divide the range of the variable into its unique values; break the variable into subsets, one subset per unique value; count the number of elements in each

subset. The resulting counts are the bar heights of the histogram, each bar associated with a different value of x. In other words, we `decompose` the variable, which generates a new relation with an attribute that contains the subsets. We then apply a `transformation`, with an embedded `reduction`, which maps each subset relation to a single value, the "count" statistic of the subset. Now consider a procedure for building a contingency table for a relationship between x and some other discrete-valued variable y. We divide the relationship into subsets, one corresponding to each unique combination of x and y values. We record the number of observations in each subset, associating each count with a different x, y combination. The resulting values are the cell counts for the contingency table. A contingency table can thus be seen as a two-dimensional analog of a histogram.

Such similarities between statistical procedures are not unusual. Constructing a table of median Duration values for the PHOENIX dataset, for example, is similar to constructing a histogram or contingency table as above. Different types of resistant line construction share the same procedural structure. A procedure for kernel density estimation has the same structure as the histogram procedure, but uses different decomposition and reduction functions. Kernel density estimators also share structure with the more complex procedures of robust regression [Fox and Long, 1990]. In the next two chapters we will discuss more complex and more flexible combinations of primitive operations.

4.3 Related Work

Other AI researchers, especially in machine learning, have taken a more direct approach to the problem of EDA. In the end, the application of exploratory operations is aimed at uncovering potentially interesting relationships between variables and groupings of observations. It might seem that machine learning should be well-equipped to solve this problem. In fact, some researchers have argued that their machine learning techniques do encompass EDA, if EDA is defined as "discovering stable structure and

groupings in a large database of objects described as attribute value pairs"—in other words, as a kind of clustering [Biswas *et al.*, 1991, p. 591]. Nevertheless, other aspects of EDA are equally useful: descriptions of functional relationships, transformations to improve the clarity of patterns, hierarchical reductions of groups of observations, and so forth. More importantly, even though machine learning techniques can carry out some parts of an exploratory analysis, they have built-in limitations, due mainly to their lack of subject-matter knowledge—knowledge of what the data mean.

Consider a sampling of comments associated with datasets in the machine learning repository at the University of California, Irvine [Murphy and Aha, 1993]. The comments highlight the need for human involvement in the exploration process.

- "This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. ... Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence [of heart disease] (values 1, 2, 3, 4) from absence (value 0)." [Dataset 31-34 (heart-disease)]
- "The non-standard set of attributes have been converted to a standard set of attributes according to the rules that follow... A property such as age $>$ 60 is represented as a boolean attribute with values *f* and *t*." [Dataset 3-4 (audiology)]
- "BILIRUBIN is [a] continuous attribute ...however, [its values] represent so called 'boundary' values; according to these 'boundary' values the attribute can be discretized." [Dataset 35 (hepatitis)]
- "In line with the use by Ross Quinlan (1993) in predicting the attribute 'mpg', 8 of the original instances were removed because they had unknown values for the 'mpg' attribute." [Dataset 5 (auto-mpg)]
- "37 cases (5%) had one or more missing values... These were replaced by the mode of the attribute (categorical) or the mean of the attribute (continuous). [Dataset 72.1 (australian)]

- "The next-to-last y value appears clearly to be a transcription error, especially since the change of a single character—the initial 2—would be enough to bring it in accord with the smooth relationship of the figure..." [Schaffer, 1990, p. 122]
- "If we are willing to accept the absolute temperature T as observable, however, the relationship simplifies to $y = k_1 e^{k_2/T}$... Moreover, if we are willing to accept the *reciprocal* absolute temperature as a fundamental observable—on the basis of its common occurrence in physical formulas—then the relationship reduces to a simple exponential." [Schaffer, 1990, p. 126]

This is a selection from dozens of examples. While autonomous algorithms often perform very well given specific tasks, human judgment and knowledge of subject matter context play an indispensable role in refining problems, selecting and parameterizing algorithms, and evaluating results.

4.4 Summary

Most EDA procedures can be represented by combinations of three simple, heavily parameterized operations. Reductions simplify variables and relationships by boiling them down to a single number, or a small set of numbers. A common example of a reduction is the mean. Transformations apply a function to a variable or relationship, generating another data structure containing appropriately transformed values. The log transform is a familiar transformation. Decompositions break a variable or relationship into subsets, as when we bin a variable for histogramming, or when we decompose a relationship into constituent clusters.

These operations have surprising power in representing common EDA procedures. Unfortunately, this power comes at a price: searching directly through the space of primitive operations poses an intractable problem. Existing techniques for autonomous search offer a partial solution, but have clear limitations. An effective

solution will involve restructuring the search space to reduce its size and incorporating human knowledge in the generation and evaluation of results.

CHAPTER 5

PLANNING AND EDA

If we were to generate exploratory results by searching through the space of primitive operators defined in Chapter 4, we would face an enormous, intractable problem. Fortunately, we have another possibility: we can cast EDA as a planning problem. Humans make use of abstraction, problem decomposition, and procedures in exploring data; these are exactly the sources of power AI planners rely on to attack large search problems. There is a close correspondence between EDA and planning, one we can capitalize on in building an automated assistant.

AIDE is designed around a partial hierarchical planner. Partial hierarchical planners are a type of reactive planner, designed for complex, rapidly changing environments. In many ways the EDA search space can be viewed as such an environment: we can't plan every conceivable action in advance; each new nugget of information can change our view of the problem; a course of action we initially thought appropriate may suddenly become useless. Partial hierarchical planners have properties designed to help them cope with such environments. They can exhibit complex behavior, relying on pre-compiled, often hand-constructed plans to guide their actions. They are responsive to changes in the environment; they generate plans on the fly, rather than elaborating a plan to completion before executing it. Plan structures can be modified opportunistically in response to new information.

The exploration process, in the planning framework, becomes a matter of plans executing to satisfy goals. At the top level, a goal is established to explore a dataset. Plans are activated to accomplish this task. These plans establish more specific, detailed goals in their execution, goals of building models and describing individual relationships.

The planner that carries out this process has a very simple model of execution. The basic data structure is a stack of control units. Goals, plans, actions, and control constructs for sequencing, conditionalization, iteration, and so forth, are specialized types of control units. Control unit execution is controlled and guided by meta-level mechanisms to activate plans, bind variables, and switch between the most promising courses of action as the exploration proceeds.

The design of AIDE planner draws heavily on existing systems, especially RESUN [Carver and Lesser, 1993] and PHOENIX [Cohen *et al.*, 1989]. The AIDE planner is nevertheless significant in its own right: it provides a simple, powerful execution model for partial hierarchical planning; its plan language provides features that existing planners have largely neglected (e.g. plans and control constructs as first-class objects); its representation of procedural knowledge is sufficiently general to be useful in domains other than EDA..

5.1 EDA as Planning

Planners formulate problems in terms of states, goals, and combinations of actions to achieve goals. A planner solves a problem by constructing a step-by-step specification of actions that move from the initial conditions (the start state) through intermediate states to the desired conclusions (the goal state). The construction can be complicated by constraints on the application of actions, uncertainty about the effects of actions, and unforeseen interactions between goals, among other difficulties. Planners rely on task decomposition to solve complex problems, using knowledge about states, actions, and goals to structure the search at different levels of abstraction [Russell and Norvig, 1995]. The simplest planners work by backward-chaining. Given a goal state, a planner begins by examining those actions that achieve the goal state. By treating the preconditions of these actions as goals to be satisfied in turn, and taking note of potential interactions between actions, the planner recursively generates a specification of appropriate actions.

Problems can sometimes be solved much faster (exponentially faster [Korf, 1987]) by planning than by direct search. How can we tell whether a problem is amenable to planning? Three types of knowledge characterize planning: abstraction, problem decomposition, and procedures [Korf, 1987]. These areas of knowledge are central to the techniques humans use in exploring data—EDA is thus a planning problem.

This becomes clear, while still remaining implicit, in John Tukey's introduction to *Exploratory Data Analysis*, quoted in Chapter 1:

> A basic problem about any body of data is to make it more easily and effectively handleable by minds—our minds, her mind, his mind. To this general end:
>
> - anything that makes a simpler description possible makes the description more easily handleable.
> - anything that looks below the previously described surface makes the description more effective.
>
> So we shall always be glad (a) to simplify description and (b) to describe one layer deeper [Tukey, 1977, p.v].

Tukey's account of exploration emphasizes two related aspects: description by hierarchical problem decomposition and description through abstraction. We saw concrete examples of these notions in the exploration of the PHOENIX data in Section 2.1.

We began by noticing a gap in the distribution of FirelineBuilt values, and that the patterns in the two subsets derived by this observation were very different from one another. These subsets are shown in Figure 5.1. While it might be possible to generate a complex description to fit all the data in the relationship, we instead decomposed the relationship into two parts and described each part separately. Combining the resulting descriptions was then a matter of explaining the gap in FirelineBuilt values, for which we found the variable Outcome. This process is hierarchical problem decomposition. Many EDA problems are solved this way, by breaking the data into parts, describing the parts, and combining the descriptions into a more comprehensive result.

We continued with an exploration of the successful PHOENIX trials. Indications of linearity included a moderate correlation, no evidence of curvature, and only a few outlying values. We thus decided to fit a line, as shown in Figure 5.2. There are many ways one can fit a line to a relationship. The simplest is a least-squares or regression fit. Resistant lines, which are less sensitive to outlying points, offer another set of possibilities: there are at least half a dozen ways to construct such lines [Emerson and Hoaglin, 1983]. Alternatively, we could remove those points we consider outliers and then fit a line, or downweight those points in a weighted regression. Many of these procedures involve iterative improvement. These details, however, are unimportant at the time we decide that the relationship is approximately linear. We first make this decision, and then fill in the details about how the line should be fitted. Most EDA procedures follow a similar pattern of abstraction.

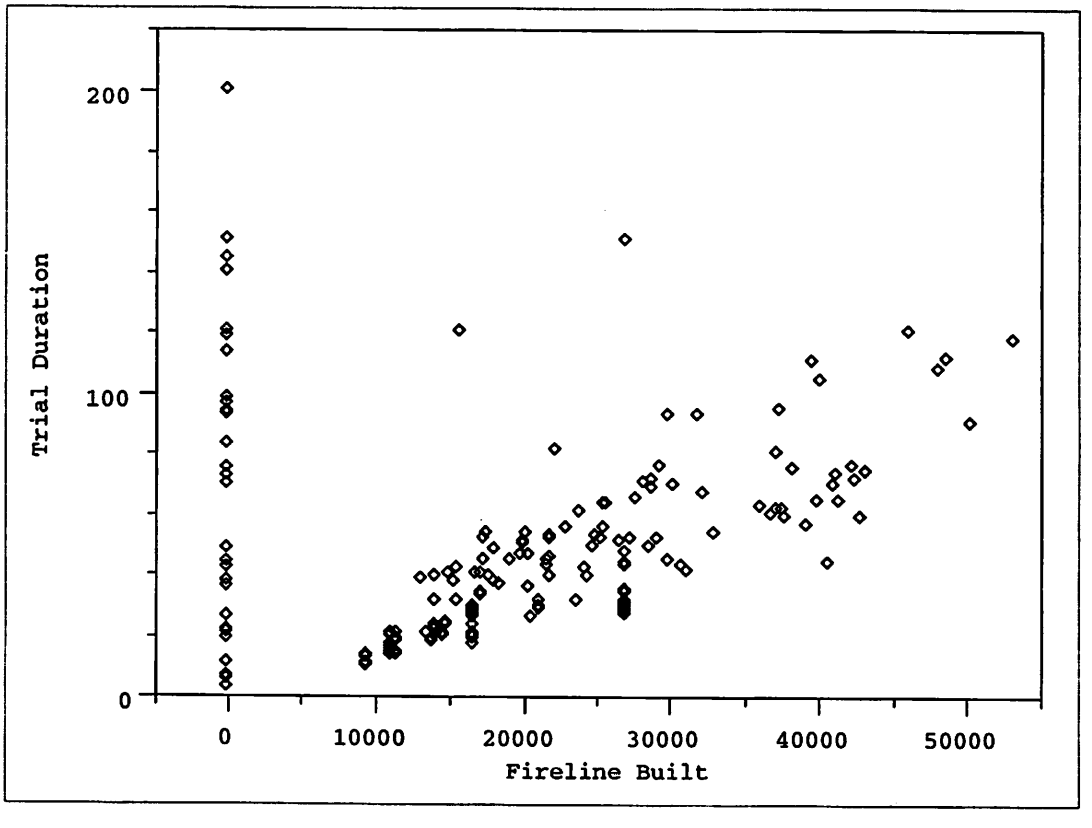


Figure 5.1. FirelineBuilt versus Duration

In addition to using hierarchical problem decomposition and abstraction, EDA is also strongly procedural. In other words, when we execute an exploratory operation we generally have a good notion of the next few operations that we should execute. In the PHOENIX data, after we decomposed the relationship into clusters, as shown in Figure 5.3, we aggregated the clusters to search for further patterns. When we fit a line to a relationship, we run tests for patterns in the residuals. More complex procedures for constructing descriptions are extremely common in the EDA literature [Hoaglin *et al.*, 1983; Hoaglin *et al.*, 1985].

Finally, as others have noted, data modeling procedures are inherently constructive [Lansky and Philpot, 1993; Huber, 1994]. Exploration is similarly a constructive process. The results of an exploratory session are not simply the p-values, tables, graphs, and so forth that have been computed. The interpretation of these individual

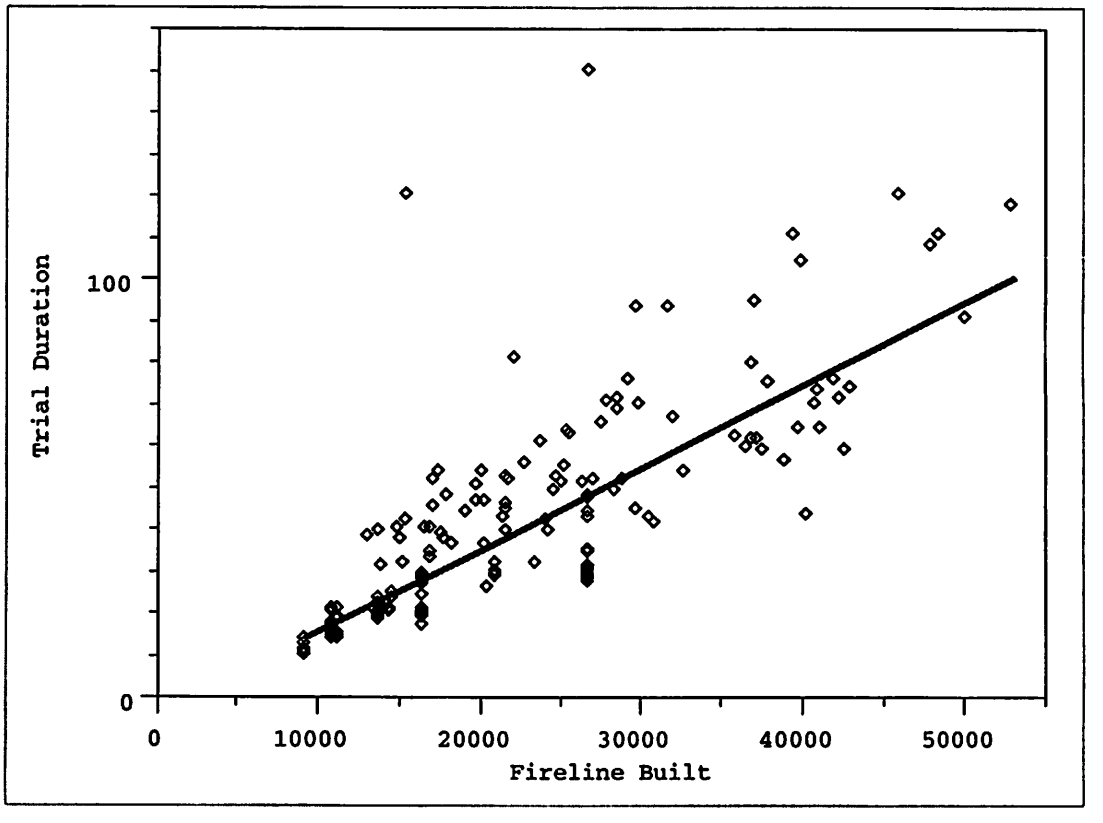


Figure 5.2. Linear fit of FirelineBuilt versus Duration, successful trials

results depends on how they were derived. Interpretation of the residuals of a linear fit depends on whether a regression or resistant line was applied; individual cluster properties depend on clustering criteria. In many cases even the knowledge that some operation has been applied and has failed can influence our judgment of the importance of a related result. As Peter Huber puts it, "Data analysis is different [from word processing and batch programming]: the correctness of the end product cannot be checked without inspecting the path leading to it." [Huber, 1994, p. 69] Huber's path corresponds to a plan, in AI terms.

By capitalizing on knowledge of abstraction, hierarchical problem decomposition, and procedures, we can recast EDA as a planning problem. Thus, rather than taking single steps through an enormous search space, a planner for EDA can search through a much smaller space of more abstract states.

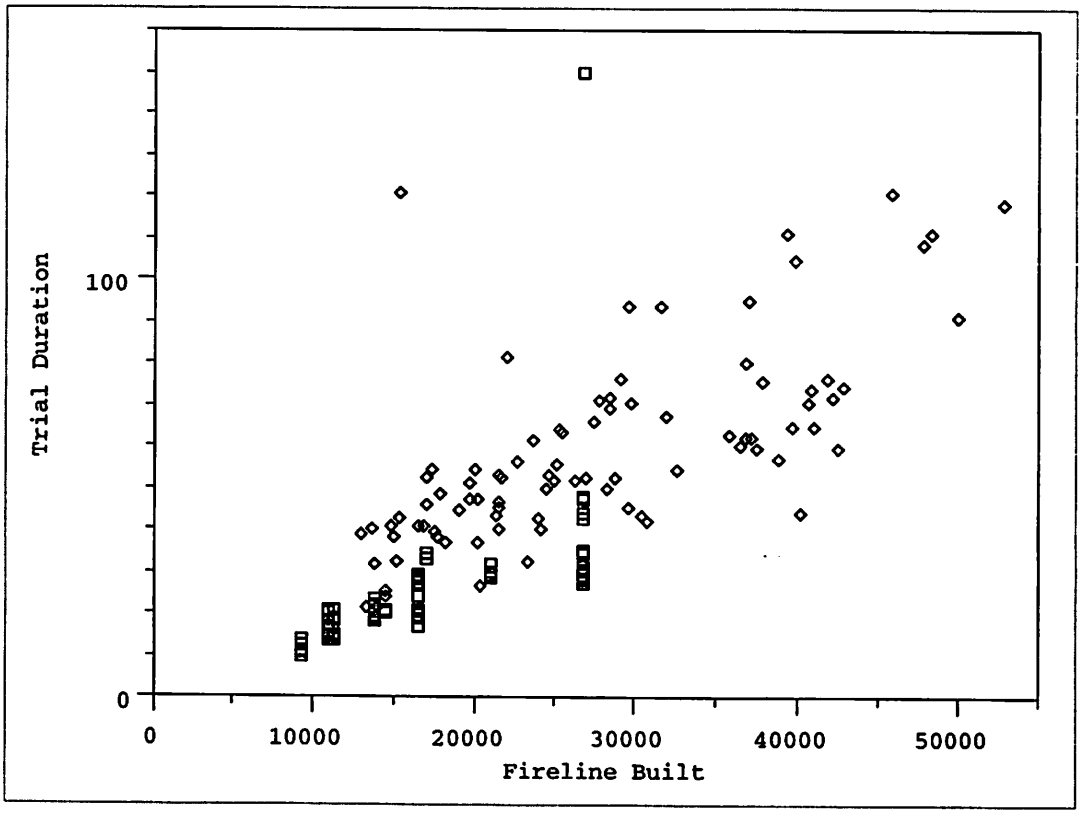


Figure 5.3. Clusters in FirelineBuilt versus Duration, successful trials

5.2 EDA as Partial Hierarchical Planning

An planning formulation of EDA provides a basis for automated strategic support of the task. There are a great many approaches to planning, each with advantages and drawbacks. To help us settle on an appropriate planning formulation for EDA, we need to examine the process in more detail.

First, exploratory procedures rely on explicit, often complex control constructs. Many fitting procedures iteratively improve a description until it is judged adequate. A resistant line fitting procedure for (x, y) can be described informally as shown in Figure 5.4. This procedure involves a sequence of steps, a conditional test, and what may be seen either as recursion or iteration. Variables are bound to computed values and updated as needed.

Many EDA procedures take the form of tests of generated values, iteration, recursion, and other types of control. These procedures may be complex, but fortunately there is little need to generate each one from scratch; there is a great deal of commonality between procedures. In Section 4, for example, we saw that constructing a histogram involves the same procedures as constructing a contingency table: the contingency table is a two-dimensional analog of the histogram, with cell counts

```
Define Resistant-Fit (x, y, initially slope = 0, intercept = 0):
  Sort (x,y) by increasing x values;
  Divide sorted (x,y) into three equally-sized groups;
  Compute medians for each group;
  new-slope = slope + adjustment by left and right medians;
  new-intercept = functional combination of new-slope and medians;
  If (new-slope - slope) is sufficiently small,
     then exit with new-slope and new-intercept;
     else
        Extract residuals;
        Resistant-Fit (x, residuals, new-slope, new-intercept).
```

Figure 5.4. Pseudo-code for resistant fit procedure

corresponding to bin heights. We can draw similar analogies between procedures for smoothing and for generating kernel density estimates, or between resistant line fitting and locally-weighted regression curves. While sometimes novel procedures are constructed from scratch, variations on existing procedures are much more common.

Second, exploratory procedures are applied opportunistically. EDA relies heavily on the observation of indications, suggestive features of the data [Mosteller and Tukey, 1977]. At each point in the process we can determine what the next few steps should be; the details of how to proceed must often wait until we have actually performed those steps and see the types of indications that arise. Each exploratory operation is thus simultaneously an effective action and an information-gathering action. In Figure 5.3 we observed indications of clustering in our exploration of successful trials. We could not have predicted these clusters, and it would have been impractical in the extreme to generate procedures to handle every possible type of pattern that might arise in the data. Once we observe them, though, our course of action is clear: we search for other variables to explain the pattern.

These two points have a number of implications for an EDA planner.

- Plans must be able to represent control explicitly, including sequences, iteration, and conditionalization of actions.
- Actions must be able to compute numerical values for variables; planning decisions must be able to use these values.
- The planner must be able to interleave plan generation with plan execution, to take advantage of new information about how to proceed.
- The planner must be able to retrieve and apply existing plans as well as constructing new ones.

A type of planning called partial hierarchical planning [Georgeff and Lansky, 1986] meets these needs. Systems that use the approach include PRS [Georgeff and Lansky,

1987], the PHOENIX planner [Cohen *et al.*, 1989], the RESUN system [Carver and Lesser, 1993], and to some extent languages for reactive control such as XFRM [McDermott, 1992]. Partial hierarchical planning techniques were developed in an attempt to overcome the limitations of existing planners; to understand the partial hierarchical approach we first need to understand these limitations.

One of the earliest efforts in planning research resulted in the STRIPS planner [Fikes and Nilsson, 1971], the first of the so-called classical planners. In STRIPS, states are logical formulas, constructed from a finite vocabulary of symbols. Operators move from the current state of the world to another by adding or deleting formulas. Classical planners construct a plan by searching through a space of world states. The classical planner's task is complete when it has constructed a partially ordered set of operators that lead from the start state to the goal state. Execution of the plan is carried out by another, separate system.

Unfortunately, the classical approach is often inadequate to deal with problems in the real world. Plans are rarely generated entirely from first principles; they are modifications of existing plans or constructed from pieces of other plans. Plans are often more complex than sequences of primitive operations; they encode procedural knowledge about how to act, in the form of conditionals, iteration, recursion, and the like. Planners almost never have an indefinite amount of time to search a combinatorially intractable space of plans; rather, sometimes action must be taken whether the planner is finished or not. Partial hierarchical planning was developed in response to these issues.

First, and most importantly, partial hierarchical plans are specifications: they implement procedural knowledge. In the classical formulation a plan is a partially ordered sequence of actions with annotated "causal" links between actions. A partial hierarchical plan is a control schema of subgoals to be achieved. The schema explicitly specifies how the subgoals must be achieved for the plan to complete successfully. Schema constructs typically allow sequences, conditional establishment, and iteration

of subgoals. They may also provide for parallel satisfaction of subgoals, mapping over lists of subgoals, recursion, and more complex domain-specific processing.

Because these plans may be complex and thus difficult to generate de novo in a timely way, a partial hierarchical planner stores its plans in a library, to be retrieved when appropriate. This approach is comparable to that taken by case-based planning. These plan definitions are not the fully elaborated sequences of actions that are usually stored in a case library, however, but are specifications of subgoals: they are thus *hierarchical.* Information available at plan design time is thus used to reduce search at run-time.

The planner constructs plans by searching through the library, its set of partial solutions, for appropriate ways to satisfy established goals. With a classical planner, the plan would need to be fully elaborated—all primitive actions determined, and a partial order established—before any action could be executed. In contrast, a partial hierarchical planner may execute the first action of a plan as soon as it has been determined, regardless of the status of the rest of the plan. In other words, a plan may be *partial* at the time of its execution. This behavior has advantages over classical, off-line planning: in dynamic environments, information necessary to choose a specific action may not be known at planning time; in complex or uncertain environments, an action may generate too many possible results to enumerate exhaustively in advance. By interleaving planning and execution, planning effort can be reduced.

To summarize, partial hierarchical planning provides a good match for EDA. Explicit plans are well-suited to the sophisticated control often required in EDA procedures. Because EDA procedures share common structure and are relatively general, they can be stored in a library for retrieval when needed. Finally, because local EDA decisions are often data-driven (i.e., indications guide exploration), a planner that interleaves generation and execution can be more effective than one that does not.

5.3 The Planner

The design of the AIDE planner reflects the foregoing observations about EDA and its relationship to partial hierarchical planning. The AIDE planner is simple but powerful; the simplicity of its design is one of its strengths. In abstract terms, the AIDE planner does little more than manipulate stacks of *control units*. The planner is essentially a high-level language interpreter, in which the `*active-stack*` stores the current execution context. The top level planning loop is simple enough to present in pseudo-code, as shown in Figure 5.5.

In words, the planner executes the control unit at the top of the planning stack, by calling its execution method. If this generates a new control unit, then it is pushed onto the stack to be executed in turn. The process continues as long as the topmost stack element has the status `:in-progress`. If this status changes to `:succeeded` or `:failed`, then the control unit is not executed, but rather popped off the stack, its completion method being called at that point.

A control unit is similar in many ways to a stream, in the sense used by Abelson and Sussman [Abelson *et al.*, 1985]. Streams are an element of demand-driven programming, in which the execution behavior of a program element is not determined until it is actually needed. When executed, a control unit performs some action (as

```
(loop until (stack-empty-p *active-stack*) do
      (let ((current (stack-top *active-stack*)))
        (case (get-completion-status current)
              ((:unstarted :in-progress)
               (let ((next (execute-control-unit current)))
                 (when next
                       (stack-push next *active-stack*))))
              ((:succeeded :failed)
               (complete-control-unit current)
               (stack-pop *active-stack*)))))
```

Figure 5.5. Top level planning loop

a side effect) but also potentially returns another new control unit to be executed. Conventional planning structures (goals, plans, and actions) as well as partial hierarchical planning elements (control constructs for sequencing, conditionalization, iteration, and so forth) are all built around specializations of the basic control unit. The behavior of a control unit depends on the specialized definition of its execution and completion methods, as we discuss in the following sections.

5.3.1 Goals

A goal is a direct subtype of a control unit. When generated, a goal contains a list of constants and unbound variables. In this form,

```
(:SUBGOAL fit (generate-fit ?relationship ?op ?fit-structure)),
```

`generate-fit` is a constant, and the terms prefixed by "?" are variables. A goal is always generated in the context of a specific plan. If this plan has existing bindings for the goal variables, then the goal variables are immediately bound to these values. For example, some plan `fit-relationship` may bind `?relationship` to a specific value, (x, y), and then establish the subgoal `(generate-fit...)`, in which `?relationship` is then bound to (x, y).

In the classical planning framework, there is no notion of "goal execution." A goal exists to be satisfied (matched) by either a plan or a primitive operator, and it is only these latter objects that actually take any action. As a control unit, however, a goal does have a specific execution behavior: it causes plans to be activated that can potentially satisfy it. More specifically, when a goal appears at the top of the execution stack, its execution method causes the planner to search through the plan library for a matching (unifying) plan. The planning process continues with this new plan pushed onto the execution stack.[1]

[1]For the purposes of exposition, we are assuming here that for each goal that is established, only one plan in the library exists to satisfy it. Section 5.4.2 gives a fuller explanation of the process.

Again in the classical planning framework, there is no notion of "goal completion." A goal is either satisfied or not. In the control unit representation, control units are either `unstarted`, `in-progress`, or `completed`. If `completed`, a control unit has a completion status of either `:succeeded` or `:failed`. A goal completes when its matching plan completes; its completion status is that of the plan.

On successful completion, a plan will generally rebind the goal's variable bindings to reflect its success; these new goal variable bindings are then made available to the control construct or plan that initially established the goal.

5.3.2 Plans

A plan is a direct subtype of a control unit. It contains an initial set of static variable bindings, a `:satisfies` form, a set of constraint tests on variables that appear in the `:satisfies` form, a set of static properties, and a body. The body of a plan is a control schema of subgoal specifications, subgoals which must be satisfied for the plan to complete successfully. Control constructs in the schema allow sequencing (`:sequence`), iteration (`:while`), conditionalizing (`:if`, `:when`) of subgoals, as well as other domain-specific forms. A simple plan is given in Figure 5.6.

```
(define-plan fit-relationship ()
  "Describe a relationship by fitting it. Evaluate the fit."
  :satisfies  (generate-description :fit ?structure ?op ?fit-structure)
  :features   ((?structure ((:dataset-type relationship)
                            (:cardinality 2))))
  :body       (:SEQUENCE
               (:SUBGOAL generate-fit
                         (generate-fit ?structure ?op ?fit-structure))
               (:SUBGOAL evaluate-fit
                         (explore-by ?strategy ?activity ?model
                                     ?fit-structure ?deepening-result))))
```

Figure 5.6. Plan to fit a relationship

In words, this plan computes a fit for a bivariate relationship and then examines the fit for possible deviations. The `:features` form constrains the plan's applicability. The body of the plan specifies that two subgoals must be satisfied in sequence for the plan to complete successfully. The `generate-fit` subgoal may be satisfied by different subplans: AIDE's current plan library contains plans for a least-squares linear fit, different varieties of resistant line fits, and smoothing fits. The `evaluate-fit` subgoal, in each of these cases, is matched by plans that explore the residuals for patterns not captured by the fit. While not an example of deep statistical knowledge, this plan nevertheless captures a basic exploratory strategy. As Tukey writes, "Anything that looks below the previously described surface makes [a] description more effective." [Tukey, 1977, p.v] The `fit-relationship` plan makes explicit the dependence between the fitting and deepening procedures. Thus a pattern that appears in the residuals at the `evaluate-fit` step (outliers, say, for a regression fit) can be interpreted by the system as indicating that a different operation (perhaps a resistant fit) is called for at the `generate-fit` step.

Plans execute in an attempt to satisfy a matching goal. The `fit-relationship` plan, for example, is activated to satisfy the goal `(generate-description...)`. A plan executes by instantiating the top-level control unit represented in its body. In the `fit-relationship` plan this is the `:sequence` construct.

When a plan completes, it informs its matching goal of its completion status, either `:succeeded` or `:failed`. If successful, the plan may change the bindings of variables in the goal it has satisfied. For example, if the `fit-relationship` plan completes successfully, then it will pass back a result to the goal `generate-description` by binding the `?fit-structure` variable.

5.3.3 Actions

An action control unit is a subtype of the plan control unit type. The main difference between the two types is that, rather than having a schema of control

constructs for its body, an action has executable code. An action thus generates no new control unit when it executes. Otherwise, an action interacts with goals in just the same way as plans: an action is activated to satisfy a goal, and does so by rebinding the goal's variables to reflect its execution. An example is given in Figure 5.7.

Notice that an action in the AIDE planner is very different from a classical planning operator. Primitive planning operators in the classical framework are indistinguishable from goals: an operator is a goal form with all its variables bound to constants. (Some modern planners allow rebinding of variables to values at runtime, but it's not clear whether this adds any power to the representation.) A partial hierarchical planning action, in contrast, can perform any arbitrary manipulation of its input to produce bindings for its variables on completion.

The AIDE planner also allows for "anonymous" actions in the body of plan or other control construct, as in the form below:

```
(:ACTION (print 'anonymous-action)).
```

```
(define-action extract-x-residuals
    ""
   :satisfies (extract-residuals ?fit :x ?residual-relationship)
   :action    (in-action-environment
                (:bindings ((?fit :attributes (x y predicted))
                            (?residual-relationship :class relationship)))
                (copy ?fit (function 'values)
                      :key (attributes x)
                      :name (attribute-name x)
                      :output ?residual-relationship)
                (transform ?fit (function '-)
                           :key (attributes predicted y)
                           :name (derive-name (attribute y) 'residual)
                           :output ?residual-relationship)
                (values t (return-bindings '?residual-relationship
                                           ?residual-relationship))))
```

Figure 5.7. Action of extracting residuals

These actions are executable code, evaluated only for side effects. Anonymous actions always complete successfully.

5.3.4 Sequencing

A :sequence control unit executes a sequence of subordinate control units in order, for example,

```
(:SEQUENCE
  (:SUBGOAL fit (generate-fit ?relationship ?op ?fit-structure))
  (:SUBGOAL eval (evaluate-fit ?op ?fit-structure. . .))).
```

If any subordinate unit completes with a :failed status, the :sequence unit fails as well, and does not execute any of the elements that follow the failed one. Otherwise, if all subordinate units complete with the status :succeeded, the :sequence unit succeeds as well.

The :sequence control unit is a good example of AIDE's demand-driven programming style. In the example above, the ?fit-structure variable is bound only on completion of the fit subgoal. If the fit subgoal fails, then there will be no need to generate and execute eval subgoal. Thus the :sequence unit maintains a stream [Abelson *et al.*, 1985], or a "promise" of later execution, that generates the eval subgoal only when the fit subgoal has completed successfully.

5.3.5 Conditionalization

The :when control unit is implemented in a similar fashion to the :sequence unit. The :when unit consists of an executable test form and a single subordinate control unit for its body.

```
(:WHEN (typep ?relationship 'bivariate-relationship)
  (:SUBGOAL fit (generate-fit ?relationship ?op ?fit-structure)))
```

The :if control unit is a variant of the :when unit. It allows two body forms, one for if the test form returns nil, the other for non-nil.

The :when unit first evaluates its test form. If the test form evaluates to non-nil, then the :when execution method returns the subordinate unit. Planning continues with this new unit. Eventually, when this new unit completes with status :succeeded or :failed, the :when completes as well with the same status.

If the test form evaluates to nil, in contrast, then the :when unit completes with status :succeeded. This may seem unintuitive; in Common Lisp for example the when special form returns nil if its test evaluates to nil. We need to remember, however, that these constructs are deciding whether or not to act. If a :when unit decides that taking an action, or expanding a subordinate control construct, is unnecessary, then there is no need for the entire plan to halt with failure. On the contrary, the plan should proceed normally from that point.

5.3.6 Iteration

The :while control unit is another variant of the :when unit. It also consists of an executable test form and a single subordinate control unit for its body. Variations on the :while form include :repeat, for incrementing a numerical variable over a specified range, and :map, for binding a variable to successive values in a list.

```
(:WHILE (test-for-curvature ?new-relationship)
  (:SUBGOAL transform (transform ?relationship ?new-relationship)))
```

The :while unit repeatedly returns its subordinate control unit body, as long as its test form returns non-nil. In other respects it behaves exactly like the :when unit.

5.3.7 Conjunction and Disjunction

The :and unit is comparable to the :sequence unit. It specifies that all of its subordinate control units must complete successfully. Analogously, the :or unit specifies that at least one of its subordinate control units must complete successfully. Remaining consistent with the :sequence unit, :and and :or are short-circuiting. A

short-circuiting :and completes with failure as soon as any of its subordinates fail, while a short-circuiting :or completes with success as soon as one of its subordinates succeeds.

In the current implementation, subordinate control units of :and and :or forms are pursued in a fixed, left-to-right order.[2]

5.4 Control

A step in the planning process, as presented up to this point, can be represented as shown in Figure 5.8. The uppermost stack is the current execution stack, with the current element shown at its top. When the current element is executed, the either a new element is pushed onto the execution stack, as shown on the left, or the current element is popped off the execution stack, as shown on the right. Execution continues with the stack in its new state. As control units representing plans, goals, and so forth, are processed, the execution stack grows and shrinks accordingly.

To simplify our discussion we have assumed that when a goal is to be satisfied only one plan matches, the only one possible way to proceed. This is highly unrealistic; indeed, if problems were so constrained, there would be no need for planning (or any kind of search) at all. In any problem of interest, many plans can potentially satisfy

[2]A better solution would be provided by letting the ordering change dynamically. This is a feature of Carver's RESUN planning representation, in the form of subgoal focusing units.

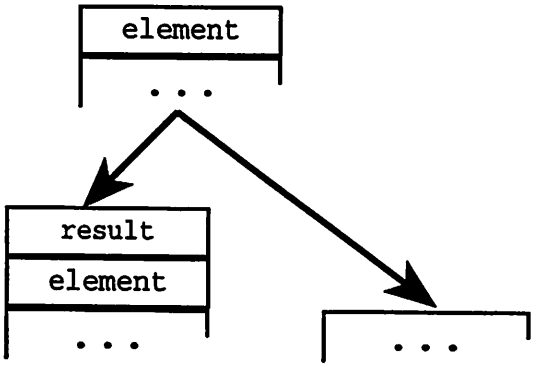


Figure 5.8. A step in the planning process

a single goal. In AIDE, for example, there are several ways to satisfy a goal that involves describing a relationship by a straight line: a regression line, resistant lines, weighted regression lines, and so forth. Each possibility is represented by a different plan; all can potentially satisfy the same goal. Similarly, when an action returns a binding for a variable, it may not always be the case that only a single binding is possible. In AIDE we can define an action to break a relationship into clusters, by binding a `?clustering-criterion` variable to some specific value. There may be a large number of plausible ways to cluster the relationship, and it may not be possible initially to select one as being the best way. The planner must maintain the different possibilities as long as they have not been ruled out.

The planning process involves more than plans that expand to control constructs that expand to subgoals, and so forth, in a recursive expansion. The process also includes branch points where the planner can decide between several possible ways to proceed. This raises a number of questions about the behavior of the planner:

- How should the planner select appropriate plans to satisfy a given goal? Analogously, how should the planner select appropriate bindings for a variable if there are many possibilities?

- In attempting to satisfy a given goal, when should the planner switch from one plan to another plan that might be more promising? When should the planner switch between variable bindings?

- When should the planner switch from trying to satisfy one goal to try to satisfy another? When should the planner switch between considering bindings for one variable to consider bindings for an entirely different variable?

Three mechanisms address these issues: control rules, focus points, and meta-level plans. We'll discuss each of these mechanisms separately. The motivation for their design depends somewhat on the requirement that AIDE be a mixed-initiative system.

Here we can only touch on these issues briefly; they are covered in more detail in Chapter 7.

5.4.1 Control Rules

When more than one plan in the library matches a goal, AIDE must decide which to instantiate. Control rules make this decision. Three steps are involved in executing a plan to satisfy a goal: matching, activation, and preference. The matching step attempts to unify the goal with the `:satisfies` form of each plan in the library. If successful, this establishes that the plan is syntactically able to satisfy the goal. `power-transform`, for example, satisfies the goal of fitting a power function to a relationship. The activation step involves running a set of rules that further test the applicability of plans, in order to activate or deactivate them. The `power-transform` plan is only activated in the presence of curvature indications. The preference step involves running another set of rules that apply preferences to the active plans. In the presence of a curvature indication, the `power-transform` plan is preferred to the `linear-regression` plan. Evaluation of the bindings of plan variables follows a similar procedure.

The distinct matching and activation phases let the planner incorporate contextual information into the decision about which plans should be pursued. Contextual information includes, for example, the type of model a user has selected to describe a dataset. If the user is building a regression model, then the ordering of plans to describe a relationship differs from the ordering for a cluster model.

More importantly, flexible interaction with the user requires these separate phases. Imagine, in contrast, a system that encodes all activation criteria directly in the `:satisfies` form of its plans. It runs all those plans that match a given goal; those plans that do not match are never considered. Suppose, however, that some of the planner's activation criteria are incorrect. In a mixed-initiative system, we would like to be able to tell the planner, "Run plan P, even though your internal processing

hasn't activated it." Unfortunately, because the planner has ruled out plan P on syntactic grounds, it cannot follow the directive. To be effective in a mixed-initiative system, the planner must distinguish its preferences, or soft constraints that can be overridden, from hard constraints that cannot.

5.4.2 Focus Points

Candidate plans and plan variable bindings are maintained by focus points. Focus points fit into plan execution as follows. As plans, goals, and control constructs (i.e. different types of control unit) execute, they generate an active execution stack. Whenever there arises a choice between plans to satisfy a goal or values to which a variable can be bound, a focus point is generated. This focus point maintains an agenda of newly generated execution stacks, each containing as its top element a control unit representing one of the candidate plans or variable bindings. The execution method of the focus point—a focus point is yet another type of control unit—selects one of these stacks to become the active execution stack, and returns its top element. The planner continues with the new stack and the new control unit. The planning code presented in Figure 5.5 can easily be revised to account for this change, as shown in Figure 5.9. When executed, a control unit may potentially return another control unit and in addition a new execution stack.

Each focus point maintains an agenda of candidate possible plans or variable bindings. These candidates are those activated by control rules during the evaluation phase; as might be expected, then, the agenda is ordered by applicable preference rules.

Just like other types of control units, such as plans and goals, a focus point's behavior is determined by its specialized execution and completion methods. It will be simplest if we discuss only the behavior of focus points for multiple plans; focus points for multiple variable bindings behave similarly. The execution method of a focus point returns the plan associated with the one of the stacks in its agenda;

this stack also becomes the active execution stack. The process is diagrammed in Figure 5.10. When a plan that was initially generated by a focus point eventually completes, the focus point is informed of the completion. It can then either complete itself, informing the goal of the success or failure of the plan, or select a different plan and stack to try a different way of satisfying the goal.

Let's take the PHOENIX analysis again as an example. In exploring the dataset, a goal is established to describe the relationship (Effort, Duration):

```
(:SUBGOAL describe (describe-relationship ?relationship
                                          ?descriptive-operation ?result)),
```

```
(loop until (stack-empty-p *active-stack*)
      do (let ((current-element (stack-top *active-stack*)))
           (case (get-completion-status current-element)
             ((:unstarted :in-progress)
              (multiple-value-bind (next-element new-stack)
                  (execute-control-unit current-element)
                (when next-element
                  (stack-push next-element *active-stack*))
                (when new-stack
                  (setf *active-stack* new-stack))))
             ((:succeeded :failed)
              (complete-control-unit current-element)
              (stack-pop *active-stack*)))))
```

Figure 5.9. Revised top level planning loop

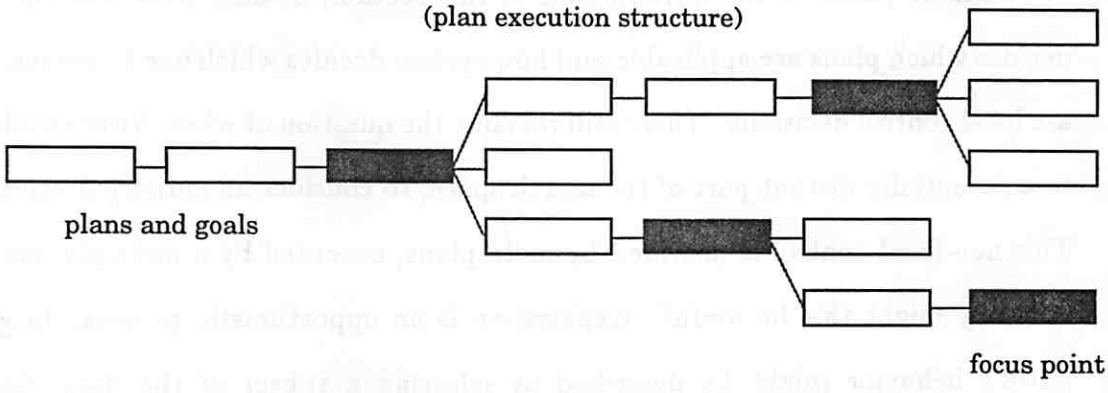


Figure 5.10. Branches in the planning process

where `?relationship` is bound to (Effort, Duration). Because the relationship has indications of local clustering, but is also approximately linear, several plans are applicable. A focus point is generated to manage the plans. It executes by returning the first plan (with an associated execution stack) on its agenda, a plan for fitting a line to the relationship. When this plan completes, the focus point then must decide whether the linear fit is sufficient as a description of the relationship, or if other plans should be pursued as well. A focus point for high-level exploration goals of this type always pursues all matching plans. Thus the focus point is executed again, causing the next plan, one for exploring clusters in the relationship, to be returned. Only when all activated plans have been executed does the focus point complete its execution. This behavior gives the planning process an iterative flavor, an essential property of exploration.

A focus point is thus responsible for interpreting the results returned by its candidate plans. One focus point hight return as soon as its first plan completes, whether successfully or not, while another focus point might continue to execute until one of its plans succeeds. Because of the object-based representation of focus points, specialized behavior is easy to implement and manage.

5.4.3 Meta-Planning

Control rules and focus points provide the mechanisms to answer the first two sets of questions posed in the introduction to this section, dealing with how the system decides which plans are applicable and how system decides which one to pursue. These are local control decisions. There still remains the question of when AIDE should jump to a potentially distant part of the search space, to consider an entirely different goal. This non-local control is provided by meta-plans, executed by a meta-planner.

Why might this be useful? Exploration is an opportunistic process. In general, AIDE's behavior might be described as selecting a subset of the data, finding a specific plan for the data, executing the plan, examining the results, and continuing.

This style of processing is fine when all goes as planned, but sometimes unexpected results are generated. In a regression fit, two or three points could turn out to have unreasonably high leverage. Two or three clusters out of many could show a strong linear relationship. In such cases, the appropriate response is to retreat to an earlier step in the exploration—even though the current plan has not completed execution—to take account of this new information. This opportunistic backtracking, which has been called "refocusing" [Carver and Lesser, 1993], is handled by the meta-planner.

Whenever a focus point is executed (or re-executed, on plan completion) the meta-planner is invoked. The meta-planner is identical in design to the planner that executes base-level plans for description, modeling, and so forth. The meta-planner has a single, specialized purpose, however; it is responsible for deciding which focus point should be active and which of its agenda items should be pursued. In the usual course of processing, the meta-planner takes as input the currently active focus point, along with its agenda, and returns the same focus point, with the best agenda item (as rated by preference rules). This is all handled by a single default meta-plan. Different meta-plans are activated, however, under different circumstances, and these serve to change the active focus point or its currently executing agenda item. The current set of meta-plans in AIDE's library implement a depth-first search in the default case. In some situations the agenda of a focus point are explored in breadth-first fashion; it is useful, for example, to partially execute all of the residual exploration plans for a linear fit, until any relevant indications are derived, rather than executing each one fully to completion. In other situations the meta-planner must search through the plan network for an appropriate focus point.

In contrast to the common approach of making each meta-level decision independently, the AIDE meta-planner coroutines with the base-level planning process. The motivation for this design is straightforward: sometimes the meta-level decision

about which plan to pursue are not independent of previous meta-level decisions. This design is comparable to other successful arrangements: Wilkins describes a design in which two independent planners, PRS and SIPE, cooperate as coroutines [Wilkins, 1988]); Wilensky discusses planning and meta-planning decisions being made in a uniform framework [Wilensky, 1981]; the RESUN planner, designed as a meta-level control system, implicitly coroutines with lower-level activities.

5.5 Related Work

The design of the AIDE planner draws directly on the RESUN planner [Carver and Lesser, 1993] and the PHOENIX planner [Cohen *et al.*, 1989]. A prototype of AIDE was initially built using the RESUN planner as its processing engine, and AIDE's design naturally reflects this experience. AIDE also incorporates the ideas embodied in PRS [Georgeff and Lansky, 1986] and early work in incremental blackboard-based planners [Durfee and Lesser, 1986]. We will discuss each of these systems in turn, by describing the intended use of the planner and those properties that influenced the design of AIDE.

Approaches other than partial hierarchical planning have evolved as well. Modern classical planners can now represent and solve much more complex problems than could be handled by STRIPS. More recently, partial-order causal-link (POCL) planners with well-understood theoretical properties have received a great deal of attention in the planning literature. We'll briefly discuss these types of planners, and explain why there still remain reasons to prefer partial hierarchical planning for EDA.

Despite the availability of powerful planning systems, the AIDE planner was developed from scratch. It might have been possible to overcome technical problems, with some reasonable effort, but there is yet another practical reason to have gone this route. It is often easier to build a tool that dóes exactly what one needs than to adapt an existing general-purpose system; we can then attribute specific, desired behaviors

to specific features of the architecture of the new system, with fewer confounding factors than we would have to consider otherwise.

5.5.1 Incremental Planning

AIDE's design reflects themes in incremental planning developed by Durfee and Lesser [Durfee and Lesser, 1986]. Their approach views the resolution of uncertainty as a central issue in controlling problem-solving activity. That is, a general-purpose planner must decide which goals are worth pursuing and which sequences of actions will achieve these goals. For many problems, the information necessary to make these decisions is initially insufficient: the planner must generate short-range plans and actions to resolve uncertainty about how to behave in the longer term.

This kind of incremental, data-directed planning tends to rely on local information and decision-making to develop complete solutions. Performance can be improved, as Durfee and Lesser show, if the planner can form a higher-level, more strategic view of the problem-solving situation.

Durfee and Lesser's planner acts in the domain of vehicle monitoring, in which the task is to identify, locate, and track patterns of vehicles moving through a two-dimensional space, based on acoustical data. The planner's formation of plans, monitoring, modification, and execution are all interleaved. Planning begins with the generation of a set of spatial and temporal constraints that give a rough, abstract estimate of where solutions lie. This abstract information leads to a set of goals, to distinguish and refine potential solutions. Rather than simply attempting to satisfy the goals as they stand, the planner attempts to reduce "control uncertainty" in two ways: it orders intermediate goals so that longer-term goals become easier to satisfy (or even become eliminated entirely) and it determines a detailed sequence of actions to satisfy the nearest-term goal. A small number of domain-independent heuristics support the planner's reasoning in the task.

AIDE is not based directly on this planner, which is embedded in a blackboard system. Nevertheless there are strong similarities in their approaches to planning. Most significantly, in both systems a data preprocessing phase creates contextual information that can influence how the data should be processed further. In AIDE, this involves generating indications for the initial data and intermediate results. Other similarities between AIDE and follow-on work to Durfee and Lesser's problem-solving system are described in Section 5.5.4.

5.5.2 PRS

Partial hierarchical planning was introduced with PRS, the Procedural Reasoning System [Georgeff and Lansky, 1986; Georgeff and Lansky, 1987]. In a PRS-based robot "the plans or intentions formed by the robot need only be partly elaborated before it decides to act." [Georgeff and Lansky, 1987, p. 677] This describes partial hierarchical planning in a nutshell. PRS was tested, with the SRI robot Flakey, as an assistant in a dynamic, uncertain world. To handle the uncertainty, PRS relied on several innovative features:

- Plans execute before being completely elaborated.
- Plans are not only constructed from first principles, but may be retrieved from a precompiled library.
- Plans incorporate complex control constructs, such as recursion and iteration.
- Once in progress, plans may be modified to meet the needs of a changing environment.
- Plans capture abstract behaviors, rather than concentrating only on navigation and low-level sensor/effector tasks.
- Meta-level activities are closely integrated with the execution of ordinary actions. The representation of meta-level plans (or knowledge areas) is at the same level as the representation of base-level plans.

As discussed more fully in Section 5.2, AIDE relies on all these ideas, though implemented in a different form than in PRS.

5.5.3 PHOENIX

PHOENIX is a simulation of forest fires and fire-fighting agents in Yellowstone National Park [Cohen *et al.*, 1989]. When an area of the forest catches fire, a fireboss directs a team of bulldozers to put it out. The fireboss is a domain-dependent, partial hierarchical planner (in PHOENIX terminology, a lazy skeletal-expansion planner) whose main task is to direct and coordinate the actions of the bulldozers, which dig fireline to surround each fire.

The fireboss faces a number of difficulties in carrying out its planning task. The fireboss cannot generate plans instantaneously, but must rather build them so that they will be effective at the time that they will be executed. The simulation, however, does not stand still. The environment can change rapidly, due to randomly shifting wind speed and direction; non-uniform terrain and elevation exacerbate the problem of predicting the behavior of a fire. Bulldozers take varying amounts of time to execute their primitive tasks, which means that the fireboss's initial estimates about how to allocate time and resources may turn out to be wrong. These and other issues make the fireboss's planning task complex—it often fails and must replan.

To handle the complexity of the task, the PHOENIX plan representation is richer than that of conventional planners. There are five components of the representation: plans, actions, execution methods, timelines, and timeline entries [Howe, 1993].

Actions: Actions cause events to happen, both actual events in the simulation and "cognitive events" in the processing of the fireboss. When the fireboss is informed that a new fire has been detected, an action `act-deal-with-new-fire` is activated.

Execution methods: A single action may execute in different ways; the action's execution methods give it different means of achieving its behavior under different

conditions. The execution method associated with `act-deal-with-new-fire` causes it to search through the plan library for a type of plan that can manage the current situation.

Plans: A plan is a composite action containing a schema of sub-actions to be executed, in sequence or in parallel.

Time-lines and time-line entries: The actions, plans, and execution methods of a planner are in many ways analogous to the rules of an expert system; time-lines and time-line entries are the planner's equivalent of working memory, storing dynamic relationships between plans and actions as they execute.

A significant contribution of PHOENIX is the view that a plan is not simply an annotated partial ordering of primitive actions. Plans are instead full-fledged objects in which control can be represented explicitly. This representation allows the planner to examine and manipulate partially elaborated and partially executed plans.

In the PHOENIX plan representation, plans and actions are frames. The type `plan` is a specialization of the type `action`. Because an action contains slots for local variables, activation predicates, priorities, rescheduling triggers, and so forth, a plan inherits these slots as well. This gives us, the designers of a plan library, a great deal of power. We can build a basic plan that has a special behavior and a set of specific properties; a plans that is a simple variation can be defined as a specialization of the basic plan, inheriting most of its necessary structure.

This notion of plans as first-class objects has been adopted by the AIDE planner and taken a step further. All of AIDE's planning structures (plans, goals, actions, focus points, and so forth) are full-fledged objects, which we have called control units, with inheritance of both structure and behavior. That is, in addition to inheriting the slots of a parent plan, a plan also inherits the parent's executable methods.

5.5.4 RESUN

RESUN is a control planner that operates in a blackboard-based interpretation framework [Carver and Lesser, 1993]. The standard blackboard architecture consists of a blackboard, a set of knowledge sources, and a control mechanism [Carver and Lesser, 1994]. The blackboard is a global database of data and potential partial solutions. Knowledge sources (KSs), which embody the problem-solving knowledge of the system, examine the state of the blackboard to update solutions appropriately. The control mechanism determines which KSs become active and the order in which they execution.

A common control mechanism is an agenda, which maintains a set of potentially active KSs. On each execution cycle, the KSs are evaluated and the top-rated KS selected for execution. This kind of evaluation phase traditionally relies on a monolithic function that uses criteria applicable to all possible actions [Davis, 1980]. The RESUN planner provides an alternative to the agenda-based approach. One of the goals in designing RESUN was to support the development of sophisticated control strategies for interpretation, strategies that potentially involve large amounts of context-specific information. RESUN is thus

- script-based: one can write complex, explicit control strategies in the plan language;
- incremental: plan generation and execution are interleaved;
- opportunistic: the planner's focus of attention shifts dynamically between decision points, depending on information gained during the planning process.

RESUN's plan language provides for sequencing, iteration, and conditionalization of subgoals, as well as more complex control constructs. The functionality of this language formed the basis for AIDE's plan language. AIDE uses a simplified syntax

and extends its functionality slightly in one area, by allowing in-place actions to be executed by a plan.

Carver and Lesser describe one of RESUN's main contributions as the notion of focusing and refocusing. Suppose that the planner finds two plausible ways to proceed in solving a problem, e.g., two plans match a subgoal. The RESUN planner can make a decision between the two choices based on local, context-sensitive information. This is focusing. RESUN may also opt to pursue both plans at once. Because planning is incremental, each course of action may turn up relevant information as the plans execute. This information can be used at later points during the planning process to prune one of the initial choices, if appropriate. This is refocusing.

AIDE provides a limited, single-thread version RESUN's focusing ability, with one difference. Once the RESUN planner enters into consideration a focus point, it must select one or more of the given possibilities to proceed. In contrast, the AIDE planner can select a possibility or decide not to make the decision at all, and turn its attention to a different focus point. This capability is essential to AIDE's mixed-initiative processing. When AIDE presents a decision to the user, it cannot *force* the user to select one of the given choices. The user, with far greater knowledge of context, may wish to consider an entirely different decision. The RESUN planner can manage this, but only in a cumbersome way in its current implementation. Because this behavior is required at almost every decision during the exploration, it made sense to design the AIDE planner to support it easily.

One of RESUN's applications is in the architecture of IPUS (Integrated Processing and Understanding of Signals) [Lesser *et al.*, 1995]. IPUS has as its goal the development of framework that combines formal signal processing theory with the idea of reprocessing. When, in the course of analyzing its data, an IPUS-based system encounters discrepancies between its expectations and its results, the data can be selectively reprocessed to diagnose the differences and explain them. In many ways AIDE can be viewed as an application of the IPUS architecture to the domain of EDA.

For example, if AIDE observes clusters in the residuals (x, r) of a linear fit of (x, y), its pursuit of these same clusters in (x, y) is a way of reprocessing the data. AIDE differs from IPUS, though, in its lack of formal models for exploration and its strong reliance on the user in deciding when to reprocess data.

5.5.5 Classical Planners

Modern classical planners, such as SIPE [Wilkins, 1988], O-PLAN [Currie and Tate, 1991], and PRODIGY [Carbonell *et al.*, 1992], have overcome many of the limitations of earlier classical planners. Their operators can perform functions on input variables, rather than simply matching; they often interleave plan generation and plan execution; they can take advantage of earlier planning experience to help solve new problems. Nevertheless, partial hierarchical planning is still preferable for its explicit representation of control in its plans.

A plan is sometimes described informally as a sequence of actions leading from an initial state to a goal state. We have been careful to avoid this exact phrasing. Consider the two plans in Figure 5.5.5. Plan A is typical of the plans produced by a classical planner, while Plan B could be generated or retrieved by a partial hierarchical planner. In many situations, perhaps even all situations, the two plans will be identical in their execution behavior. Nevertheless, there are significant differences between the two.

In EDA, as well as other domains, one often needs to know the decisions that led up to some point in order to evaluate a result. In other words, decisions about procedural combinations of operations must be represented explicitly. For example, if one is presented with a resistant line as a description of a relationship, it is reasonable to wonder why a least-squares line was not tried instead. The sequence of steps that led to the construction and rejection of the least-squares line are not part of the sequence leading to the resistant line; nevertheless it can be that the latter sequence

was chosen only after examination of other possibilities. For a result to be evaluated correctly, these possibilities need to be explicit in the executing plan.

The difference between Plan A and Plan B also has serious implications for performance. Think of a classical planner searching for a plan a hundred operators long, and contrast this with a partial hierarchical planner producing a "Repeat 100 times" form. This can mean the difference between finding a plan and very quickly running out of time or memory.

Classical planners rely on a relatively inexpressive notion of a plan as a sequence of operations. Case-based and derivation-replay planners [Hammond, 1986; Veloso and Carbonell, 1993] use libraries that contain annotated traces of planning sessions to develop new plans. Nevertheless, the resulting plans are still partially ordered sets, rather than the richer structure provided by explicit control constructs. The question of how classical planners can build plans with more expressive operators,

Plan A
(:ACTION Operator A)
(:ACTION Operator B)
(:ACTION Operator A)
(:ACTION Operator B)
(:ACTION Operator C)

Plan B
(:SEQUENCE
(:REPEAT *2 times*
(:ACTION Operator A)
(:ACTION Operator B)
(:IF *(some condition holds)*
then (:ACTION Operator C)
else (:ACTION Operator D))))

Figure 5.11. Two plans

including conditionals and iteration, is currently an open research problem. It seems likely that the areas of classical planning and partial hierarchical planning will soon become indistinguishable; at this point, however, there remain important differences.

5.5.6 Partial-Order Causal Link Planners

Classical planning issues have also been addressed by research in partial-order causal-link (POCL) planning. These planners use the classical representation of world states and operators, but rather than searching through intermediate world states, POCL planners search through a space of partial plans in their efforts to generate an appropriate sequence of actions. POCL planning algorithms (e.g. TWEAK [Chapman, 1987], SNLP [McAllester and Rosenblitt, 1991], UCPOP [Penberthy and Weld, 1992]) and hierarchical task network planners (e.g. HTM [Erol *et al.*, 1994]) have a significant advantage over classical systems: they are provably sound (i.e. they will generate no incorrect plans) and provably complete (i.e. they will eventually find a correct plan, if one exists).

Despite the attractive theoretical guarantees of POCL planning, which we can by no means provide for partial hierarchical planning, there are still reasons to prefer the latter approach in building a planner for EDA. The problems concerning the expressiveness of plans represented as sequences of actions apply here as well. Even if these were solved, it would be difficult to apply a POCL planner directly to the EDA domain. For example, soundness and completeness proofs generally rely on a view of planning as theorem proving. In some partial hierarchical planners, however, actions may deal with variables whose values vary along a continuous range; deciding that an action has succeeded cannot rely solely on whether it unifies with a given goal. The closed world assumption is an integral part of the planning-as-theorem-proving view, but it fails to hold for representations of many interesting domains, including EDA. The interleaving of plan generation and execution poses further problems for proving the soundness of a partial hierarchical planner. Any irreversible action taken before

a plan is complete can potentially render the plan incorrect; ensuring that the effects of an action are reversible can be as difficult as the planning problem itself. Finally, there are issues of the efficiency of plan generation and the perspicuity of the plan generation process. These points make POCL and related planners inappropriate for the domain of AIDE.

5.6 Summary

Exploration can be cast as a planning problem. EDA makes use of abstraction, problem decomposition, and procedural knowledge, three defining characteristics of planning. More specifically, AIDE's design exploits a striking similarity between interactive data exploration and partial hierarchical planning. A partial hierarchical planner has these properties: a plan library, which obviates the need to generate plans from scratch; hierarchical plans, which may contain subgoals, rather than explicit primitive operators; explicit planning control constructs, or procedural specifications for the way subgoals are to be satisfied; interleaved generation and execution; and sophisticated meta-level reasoning. Partial hierarchical planners were originally developed for complex, dynamic domains in which a planner must often act under time pressure.

This kind of planning is well-suited to exploration. First, EDA is reactive. One cannot anticipate every pattern that might possibly appear in the data; rather, the analysis is driven by the data, which argues for integrating the generation and execution of procedures. Second, exploratory procedures often need explicit control. Some common EDA techniques, like resistant line generation, smoothing, and lowess, are iterative, while other techniques need sequencing, conditionals, mapping, and other kinds of control. Finally, exploration is constructive. An exploratory result is not simply a graph or a statistical summary, but also includes a set of supporting decisions, which provide context for results to be interpreted.

AIDE plans as follows. When a dataset or relationship is presented to the system, a goal is established for its exploration. The planner searches through its library for an appropriate plan and expands it, that is, establishes a set of new subgoals to be satisfied. These subgoals are satisfied in turn by plans from the library. Because several plans may satisfy a single goal, the planner must rely on control rules to select the best possible choice. As planning continues, the planner may sometimes backtrack to one of these decisions to make a different selection. The process continues until the goal at the top level has been satisfied. In AIDE, exploration is a problem of constructing and navigating through a network of decisions. Execution of each primitive action generates one or more new results; the network of decisions and results combines all the findings.

CHAPTER 6

STATISTICAL STRATEGIES

A statistical strategy is a formal description of the actions and decisions involved in applying statistical tools to a problem [Hand, 1986]. AIDE's library contains a set of strategy components, in the form of plans, for common exploratory procedures, such as fitting lines, examining clusters, partitioning relationships, and so forth. These plans are activated within the context of other plans. In this context, decisions are made by activation and preference rules, which can explicitly represent relevant strategic considerations. The combination of these plans and rules in a specific context constitutes a statistical strategy.

AIDE's strategies and their components are taken from the literature of several fields: statistics, statistical expert systems, machine learning, and knowledge discovery in databases. Part of the challenge of building AIDE was to fit a set of such diverse techniques into a single framework. This chapter describes, in informal terms, the significant strategies in AIDE's plan library and explains how they are combined. A few representative examples of these strategies, in the syntax of AIDE's plan language, are given in Appendix B.

6.1 Strategies for Fitting Lines

A common approach to describing patterns in data is to build linear models of relationships. In the simplest case, this means fitting a line to a relationship between two variables. AIDE's plans include a variety of techniques for fitting lines. There is the simple regression line, which minimizes the least squares error between each y value and predicted $\hat{y}$ value, and can be effective when the relationship is

"well-behaved." In other cases, AIDE can apply resistant line fitting procedures, which give better results when outliers are present or when the variables are not distributed smoothly. Yet another possibility is a weighted regression line, a variation on the basic least-squares procedure in which outlying data points are downweighted to reduce their influence on the linear parameters.

These procedures all generate linear descriptions of bivariate relationships. The regression-based techniques can be generalized for multivariate relationships, but this possibility has received only cursory attention in AIDE. Fitting lines is a type of modeling, which involves a variety of assumptions about the data. These assumptions are relatively easy to test in a simple regression between two variables, but, as we add variables, problems grow harder to detect. Often in practice one makes assumptions without the ability to test them. When exploring data it is useful to make as few assumptions as possible, so that unexpected findings can more easily come to light. AIDE concentrates on the simpler techniques.

6.1.1 A Simple Regression Strategy

An unweighted least-squares linear fit of a bivariate relationship is the most straightforward fitting procedure implemented. Given a relationship (x, y), the plan `regression-fit` fits a line

$$\hat{y} = ax + b$$

that minimizes the sum of squared residuals,

$$\sum_i (\hat{y} - y)^2.$$

The plan has two parts, the first part computing the parameters of the line, the second generating information about the adequacy of the line as a description of the relationship. The residuals r are generated and explored (a) as a batch of numerical values, (b) combined with the original x values, and (c) combined with the new $\hat{y}$ values. The leverage of the each x value is also examined. Examples of each of

these results, for the relationship (FirelineBuilt, Duration) from the PHOENIX data, are shown in Figure 6.1. Ideally, a linear fit to a relationship should give residuals that show no obvious patterns; that is, if the line captures all of the structure in the data, then what is left should be only noise. In the plot of residual observations in Figure 6.1(a), however, one can easily see outliers. In Figure 6.1(b), which plots the residuals by the original x variable, the variance of the residuals increase as x increases, possibly indicating a more complex relationship between the two variables. Plotting y/x by x could remove this pattern, but this improved description would have to be balanced against the increased complexity of interpreting y/x in real-world terms. Note also that there seems to be a slight downward slope to the residuals, when they should be completely flat. Figure 6.1(c) plots the residuals by the predicted response, $\hat{y}$. This is identical to Figure 6.1(b), because there is only one predictor variable, but such a plot can give useful information for a multiple regression model. Finally, the leverage relationship in Figure 6.1(d) plots the influence each point has on the slope of the line. We see that leverage is distributed unevenly, that points with higher values of x are much more influential than those with lower values.[1] Each of these residual diagnostic plots is generated and explored by a different plan.

What we call a regression strategy is then the `regression-fit` plan, the activation and preference rules that determine when it is applicable, and the subordinate plans that compute the linear fit parameters and explore the residuals. By separating the phases of a strategy in this way we can often generalize components for reuse.

[1]The leverage of a point x_i is equal to

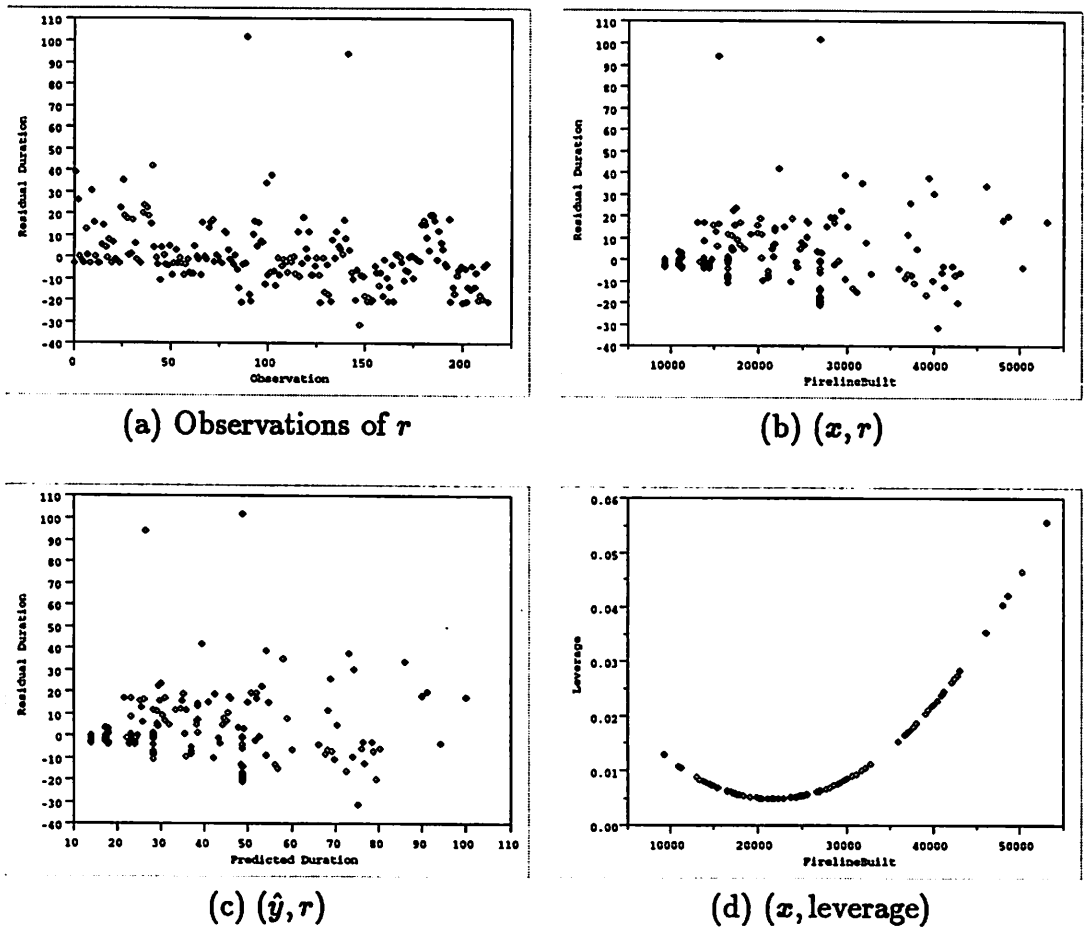
$$\frac{(x_i - \bar{x})^2}{\sum_j (x_j - \bar{x})^2} + \frac{1}{n}.$$

In other words, the amount of influence a point (x_i, y_i) has on the slope and intercept of a regression line is proportional to the squared difference between x_i and $\bar{x}$. This is a reasonable diagnostic measure for lines based on least squares computations [Goodall, 1983].

6.1.2 A Weighted Regression Strategy

The `weighted-regression-fit` plan is closely related to the `regression-fit` plan just described. As before, the parameters of the line minimize the sum of squared residuals. Recognizing that some values may be overly influential on the parameters of the line, however, this strategy weights the contribution of each point to the equation. Because there are many different ways of determining outlying data points, there are many possible weighted regression lines.

The `weighted-regression-fit` plan maintains a variable `?weights`. For each distinct indication of outliers, `?weights` is bound to a different value, the bindings being maintained by a variable focus point. (See Chapter 5 for a discussion of focus



(a) Observations of r

(b) (x, r)

(c) $(\hat{y}, r)$

(d) $(x, \text{leverage})$

Figure 6.1. Regression residual plots

points.) Outlying points are given a weight of zero, all other points a weight of one. For example, if AIDE determines that y has three outliers using a resistant fourth-spread test,[2] `?weights` will be bound to a list of ones and three zeroes in appropriate positions. The variable focus point can maintain several instantiations of the same plan, one for each possible weighting. Otherwise the weighted regression plan is identical to the unweighted plan.

6.1.3 A Resistant Line Strategy

A resistant line is similar to a weighted regression line in that both techniques are aimed at reducing the effects of an uneven distribution of data points, especially outliers, on the parameters of the line. The resistant line procedure differs from least-squares procedures, however, in a fundamental way. A resistant line fitting procedure adds resistance to its estimates essentially by ignoring data. This is not unusual. The median, for example, is a resistant estimator of location in that extremely large or small outlying values are ignored in its calculation. In fact, only the central value (or the mean of two central values, for batches of even size) is considered at all. The three group resistant line [Emerson and Hoaglin, 1983] relies on the resistance provided by medians. It breaks a relationship (x, y) into three partitions of equal size, using the tertiles (33% quantiles) of the x values. Medians are computed for each partition: $\langle x_L, y_L \rangle$, $\langle x_M, y_M \rangle$, and $\langle x_H, y_H \rangle$. The slope of the resistant line is then determined by $\langle x_L, y_L \rangle$ and $\langle x_H, y_H \rangle$, while the intercept is adjusted upward or downward by $\langle x_M, y_M \rangle$.

Notice that if we were to fit a regression line to a relationship (x, y), extract the residuals r, and then fit another regression line to the relationship (x, r), this new line would have zero slope. This is intuitively reasonable: We expect such a description

[2]The fourth-spread test uses the 25% and 75% quantiles of a variable (f_L and f_H) as boundaries. If an observation is greater than $f_H + 1.5(f_H - f_L)$ or less than $f_L - 1.5(f_H - f_L)$ it is considered an outlier. The fourth-spread test is resistant in that a few very large or very small observations will not change its performance. The same is not necessarily true for tests that use classical statistics, such as the standard deviation, where the size of outlying observation can affect thresholds values.

of a relationship as a line, to capture all the "linearity" present in the data. This reasoning applies to resistant lines as well. The three group procedure, however, is not guaranteed to generate residuals with zero slope after a single application. While least-squares procedures usually generate a unique fit, this is often not the case for resistant procedures, which may rely on statistics (such as the median) that do not always generate unique results. Thus we must iterate the three group procedure, adjusting the slope and intercept of the fitted line, until the residuals pass a heuristic test for "flatness".

The `resistant-fit` plan implements the three group procedure, using an iteration construct and conditionals to iterate the fit and test for flatness of residuals. Except for the iteration in the computation of the linear parameters, its structure is similar to that of the least-squares fitting plans. The same procedures for exploring the residuals of least-squares lines are applicable here as well, except for the examination of leverage. The notion of leverage is useful when the influence of a point depends on its squared residual error. Because we have dispensed with the least-squares approach in favor of a more resistant technique, leverage is irrelevant.

6.1.4 Control

Activation rules trigger a `linear-fit` plan to describe a relationship if its correlation is above a heuristic threshold. This plan creates the context for the more specific plans described above. It immediately establishes a goal to select an appropriate type of linear fit. Preference rules examine the relationship for x and y outliers. If there are outliers present, the resistant plan is selected; otherwise the unweighted regression plan is used. A weighted regression line requires a weighting vector for each data point, which in the current implementation means identifying specific points as outliers. Because this identification can depend on human judgment, the weighted regression plan must be explicitly selected by the user, rather than by AIDE's activation rules.

Rules thus activate a specific linear fitting plan, which is then executed. In its execution, the plan triggers a subordinate plan to generate the linear fit parameters followed by several potentially relevant residual exploration plans. In the normal course of execution, the user is shown a list of options at each decision point, along with AIDE's preference. At the level of exploring a relationship this might consist of the plans `linear-fit` and `decompose-clusters`, with a preference for fitting a line. The user can then make a selection or simply indicate that AIDE should proceed. In some situations, however, we want AIDE to carry out a series of decisions on its own. This is the case in the exploration of residuals. Rather than asking the user, "Should residuals be examined?" and "Should a leverage plot be generated?" AIDE should execute these plans automatically and display any indications it finds in carrying out the process.

A meta-plan applicable to all linear fit plans handles this responsibility. Recall that the meta-planner is activated whenever a focusing decision arises. In this case, the decision involves which residual examination plan to execute. Rather than present this decision immediately to the user, the linear fit meta-plan activates the first residual plan, executes its initial feature generation and indication generation actions, and then backtracks to execute the next residual plan. Only when all the residual plans have been partially executed, does the meta-planner return control to the user, displaying the possible residual examination plans. Because the plans have already been partially executed, information can be presented along with each plan describing the results generated.

6.2 Straightening Strategies

Strategies for "straightening" are designed to remove the effects of higher-order functions on bivariate relationships. What remains is a relationship that can be examined with other techniques, such as linear fitting. There are two simple strategies in AIDE, both derived from early work by Tukey [Tukey, 1977].

The first approach is applicable when the distribution of one or both of the variables in a relationship (x, y) is highly skewed. It involves simply performing a log transform on the skewed variables, giving a result such as $(\log x, y)$ for further exploration. The `skew-reducing-transform` plan implements this procedure.

The second approach involves the implicit fitting of a power function to the relationship. The procedure first measures the curvature of the relationship, based on three representative points (as is done in generating a resistant line.) It then partitions the relationship, using the $n/8$, $n/2$, and $7n/8$ order statistics of x, and computes the median y of each partition. These three points are connected by lines whose slopes can then be compared. If the ratio of the two slopes is close to unity, then the relationship is considered straight; otherwise it is curved. Depending on the sign and magnitude of the ratio, the procedure follow a "ladder of transformations": y, $\log y$, $y^{1/2}$, $y^{1/3}$, ...; and y, y^2, y^3, ... Taking reciprocals gives us an additional dimension. The procedure follows the ladder by applying the appropriate transformation and checking the result for curvature. Once the slopes of the lines are sufficiently close, the process ends.

The `linearizing-transform` plan implements this second straightening procedure. It uses an iteration construct and conditionals to manage the transformation.

6.2.1 Control

The `linearizing-transform` plan is activated by rules sensitive to the presence of the three point curvature indication described above. This indication, a variant on one devised by Tukey [Tukey, 1977] can often be effective in identifying and transforming power relationships. Unfortunately, it is not very reliable, being easily misled in comparison with human judgment.

A straightening procedure is generally used in conjunction with a linear fitting procedure. That is, curvature is removed from a relationship to give a linear fitting procedure a better chance to extract still more structure. Thus, as with the meta-plan

for fitting lines, a meta-plan for straightening also combines otherwise independent plans. As its last action the `linearizing-transform` plan establishes a goal to explore the transformed relationship. This new relationship may cause the activation of plans to partition it, to transform it in a different way, to reduce it, or other possibilities. All these, however, are ignored by the straightening meta-plan, which selects the `linear-fit` plan without interaction from the user. This meta-plan coordinates with the linear fitting meta-plan, so that several autonomous steps are taken at one time by the system. If the user wants to return to the transformed relationship to apply other operations, that is also easily done.

6.2.2 Related Work

AIDE's straightening plan is comparable to function-finding algorithms in machine learning research. These algorithms build arithmetic formulas that functionally characterize the behavior of a set of variables. The term "function-finding" was invented by Cullen Schaffer [Schaffer, 1990] to describe the behavior of scientific discovery systems such as BACON [Langley *et al.*, 1987], Fahrenheit [Zytkow, 1987], ABACUS [Falkenhainer and Michalski, 1986], and KEDS [Nordhausen and Langley, 1990]. These systems generate new variables through pairwise functional combinations of existing variables, using the operations of addition, subtraction, multiplication, and division. Extensions to the basic approach include partitioning by categorical variables to find different relationships that hold in different subsets of the data.

These systems are sometimes described as performing autonomous machine discovery. However, their methods of evaluation, especially those of BACON, have been severely criticized [Schaffer, 1990]. BACON calls a functional relationship correct if a series of transformations result in a variable with "constant" values. It is difficult, if not impossible, to determine in general that a set of values is "approximate constant;" the judgment clearly depends on the range of values in the data, the range of values considered theoretically plausible, and other concerns. Schaffer showed that BACON

can behave quite badly when presented with real data. Schaffer's system E*, in contrast, treats the discovery of bivariate functional relationships as a classification problem, and bases its judgment of the best description on a comparison with plausible alternatives. Schaffer's heuristics, though domain-independent in one sense, were tested on bivariate relationships largely with correlations above 0.9, which makes them less useful for general applications.

AIDE's heuristics for straightening, though straightforward, are comparable in sophistication to those of most of the systems mentioned. AIDE's advantage lies mainly in giving the user the opportunity to review and modify its results.

6.3 Reducing Transformations

Sometimes when examining a scatter plot of a relationship we find it it difficult to distinguish a pattern from background noise. One solution to this problem is suggested by Tukey and Tukey [Tukey and Tukey, 1981]: we can "sharpen" a plot by removing points from low-density areas. AIDE uses a variation on this approach, similar to techniques for fitting resistant lines. A relationship (x, y) is broken into a number of subsets, derived by partitioning the x variable. Each subset is summarized by its median x and y values. These medians are then described by appropriate strategies: by a straightening strategy, a linear fit strategy, a clustering strategy, or some other technique.

The `reduce-relationship` plan is one example of a sharpening strategy. It partitions a relationship by binning its values into subdivisions of equal size on the x axis. The number of bins is heuristically selected, in a way similar to one used by histogramming procedures. A related `reduce-clustered-relationship` plan relies on indications of clusters in the data for partitioning, rather than reducing data by arbitrary binning. Both plans rely in a straightforward way on primitive decomposition, reduction, and transformation operations.

6.3.1 Control

Reduction plans act as a catchall for describing relationships. If no other plans are activated that might describe the relationship as a line, a power function, a set of clusters, or some other possibility, then these plans reduce the relationship to a set of representative points that might be easier to analyze. Their effectiveness is limited, however; it is easy to find a pattern where none exists simply by removing data points that surround an arbitrary pattern.

6.4 Clustering and Classification

When clustering is detected in a variable or relationship, AIDE takes two courses of action: first, attempting to predict the assignment of data points to clusters; second, to explore the behavior of individual clusters. Prediction relies on a modified version of ID3 [Quinlan, 1993], which builds a decision tree from other variables in the dataset. If the predictive accuracy of the tree is high, and if the tree contains only a few variables, then these are considered good predictors of the clustering. Exploration of individual clusters is simply the application of AIDE's standard techniques for exploring all relationships.

The `generate-clusters` plan establishes the context for the exploration of a relationship or variable R by several subordinate plans, `predict-clusters`, `predict-local-clustering`, `similar-clusters`, and `explore-subsets`. The first three subordinate plans all have a similar structure, but attempt to predict different aspects of the clustering in R. The `predict-clusters` plan uses the clustering to generate a categorical variable containing the classes of R, and builds a decision tree to predict these classes using any categorical variables in the dataset. The `similar-clusters` plan is comparable, but instead of using categorical variables it uses clusters detected in other variables and relationships in its prediction. The `predict-local-clustering` plan is activated when the clustering behavior does not encompass all the data points in R. We saw an example of this in the vertical clusters of the (FirelineBuilt, Duration)

relationship in the PHOENIX dataset. The `predict-local-clustering` plan builds a decision tree that predicts two classes, points that fall into any cluster, and points that do not. Finally, the `explore-subsets` plan uses a mapping control construct to explore each of the subsets generated by the clustering.

6.4.1 Control

There are half a dozen indications related to clustering in multivariate relationships. Indications may detect "gaps" in the distribution of a variable, recurring values in a continuous distribution, or localized clusters by other heuristic methods. Rules incorporating these indications activate the top-level `generate-clusters` plan.

As with the examination of residuals in the context of linear fitting, there are some actions AIDE should always take when exploring clusters. When `generate-clusters` executes, a meta-plan causes subordinate plans for predicting the clustering behavior to execute automatically, with no user interaction. The decision to explore clusters individually is left to the user. When control is returned to the user, the results of the prediction plans are presented in summarized form. For example, the `predict-clusters` plan could be annotated with the intermediate result, "Best prediction accuracy of 0.80 by variable x." These annotations can help the user decide stop at a given point or to pursue possible connections with other variables.

6.4.2 Related Work

Clustering in AIDE relies mainly on traditional numerical techniques, in particular single linkage clustering algorithms [Sibson, 1973; Everitt, 1993]. Single linkage clustering is a recursive technique that begins by assigning each data point to a separate cluster. The two clusters closest together are then merged, where closeness is determined by the shortest distance between two points in separate clusters (trivial in the one-point-per-cluster case.) The process continues until all clusters have been merged, resulting in a hierarchy of increasingly finer clusters. At higher levels in

the hierarchy the distances between clusters are larger, while the lower levels contain more tightly clustered points. Heuristically chosen values then cut the hierarchy at different points to partition the data. By choosing different criteria for determining cut values, we can generate sets of clusters with different characteristics.

Numerical algorithms have limitations in their use of a distance metric to cluster observations. For example, imagine two sets of points, one set forming the outline of a circle, the other a superimposed square. A numerical clustering algorithm would have serious difficulty coming up with a meaningful description of the groups, while for us it would be easy to generate a new attribute, attached to each observation, taking on two possible values, "point on a square" or "point on a circle." Conceptual clustering algorithms were developed with this kind of goal in mind. They attempt to take global, gestalt properties of observation classes into account in an attempt to capture general notions of similarity [Michalski and Stepp, 1983]. Unfortunately, current clustering algorithms (e.g. COBWEB [Fisher, 1987], CLASSIT [Gennari *et al.*, 1989], ITERATE [Biswas *et al.*, 1991; Biswas *et al.*, 1994]) are not able to generate descriptions like "point on a circle," instead concentrating mainly on grouping observations consisting of categorical variables.

General-purpose autonomous clustering is a very difficult problem [Everitt, 1993], partly due to the importance of contextual knowledge. Research in human analogy-making has shown that context can play a large part in determining similarities [Gentner, 1983]. Tversky [Tversky, 1977] points out that while machine learning heuristics can capture similarities of objects given some fixed set of features, the remaining problem of selecting this set is usually left unaddressed. Evaluation of clusters in AIDE is relatively weak, relying largely on the user to interpret the importance of the clusters.

6.5 Modeling

AIDE can construct several different types of models for a dataset. While exploratory techniques are often aimed at generating descriptions of low-dimensional projections of a dataset, such as single variables and bivariate relationships, modeling procedures can give a broader picture of the data. They generally do this by making assumptions about the data and the type of result desired. Regression modeling algorithms sometimes assume, for example, that a single linear model can adequately describe the data, rather than several distinct models applying to different, possibly overlapping subsets of the data. Causal modeling algorithms usually assume that individual variables should be interpreted as causes; they will not generate a new variable from x and y, for example, in order to say that the ratio x/y predicts z. Clustering algorithms often do not generate functional transformations of their input data: such actions can distort "natural" cluster behavior, making user interpretation an important issue [Duda and Hart, 1973].

AIDE's plan library was designed with an iterative approach to modeling in mind, where modeling actions are interleaved with exploratory actions so that each kind of action may inform the other. The assumptions made by a specific modeling algorithm can be tested by exploratory activities, while modeling results can guide exploration in appropriate directions. AIDE's library contains plans for four types of models. In the current implementation, AIDE's activation and preference rules specify a single modeling plan as the default for all datasets. The user can select another set of modeling plans as more appropriate, if desired, but AIDE's rules do not assist in this decision.

6.5.1 A Causal Modeling Plan

AIDE's causal modeling is based on Pearl's IC [Pearl and Verma, 1991] and Spirtes et al.'s PC [Spirtes *et al.*, 1993] algorithms; in earlier implementations of the AIDE plan language, Cohen's FBD [Cohen *et al.*, 1996] was incorporated as well. The causal

modeling plan currently implemented in AIDE is most similar to PC. PC, like many causal modeling algorithms, begins by assuming that all variables are related, and proceeds by eliminating those relationships determined to be indirect. This means that it constructs a complete graph and then iteratively removes arcs until the "true" model remains. Causal modeling is inherently exponential, because two variables may be screened off by any number of other variables in the graph; in the worst case an algorithm must test the power set of $n - 2$ variables for all pairs of variables in the graph. The iteration can be managed in an efficient way, however, so that when the true model is sparse, the algorithm terminates quickly.

The key to algorithms like PC is deciding whether variables a and b are conditionally independent, i.e., whether changes to a can affect b, if we hold some set of other variables constant. The general approach in testing for conditional independence is to use a G-test or χ^2 test for categorical variables and a test of partial correlation for continuous variables. Two variables are conditionally independent if the result of the partial correlation test, or the appropriate categorical test, is not significant. The set of variables that gives this independence is called the separating set; as an efficiency measure, algorithms may impose a fixed limit on the size of any separating set.

The logic of the partial correlation test is as follows. Fisher's r-to-z transform tests whether the partial correlation of two variables a and b, with a set of variables c held constant, is significant at a given level (e.g., $p = 0.05$). It works by transforming their correlation (r) into a normal score (z), which we can then easily evaluate. Large values of the test result indicate that a partial correlation is significantly different from zero. If the test does *not* establish that the partial is significant, then we say that the variables are conditionally independent. That is, if a and b really are related, then they ought to be significantly correlated regardless of what we hold constant in set c. So to build a causal model, we choose a confidence level and generate a graph based on our conditional independence calculations. Note that with this test we can only

draw tentative conclusions about causal relationships: there may be many reasons that a partial correlation is not significant, but our assumption is that it is due to real conditional independence.

What happens if we change the level at which we consider a partial correlation significant? Suppose we have a significance level of 0.1, and find no variables rendering the two variables of interest, *a* and *b*, conditionally independent. Suppose now that we lower this level to 0.05—suddenly we have conditional independence. This simply means that we have used a more stringent criterion in judging whether *a* and *b* are related. Under this more stringent criterion, they cannot be assumed to be directly related. Thus, in building a causal model, if we want more links, we lower the confidence (we relax it); if we want fewer, we raise the confidence (we tighten it).

The logic of the G-test and χ^2 test is similar to the partial correlation test. We compute whether two categorical variables are conditionally independent by collapsing their relationships with other categorical variables into a two way table, and then testing for independence [Wickens, 1989]. Like the partial correlation tests, this approach has its set of pitfalls, and the same caveats hold about conditional independence relationships being tentative.

These heuristics for testing conditional independence are the only point of special interest in AIDE's `build-causal-model` plan. The plan can be divided into three broad stages. The first builds a complete graph (in matrix form); the second iteratively removes edges with conditional independence actions; the third actually generates the remaining relationships (using information from the matrix) and provides for their exploration.

6.5.2 A Cluster Modeling Plan

AIDE's cluster modeling plan is based on an algorithm developed by Fowlkes, Gnanadesikan, and Kettenring for hierarchical cluster models [Fowlkes *et al.*, 1987]. In contrast to AIDE's basic clustering plan, which simply describes clustering in a single

relationship, a cluster modeling procedure attempts to describe an entire dataset, possibly incrementally, in terms of its clustering behavior. The cluster modeling plan implemented in AIDE is an example of a forward selection algorithm. Forward selection algorithms build a model of a set of variables by starting with a model that includes just one variable, then considering models with two variables, and so forth, until a stopping criterion is met. At each stage a new model is selected as the "best" model, and may form the basis for evaluation of further models. The best n variable model may or may not contain the $n-1$ variables of the previous best model; this depends on the algorithm. Stepwise linear regression is a familiar example of a forward selection algorithm [Weisberg, 1985].

Forward selection algorithms depend on two types of decisions: whether one model is better than another, and when the algorithm should stop and return the current best model. (In stepwise regression, model evaluation is usually based on an adjusted R^2, and the algorithm continues as long as individual variables are above the "F-to-enter" threshold.) In the case of AIDE's cluster models, evaluation is based on an assignment of points into K clusters, K varying over $1, 2, \ldots, K_{max} = 8$. Each assignment is evaluated by

$$S^d(K) = S(K) - E(S(K)),$$

where

$$S(K) = tr({T^{-1}}_p B_p)/p \qquad \text{and} \qquad B_p = T_p - W_p,$$

W_p and T_p being the within group and total sums of cross-product matrices respectively, tr denoting the trace statistic. In words, $S(K)$ gives a single statistic summarizing how well a specific assignment of individual points to clusters describes the distribution of the data. This summary is based on the variance in data internal to each cluster and the variance of all the data taken together. We can measure the "goodness" of a specific assignment of points to clusters by comparing it to a

set of random assignments. We essentially construct informal confidence intervals by empirically computing $E(S(K))$, the null expected value, by Monte Carlo sampling. If an assignment gives a sufficiently high $S^d(K)$, we select that assignment to break our data into clusters. If there are no high values of $S^d(K)$ at some iteration, the process terminates with the current best model.

The effect of the `build-cluster-model` plan is to generate a hierarchical cluster model, which in AIDE's data representation is a tree of increasingly finer partitions. The partitioning criterion at each stage j is based on the clusters in a set of j variables. The variables included at each stage are furthermore a superset of the variables of each previous stage. The partitions can be explored at each stage for structure that may not be captured by the clustering descriptions.

6.5.3 A Multiple Regression Modeling Plan

The `build-multiple-regression-model` plan in AIDE's current library simply computes a regression model for all variables specified by the user, and establishes a focus point to manage the exploration of all direct predictive relationships (i.e., between each predictor variable and the response variable) and all relationships between the predictor variables. This serves to establish context for interpreting potential patterns, such as collinearity, but there is no sophisticated modeling involved.

In an earlier version of the plan library, a forward selection algorithm [Fowlkes *et al.*, 1987] was implemented. As with the cluster modeling plan, the regression model is built one variable at a time. Evaluation is based on Mallows's C_p measure [Mallows, 1973], or

$$C_p = \frac{RSS_p}{s^2} - (N - 2p),$$

where RSS_p is the residual sum of squares of a regression fit using the current set of variables, s^2 an estimate of the variance (here the mean square for the complete equation using all variables), p the number of variables and N the number of observations. Sets of predictors for which C_p is close to p are considered good.

Again in an earlier version of the plan library, a strategy similar to one developed by Gale and Pregibon [Gale, 1986] was implemented. Section 6.6.2 contains a description of the strategy.

6.5.4 A Default Modeling Plan

The default modeling plan acts mainly to establish a focus point through which variables and relationships can be iteratively examined. This, the weakest modeling strategy in the library, provides little context for interpreting results. It rather serves as a framework for the incremental extension of descriptions.

6.6 Related Strategies

The strategies described in this section are relatively sophisticated compared to those implemented in AIDE. This is partly because they are standalone applications, and partly because they address more specific data analysis problems in detail. Portions of these strategies have been implemented in AIDE to test the effectiveness of the planning representation. Much has not been implemented, however, due to the implementation effort and computational cost of running these highly specific algorithms in a general-purpose system.

Note that conventional modeling algorithms are usually tools rather than strategies. Suppose we have a classification algorithm that performs well in selecting variables, computing predictive power, and so forth, given appropriate input data. When we give this algorithm a medical dataset, though, it tells us that a specific condition can be predicted with perfect accuracy by a patient's social security number. The classification procedure lacks necessary strategic knowledge about its own application. All of its assumptions about the data and its domain of application are implicit, and it has no internal way of dealing with deviations from them. Comparable observations hold for regression algorithms, causal modeling algorithms, clustering algorithms, and other types of modeling. The development of a strategy can involve significantly

more work than designing an algorithm, and there are relatively few in the statistical literature.

6.6.1 Regression: Daniel and Woods

The earliest and probably best-known strategy for data analysis is due to Daniel and Woods [Daniel and Wood, 1980]. Their approach entailed close human supervision over the process, with computer programs used in detailed searches for appropriate equations. The authors comment that for the set of fifteen problems they considered, they searched about 5400 equations—an impressive number for 1971.

1. Objectives
    - Estimate effects
    - Predict responses
2. Preliminary examination of data
    - Ordering in space or time
        - Sort and list by magnitude of independent variables
        - Locate clusters for estimates of error
    - Plotting
        - Factor space
        - Time sequence
3. Construct full equation
    - Past experience: Engineering, physics, and chemistry
    - Past experience: Analytical, measurement, and process error
4. Linear least squares fit on full equation: Check results for...
    - entry errors
    - outliers
    - forgotten factors
    - appropriateness of functional forms
    - nesting
    - relations among independent variables
    - normality of residuals
    - residual error and compare with previous estimate

Figure 6.2. A partial strategy for regression (Daniel and Woods)

The first four steps of the strategy are given in Figure 6.2. The full strategy is about fifteen steps, including conditional branches and iteration.

6.6.2 Regression: Gale and Pregibon

A portion of REX's strategy is given in Figure 6.3. Gale and Pregibon [Gale, 1986] believe that the strategy is complete for simple linear regression and that it gives reasonable results for multiple regression. That is, it considers all the important factors in ensuring that the model generated will have no serious flaws. The strategy involves checking the response variable for specific properties that may need to be repaired, such as granular distribution, negative weighting, skewness, and so forth. Predictor variables are then checked individually for atypical points, nonlinear influence on the response variable, and other possible problems. Once all problems have been repaired (if possible) and the regression computation has been carried out, the residuals are examined for further problems.

```
regress: we can regress y or x meaningfully
  active: there is a problem in the regression
    checkdata:there is a single variable problem
      checkw: one or more weights are negative
        wdelete: observations with negative weights should be deleted
      check missing: data values are missing
        reinitialize: observations with missing data values must be deleted
      checky: there are irregularities in y
        ygranularity: the distribution of y is granular
          expert: human expertise is required to complete the analysis
        yspacing: the successive y values are equispaced
          expert: human expertise is required to complete the analysis
      ...
      checkX: there are irregularities in X
        xiterate: there is another column of X
                ...
```

Figure 6.3. A partial strategy for regression (Gale and Pregibon)

6.6.3 Regression: Faraway

A regression model is often the result of several stages of analysis, involving many different decisions. If we make predictions using the model, without regard for these decisions, our inferences may be biased. Faraway's RAT (Regression Analysis Tool) attempts to account for model selection decisions in generating a regression model [Faraway, 1992]. RAT does not contain a strategy, but rather the components necessary for a strategy, as shown in Figure 6.4. These components check for problems and repair them if they can.

The ordering of these activities is chosen by the user. The list is not exhaustive, but typical of the kinds of data-analytic actions that occur in practice. These are the types of actions included in AIDE's strategies.

6.6.4 Other Strategies

A few other strategies have been published in the literature of statistical expert systems, notably one for multiple analysis of variance (MANOVA) [Hand, 1986] and one for collinearity analysis [Oldford and Peters, 1986]. Even after more than ten years of research, however, there remain many open problems. As Faraway writes [Faraway, 1994, p. 213], "Production of [regression strategy] systems for the automatic analysis

Check for skewness and transform if necessary.
Check and remove outliers.
Check and remove influential points.
Check for nonconstant variance and reweight if necessary.
Check for a Box-Cox transform on the response.
Check for necessary power transformations of predictors.
Perform variable selection by backward elimination.
Restore previously excluded points.

Figure 6.4. Strategy components for regression (Faraway)

of data has been stalled primarily by the difficulty of integrating the real-world context of the data." This is a problem we have attempted to address with AIDE.

6.7 Summary

AIDE represents statistical strategies in the form of sets of plans and control rules. These plans may share component subordinate plans, which may be distributed across strategies. Control rules for activating and imposing preferences on plans may be similarly general. Thus, for example, AIDE's regression strategy shares a great deal of structure with its resistant line fitting strategy; structure is shared at lower levels of abstraction between other, less obviously related strategies.

AIDE's strategies can be roughly divided into the following categories: fitting lines, straightening curved relationships, transforming by reduction, clustering and classification, and modeling. While these strategies tend to be simple in comparison with others in the EDA literature, in AIDE they are combined in a single representational framework. In addition to carrying out specific strategies for fitting lines or examining clusters, the system can also take over some of the responsibility for choosing these courses of action, rather than leaving all high-level decisions entirely up to the user. The specific strategies implemented in AIDE are central to the system's abilities to act autonomously as well as to accommodate the user.

CHAPTER 7

COLLABORATIVE EDA

Our discussion so far has taken a system designer's view of EDA: formulating the problem, designing a solution, developing an implementation. Our consideration of user involvement, an essential aspect of EDA, has remained implicit. In this chapter we explain how the concepts and representations developed up to this point help users in their data exploration.

In the user interface the tradeoff between autonomy and accommodation stands out clearly. AIDE is designed to look and behave much like a conventional statistical interface on the outside. With a full set of statistical operations AIDE supports the conventional exploratory cycle of examining a dataset, selecting and applying statistical operations, examining the results, and repeating the process. User activities are thus accommodated: it is possible for the user to ignore AIDE's suggestions and commentary entirely, treating the system as a passive, conventional system. AIDE's plan representation nevertheless gives it limited autonomy, the ability to make decisions and act in a reasonable way on its own.

The tradeoff between autonomy and accommodation is one of several issues in the larger picture of mixed-initiative planning. Researchers have established criteria by which we can judge whether a planner can be considered a mixed-initiative system, rather than simply a user interface that supports plan construction, or an autonomous planner that accepts user advice. AIDE meets most of these criteria, within the statistical user interface framework. Human-computer interaction research has also identified necessary properties of collaborative systems; AIDE can be considered to some extent a collaborative system, again within the constraints of its domain.

This chapter is divided into three parts. We first discuss the general concepts that arise in mixed-initiative planning for data analysis and explain how the planning representation makes the intelligent aspects of AIDE's behavior possible. We then illustrate AIDE's behavior with an example and extensive commentary. Finally, we show where AIDE fits in the framework of data exploration systems that concentrate on human-computer interaction issues.

7.1 Concepts

Let's revisit our notion of an assistant, by way of an analogy. Imagine that you're a carpenter, with a business expanding faster than you can handle. New machinery will help, specialized equipment that lets you do particular jobs more easily. Farming out some of the work to other shops can help too; given detailed instructions, subcontractors can do much of the work without involving you at all. Still you have too much to do. You need an assistant: someone who can use the same tools you do, without direct supervision, but who can also follow your explicit instructions when reaching unforeseen difficulties in the work. An assistant can act sometimes as an extension of your own intentions (as a tool), and sometimes as an independent, autonomous agent.

Conventional systems for statistical analysis tend to be exclusively tools or agents. AIDE is more flexible, leaning toward one role or the other, depending on the situation and its own abilities. AIDE's plans implement the step-by-step procedures required of an assistant. Most plans contain internal decision points, or focus points, about which course of action should be taken in which situation. The system acts autonomously when it handles these decisions on its own, but it behaves more like a tool when it presents choices to the user for directions about how to proceed.

Maintaining the balance between autonomy and accommodation is harder than might appear at first glance. If the system can take actions without consulting the user, it can as a natural consequence drive the analysis into inappropriate areas,

possibly losing the user in the process. Similarly, the user is not constrained to take only those actions the system finds reasonable; on the contrary, with better visual pattern-detection abilities and knowledge of what variable values actually mean, the user should be able to take the analysis in whichever direction appears appropriate. The system must nevertheless be able to follow in the user's path, ready to contribute to the process once it again reaches a familiar area.

AIDE's planning representation alleviates this problem. Plans attempt to implement common statistical practice, relying implicitly on the user's knowledge of what actions are appropriate in a given situation. Thus if the system takes the initiative by executing a sequence of actions without consultation, the result will rarely be completely incomprehensible to the user. In the worst case, the user will think, "Why did the system choose to take that action in this situation?" while in the best case the response will be, "That's exactly what I would have done."—or, equivalently, no response at all, as the user accepts AIDE's activity without thinking about it. The examination of residuals is a simple example of the best case: the exploration process continues in an expected direction.

Mixed-initiative concepts have been explored in some detail by planning researchers. James Allen's TRAINS work draws a useful analogy between mixed-initiative planning and dialog between problem-solving agents [Allen, 1994]. Allen identifies three distinguishing characteristics of the approach: flexible, opportunistic control of initiative; the ability to change focus of attention; mechanisms for maintaining shared, implicit knowledge. Mixed-initiative systems display these characteristics to a greater or lesser extent, depending on the domain in which they operate and the requirements on their behavior.

To see how these concerns are reflected in AIDE, let's examine the most common types of user interaction with the system. AIDE interacts with the user at focus points, which appear when a choice is required about which data should be explored, which plans should be applied, how results should be further handled, and so forth.

The system presents the user with a list of its potential actions, drawn from the agenda of the current focus point. The user may select one of these possibilities, or a menu option, or may simply indicate that AIDE should proceed further (giving no substantive guidance.) Thus AIDE's list of actions can be interpreted as advice for the user, but serves equally well as documentation of the system's intentions.

When the user selects a specific course of action, the system can respond in different ways.

- AIDE can present the result of the action, and do nothing more. This happens when none of AIDE's plans is applicable in the given situation. Because AIDE's plans concentrate on exploratory rather than confirmatory statistics, for example, there are no plans dealing with the interpretation of many statistical tests. Told to perform a *t*-test, the system will simply present the result.
- AIDE can follow a plausible line of analysis and present a derivation of the result requested by the user. For example, when the user selects an action intended to remove curvature from a relationship, AIDE performs the transformation, making the necessary internal decisions about which variables to transform and which functions to apply, fits a line to the transformed relationship, and presents the result. In a case like this, AIDE has a good notion of where a course of action will lead, and can jump directly to the destination.
- AIDE can test many possible lines of analysis and present its intermediate findings to the user, to facilitate a more informed decision. For example, if the user selects a specific clustering criterion for a relationship, this can eventually lead to the identification of an interaction with another variable or relationship and opportunities for further exploration. AIDE explores the relevant possibilities, searching for strong patterns. When finished, rather than presenting its best result to the user, it presents all the possibilities, annotated by the information AIDE has gained. Here AIDE cannot jump directly to a

single destination but can supplement the user's contextual knowledge with its own evaluation of the relevant possibilities.

In terms of Allen's characterization of mixed-initiative planning, AIDE's control of initiative changes with context. That is, whether a decision is presented to the user or is made internally depends on situation-dependent factors. In the relationship transformation procedure above, AIDE makes several decisions that are not presented to the user. This also happens, for example, when AIDE determines that only one plan is able to satisfy a given goal (i.e., all others rendered inactive by its control rules); such a decision point is not presented to the user. The user is nevertheless free to take the initiative at any point in the exploration. AIDE's presentation of a list of plausible actions does not force the user to take one of them. Its support of conventional interaction with a statistical user interface allows the user to select operations from the menu bar, among other possibilities, entirely disregarding the suggestions and results AIDE generates.

The second of Allen's criteria involves focus of attention. When the user changes the focus, AIDE follows suit. For example, the user may decide to abandon exploration of relationship (x, y) for exploration of (z, w), simply by choosing "Select variable/relationship" from the menu bar and constructing the new relationship. AIDE interprets this as a shift in focus of attention, and runs the appropriate meta-level actions to match the shift. That is, the meta-planner searches through the focus point network for the focus point that manages variable and relationship exploration in the currently selected partition. It returns this focus point, along with the appropriate relationship, as the point at which exploration should continue. AIDE also makes limited decisions on its own to change focus. For example, after fitting a line to a relationship and generating residuals, AIDE presents a set of residual examination and other plans to the user. An "Okay" gesture, which usually indicates that the top-rated plan for the current decision should be activated, rather in this context causes AIDE to

refocus on other plans for exploring the relationship. The interface provides explicit navigation mechanisms to help the user follow AIDE's shifts of attention.

Allen's final point deals with shared context. Part of the shared context between the user and AIDE is implicit in the plans stored in the library. These plans are intended to implement common statistical procedures, so that the user, when encountering a focus point, will meet a plausible and perhaps even familiar decision. Navigation mechanisms further support the sharing of context. The user can view the data under consideration, results constructed, commands carried out, and planning structures leading to the current state. There is no complementary facility for AIDE to ask for clarification of user actions, however, which could clearly be helpful in some situations.

These criteria identify AIDE as a mixed-initiative planner. As a general solution, AIDE is incomplete. Its initiative is limited, for example, by the goals of exploration; once AIDE has presented a result, it must wait for the user to digest the information before it can continue. Its "dialog" with the user is highly constrained by the statistical user interface framework. Nevertheless, though AIDE does not solve all of the problems of mixed-initiative planning, it provides a working example of how the central issues can be addressed.

Systems like AIDE can be generally described as collaborative systems. Collaboration is a process in which two or more agents work together to achieve shared goals. Loren Terveen has identified a set of issues that must be addressed by any system that collaborates in an intelligent way with its users [Terveen, 1993].

- The system must be able to reason about shared goals to be achieved. It must be prepared to change its goals to meet new problems or new user requirements.
- The system must decide how to achieve its goals, which parts of the task it should take on, and how to coordinate its activities with the user. More

concisely, the system must be concerned with planning, allocation, and coordination.

- The system must be aware of the context it shares with the user. This includes implicit knowledge that can suggest and justify actions. Through this context the system can track progress toward goals, its own progress as well as that of the user.
- Collaboration requires both implicit and explicit communication between the system and the user about goals, coordination, and evaluation of progress.
- The system must adapt and learn. Because the effectiveness of human collaboration tends to improve with time, as the participants become more familiar with one another and the task at hand [Rouse *et al.*, 1987], the system should similarly adapt its behavior for the purposes of a single collaborative session as well as learn how to improve future collaborations.

This is a more general view of the issues considered in mixed-initiative planning. In AIDE we have concentrated on the issue of planning, allocation, and coordination. AIDE's design nevertheless provides partial solutions for the other areas as well. Reasoning about goals is provided by explicit plans and control rules. The system keeps close tabs on the context it shares with the user. To that end, every effective action taken by the user is interpreted as a planning or meta-planning action. This makes it straightforward for AIDE to track its own progress and the user's progress toward explicit goals. Exchange of contextual information is explicit in the system's presentation of information about dataset features and indications, possible actions at different decision points, and preferences among these actions.

AIDE has limitations with regard to these points as well. The system reasons about which parts of the task it can take on, but not not *whether* it should take the initiative. Its coordination with the user remains restricted by the statistical interface framework; it will not, for example, ask the user why one particular action

was taken rather than another. And while AIDE adapts its behavior to individual datasets, it does not learn, nor does its behavior change with different users. These are not necessary conceptual limitations, but they are a part of the current design and implementation.

7.2 Interacting with AIDE

Interaction with AIDE takes place through two frames, the main interaction frame and the navigation frame. Each frame is divided into several panes with different responsibilities. Users spend most of the time interacting with AIDE in the main interaction frame, using the navigation frame to re-orient themselves in the exploration or to redirect the process in ways more complex than can be handled by the main interaction frame.

7.2.1 The AIDE Interaction Frame

The main interaction frame is shown in Figure 7.1.

Interaction with AIDE in the main interaction frame is through

- Menu commands. The list of menu headers, which contain groups of menu commands, appears across the top of the frame.
- Interaction regions. These four regions subdivide the main body of the frame. A region contains one or more panes.
- Button selections. The set of active buttons runs across the bottom of the frame.

The upper left interaction region is the *source pane.* The source pane displays the data currently under consideration, what we call the current structure. The current structure may be a variable, relationship, or the entire dataset; more specialized structures represent decision trees, graphical models, tables, and other types of dataset. The source pane displays the current structure in an appropriate form. Variables can

be displayed as histograms, scatter plots, or row-line plots, relationships as scatter plots or contingency tables, datasets as graphs or data tables. Specialized structures have associated display forms as well. As the focus of the exploration changes, the current structure and its display in the source pane follow accordingly.

The upper right region is the *result pane*, which contains descriptions and the results of exploratory operations applied to the current structure. Descriptions are features of the current structure, such as summary statistics (e.g., mean, median, standard deviation), and indications, or heuristic judgments about properties of the current structure (e.g., evidence of clustering, linearity, nonlinearity.) AIDE uses these features and indications internally as part of its evaluation and manipulation of the data. They are presented to the user as a way of maintaining context, so that the

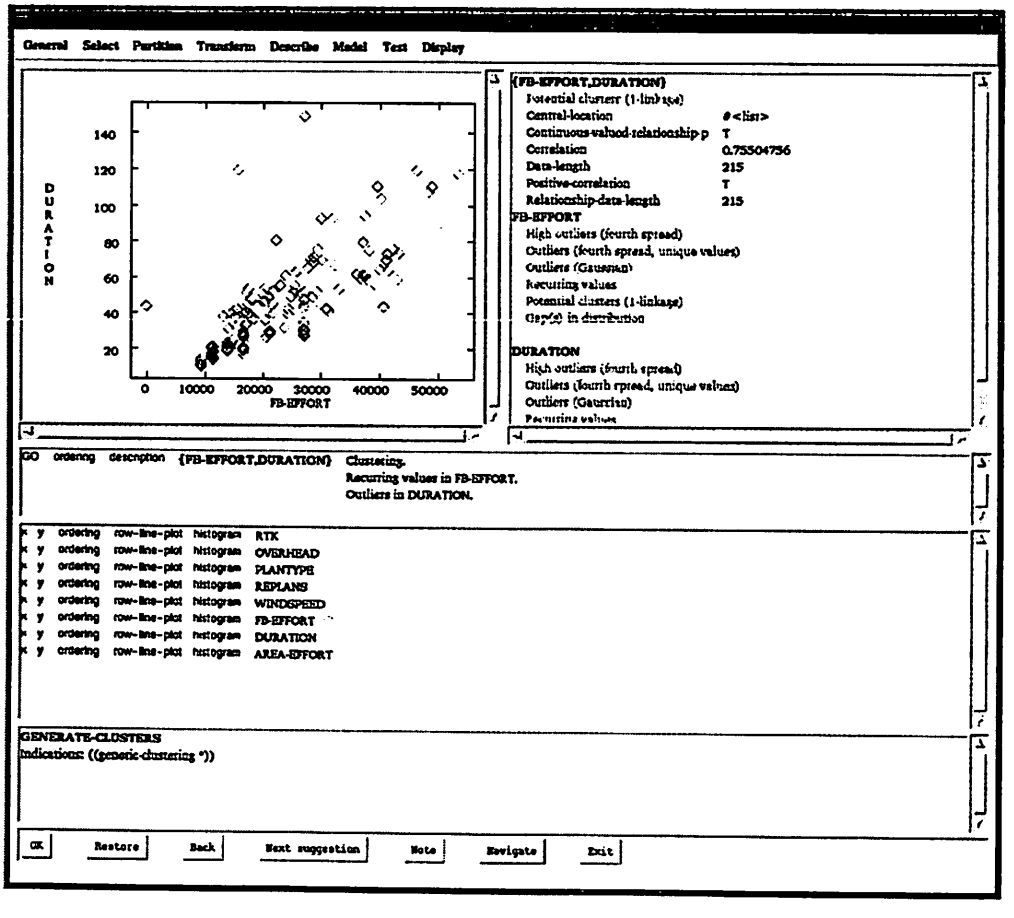


Figure 7.1. AIDE's main interaction frame

user can see what AIDE thinks are relevant patterns in the data, the triggers that cause AIDE's actions. Indications are active, meaning that when the user selects an indication with the mouse, the display changes to document the indication's value. On selecting a clustering indication for a relationship, for example, the user sees a scatter plot with points in different clusters displayed in different colors. Indications of outlying values, of "gaps" in a distribution, of recurring values in a distribution, and of skewness are displayed similarly. Indications of curvature in a relationship are displayed as a scatter plot with a superimposed piecewise linear curve. Other indications have specialized displays as well.

The result pane also displays the results of EDA operations. For example, if a relationship is displayed in the source pane, and a log transform of one of the variables is called for, the resulting transformed relationship is displayed in the result pane. A simple linear regression causes a scatter plot with a superimposed line to be displayed in the result pane. The results of statistical tests are displayed in tabular form in the result pane. When the result is a data structure that can be explored further, it is called the current result.

The lower right region is the documentation pane. It duplicates the contents of the result pane, and contains additional, supplementary information about a given result. For example, if the current structure is a relationship, and the current result a linear regression, the source pane displays the relationship while the result pane displays a scatter plot with a superimposed line. The documentation pane contains a copy of the result pane plot, plus additional relevant information about the fit, including its slope and intercept, coefficient of determination (R^2), and significance (p-value). While the source and result panes contain snapshots of the current exploratory status, the documentation pane maintains a scrolling history of all the actions that have been taken since the beginning of an exploratory session.

The lower left region is concerned with communicating AIDE's intentions to the user. It contains three panes: the *selection pane*, the *suggestion pane*, and the *locator*

pane. The selection pane contains a list of possible actions the user can take. These are the actions AIDE deems relevant in the given context. Each action has an associated set of active items, which give the user documentation about why an action is relevant and why the actions are presented in the given ordering. One active item associated with each action is the "GO" item, which directs AIDE to execute that action. The suggestion pane contains the action AIDE judges to be the best action in the current situation. The locator pane is intended to help the user keep track of any decisions AIDE makes internally, so that they may be reviewed later if desired. The locator mechanism is part of navigation, which is described in more detail in Section 7.2.2.

The interaction regions provide the user with information about the state of the exploration, along with a few well-delimited ways of directing AIDE to perform specific actions. The user has more flexibility in communicating with AIDE through menu commands. Through these commands the user can load a dataset, compose variables into relationships, compute summary statistics, generate linear models, partition data, run statistical tests, and so forth, all the capabilities of a conventional menu-based statistics interface. The categories of operations are described here only briefly; a full description is given in Appendix C.

General: General commands initiate and end an exploratory session. They also are used to load datasets and save results.

Select: Selection commands let the user choose a data structure to explore. This may involve choosing a variable or relationship from a dataset or choosing a subset of observations (a partition) of a dataset.

Describe: Description commands display selected statistics for the current structure.

Partition: Partition commands decompose the current variable or relationship into subsets, so that local patterns can be explored in more depth.

Transform: Transformation commands transform a variable or relationship by mapping a function over its observations. A log transform, for example, maps each observation a of a variable to $\log a$.

Model: Modeling commands generate structured descriptions of the data. These include linear functional models, using least squares and resistant fits, as well as linear causal models based on various conditional independence heuristics.

Tests: Test commands display the results of statistical test, such as t-tests and ANOVAs. These results are presented to the user, but not explored further.

Display: Display commands generate independent local displays of data and results. Display commands affect only the way a dataset is displayed.

Notice that these categories correspond closely to our understanding of basic EDA operations: partition operations decompose data; description operations reduce data; transformation operations, of course, transform data. Selection operations focus the exploration on specific parts of the dataset. The other categories extend exploration with modeling operations and hypothesis testing. These operations and their categorization are similar to those provided by most conventional statistical interfaces.

Button selections are one more way the user communicates with AIDE. Buttons represent strategic directives the user gives to the system. In speaking of selecting some button "X", we will sometimes refer to the "X gesture."

Okay: This button gives AIDE an acknowledgment, which is interpreted in a context-dependent way. In the context of selecting a variable or relationship, "Okay" means that AIDE should begin exploration. In the context of choosing from a set of plans, "Okay" means that AIDE should execute its top-rated choice. In other situations "Okay" may take on other appropriate meanings. Through the "Okay" gesture the user tells AIDE "Do what you think should be done next."

Suggestions: Some decision points contain so many choices that it is impractical for the user to make a reasonable selection without assistance. For example, given a dataset of n variables, which variables or relationships should one explore first? For such decision points AIDE maintains a list of suggested actions. Clicking on the Suggestions button moves through AIDE's suggestions. A related button moves the user backward through the list of suggestions already seen.

Note: This button lets the user enter a comment on the current structure or result.

Back: An exploratory session begins with the user loading a dataset and making a sequence of decisions about how to explore it: whether to explore relationship (x, y) or (z, w), whether to view a relationship as clusters or a straight line, how to partition a variable. A decision, once made, must be reversible—the discovery of unexpected patterns is central to EDA. The sequence of decisions thus becomes a tree of decisions. The "Back" button lets the user return to the parent of a decision point in the tree. For example, if the user has selected relationship (x, y) for exploration, then selecting the "Back" button brings the user back to this decision, so that another relationship can be selected.

In some situations it can happen that the user will make a selection at a decision point, immediately select "Back", and be shown a previously unseen decision. This happens because AIDE handles some decisions internally, without consulting the user. For example, if the user accepts AIDE's suggestion to fit a line to a relationship, AIDE will select the type of line without consultation. By selecting "Back", the user can revisit this internal decision to review its justification and override it if necessary. If the user is willing to accept the linear fit without further examination of the process, then the exploration simply proceeds. In the latter situation, the exploration process has been facilitated—the user need not even be aware of the intermediate steps between choosing a general course of action and the final result.

Recent The "Recent" button shows the user all the decisions along the path between the current decision point and the root of the decision tree. The user can select a decision to revisit or simply review the chain leading to the current point. By giving explicit contextual information as well as the ability to move backward through the decision tree, the "Recent" button acts as a more powerful version of the "Back" button.

Navigate The "Navigation" button brings up the navigation frame. The navigation frame can display the entire decision tree so that the user can move to arbitrary decision points, to review and modify them. It is thus more powerful than the "Recent" or "Back" buttons, but serves the same general purpose. Further navigation capabilities are described in the next section.

7.2.2 The AIDE Navigation Frame

The navigation frame is shown in Figure 7.2. The navigation frame provides some of the same functionality of the main interaction frame of Figure 7.1 but serves a different purpose. The navigation frame helps to orient the user in the landscape of the exploration process. Different display modes give the user a variety of ways to view the status of the exploratory session.

Command history The command history mode displays a list of the commands that the user has selected from the menus. Selecting a command shows the operation, the data to which it was applied, and the result generated. This facility gives a surface-level history of the exploration process. It is shallow, in that it reflects only the actions taken and not the decisions that led to their selection.

Overview The overview mode presents a graph of the dataset, with variables represented as nodes and relationships as arcs, marked appropriately to show which have been explored and the types of descriptions that have been established for

them. The user can review a variable or relationship along with its indications from this mode, and can direct AIDE to return to the initial point of exploring that variable or relationship.

Data The data mode displays the dataset in textual tree form. In addition to the variables and relationships of the dataset, this mode also presents derived data (e.g., transformations, partitions) and results.

Plan The plan mode displays the plan structure as it has been elaborated to the current point. Each focus point is presented, with documentation about the decision it manages. By browsing through this graph, the user can review past decisions. The user can also cause AIDE to return to an earlier decision to resume at that point. The plan mode is a more powerful version of the "Recent" and "Back" buttons, allowing the user to move to any point in the decision tree.

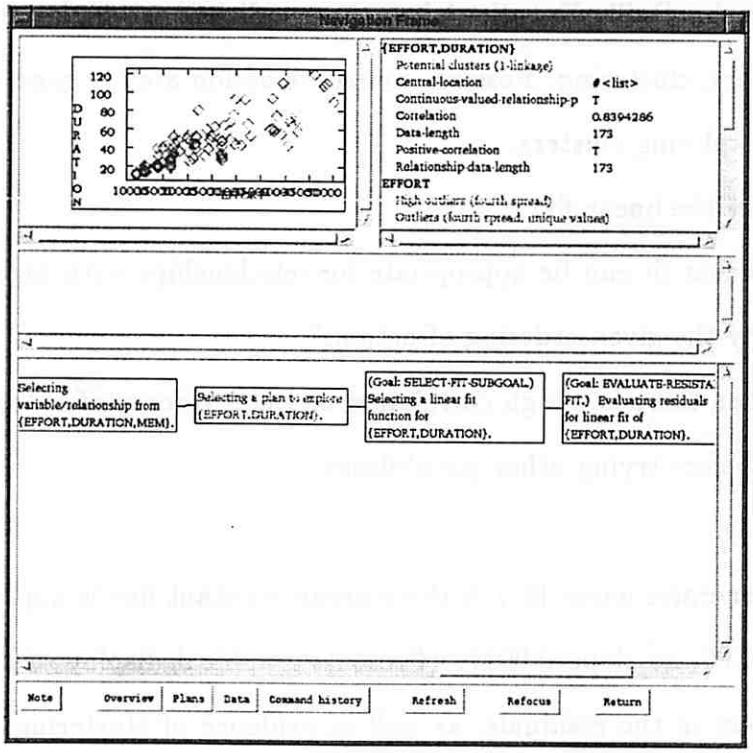


Figure 7.2. AIDE's navigation frame

7.3 A Dialog

The foregoing section describes the components of the AIDE interface. We can best show how it behaves by walking through an example, a modification of the PHOENIX example in Chapter 2. We'll first present the dialog as it happens in AIDE and then go through each interchange with a detailed explanation. Because it can be tedious to follow a description of pop-up windows, menu item selections, and graphics, the dialog is presented in a more natural form. The information exchanged between AIDE and the user is nevertheless accurately described.

User: Load dataset "Phoenix successful trials".

Aide: Dataset features: Number of observations, correlation table... Evidence of clustering in these variables... High correlation/rank correlation in specific relationships.

User: Select relationship (FirelineBuilt, Duration).

Aide: (FirelineBuilt, Duration) has these indications: high correlation ($r = 0.93$); outliers; clustering. Possible courses of action are (1) generating a linear fit and (2) exploring clusters.

User: Why the linear fit?

Aide: A linear fit can be appropriate for relationships with high correlation.

User: Why the given ordering of actions?

Aide: When there is a high correlation and no evidence of extreme outliers, we fit a line before trying other possibilities.

User: Okay.

Aide: (Generates linear fit.) A three-group resistant line is appropriate; its intercept is -0.96, its slope 0.0017. (Presents graphical display of linear fit.) There are outliers in the residuals, as well as evidence of clustering. Possible courses of action are (1) examining residuals alone, as a variable, (2) exploring (FirelineB-

uilt, ResidualDuration), and (3) exploring (PredictedDuration, ResidualDuration).

User: Show recent decisions.

Aide: (1) With the goal of exploring (FirelineBuilt, Duration), we decided to fit a line. The active alternative is to explore clusters. (2) With the goal of fitting a line, I decided without consultation on a resistant fit. Alternative is a regression fit.

User: Select previous decision.

Aide: In fitting a line to the relationship, the possibilities are (1) a resistant fit, which is in progress, or (2) a regression fit, which has not started.

User: Execute the regression fit.

Aide: (Generates regression fit.) The regression line has an intercept of -2.73 and a slope of 0.0019; $r^2 = 0.57$. (Presents graphical display of linear fit.) There are outliers in the residuals, as well as evidence of clustering. The slope of the regression line is 1.13 times the slope of the resistant line.

User: Make a note: I am confident that the relationship (FirelineBuilt, Duration) is linear, though there are outliers. I prefer the regression fit.

Aide: (Accepts note.)

User: Okay.

Aide: (Considers linear fit phase to be complete.) The plans remaining to explore (FirelineBuilt, Duration) are (1) the linear fit, which is in progress, and (2) cluster exploration, which has not started.

User: Okay.

Aide: (Begins cluster exploration.) There are several different ways (FirelineBuilt, Duration) can be broken into clusters. (Displays identifiers for potential clustering criteria: single-linkage clusters in FirelineBuilt, regions of constant values in FirelineBuilt, etc. Also displays similarity between resulting clusters.)

User: Show single-linkage clusters in FirelineBuilt.

Aide: (Displays clusters.)

User: Show constant FirelineBuilt clusters.

Aide: (Displays clusters.)

User: Select constant FirelineBuilt clusters.

This dialog presents an abstract view of the interaction between AIDE and the user. Here is a more detailed description of how the process appears to the user and what AIDE does in the background.

User: Load dataset "Phoenix successful trials".

Aide: Dataset features: Number of observations, correlation table... Evidence of clustering in these variables... High correlation/rank correlation in specific relationships.

The user loads a dataset through a menu selection, which immediately activates the planner with a top level goal of exploring the dataset. A generic modeling plan is activated to explore the dataset. Its main purpose is to establish a plan focus point that allows different types of modeling procedures to be applied. These procedures include causal modeling algorithms, multiple regression algorithms, cluster modeling algorithms, and a weaker, general-purpose exploration plan. This last plan, which is the default, simply establishes a variable focus point FP_m that maintains a variable `exploration-structure` with a list of potential bindings—all individual variables and bivariate relationships to be explored. The state of the focus point network is shown in Figure 7.3.

AIDE displays the dataset in the source pane, as a table of values or as a graph, the default. Features and indications in the dataset are presented in the result pane. These include variables with outliers, variables with evidence of clustering, highly correlated relationships, and other information. The selection pane contains the

names of the dataset variables, so that the user can select a variable or relationship for exploration. The suggestion pane contains AIDE's suggestion about which relationship to explore.

In this case, AIDE's activation and preference control rules suggest that the user explore the relationship (RTK, IdleTime).[1] RTK, which is short for "Real Time Knob", controls the fireboss's "thinking speed", or how fast the fireboss's computations proceed, relative to events in the simulated environment. By changing the value of RTK, the fireboss reacts more quickly or more slowly to the changing environment. IdleTime measures how much time the fireboss spends idle, waiting for its plans to be executed, for new information from its agents, and so forth. These two variables are very closely related, which AIDE can detect without contextual knowledge.

User: Select relationship (FirelineBuilt, Duration).

[1]For clarity, I have used the name "IdleTime" rather than "OVHD", or "Overhead", the original name of the variable.

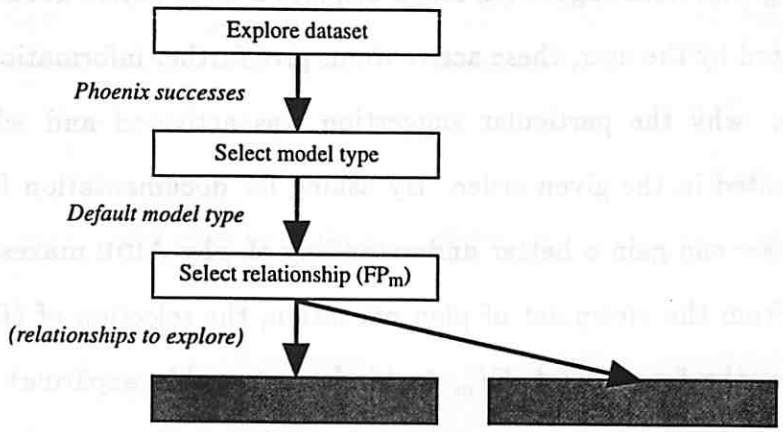


Figure 7.3. Focus point network state at variable selection time

Aide: (FirelineBuilt, Duration) has these indications: high correlation ($r = 0.93$); outliers; clustering. Possible courses of action are (1) generating a linear fit and (2) exploring clusters.

User: Why the linear fit?

Aide: A linear fit can be appropriate for relationships with high correlation.

User: Why the given ordering of actions?

Aide: When there is a high correlation and no evidence of extreme outliers, we fit a line before trying other possibilities.

Rather than following AIDE's suggestion, the user instead selects the relationship (FirelineBuilt, Duration). The interface shows the user the relationship in the source pane, as a scatter plot. Its associated features and indications, are displayed in the result pane. At this point AIDE waits for an acknowledgment that the selected structure is the desired one. Once the user indicates that the selection is complete, AIDE makes a set of suggestions (based on its activation and preference rules) about appropriate exploration operations, one of which is singled out as being the best choice according to AIDE's evaluation. In this case, a linear fit plan is judged best. Along with each suggestion AIDE displays a set of active documentation items. When selected by the user, these active items give further information in the documentation pane: why the particular suggestion was activated and why the suggestions are presented in the given order. By asking for documentation for specific possibilities, the user can gain a better understanding of why AIDE makes its suggestions.

From the viewpoint of plan execution, the selection of (FirelineBuilt, Duration) causes the focus point FP_m to bind the variable `exploration-structure` to this relationship. This focus point will be revisited whenever the user selects a different relationship to explore, each time selecting or constructing a different binding for `exploration-structure` and generating a new branch in the search space.

The active exploration plan then establishes a goal to explore the selected relationship. Activation rules, acting on indications of clustering and high correlation in the data, trigger two plans to satisfy the goal. A new plan focus point FP_p is generated to manage the two choices, as shown in Figure 7.4.

User: Okay.

Note that the user is not constrained to follow AIDE's advice. All legal menu items are active, for partitioning by other variables, for transforming the relationship, for generating statistical tests, and so forth. By simply giving the "Okay" gesture, the user indicates that AIDE's suggestion should be taken. AIDE proceeds to fit a line to the data.

Aide: A three-group resistant line is appropriate; its intercept is -0.96, its slope 0.0017. There are outliers in the residuals, as well as evidence of clustering. Possible courses of action are (1) examining residuals alone, as a variable, (2) exploring (FirelineBuilt, ResidualDuration), and (3) exploring (PredictedDuration, ResidualDuration).

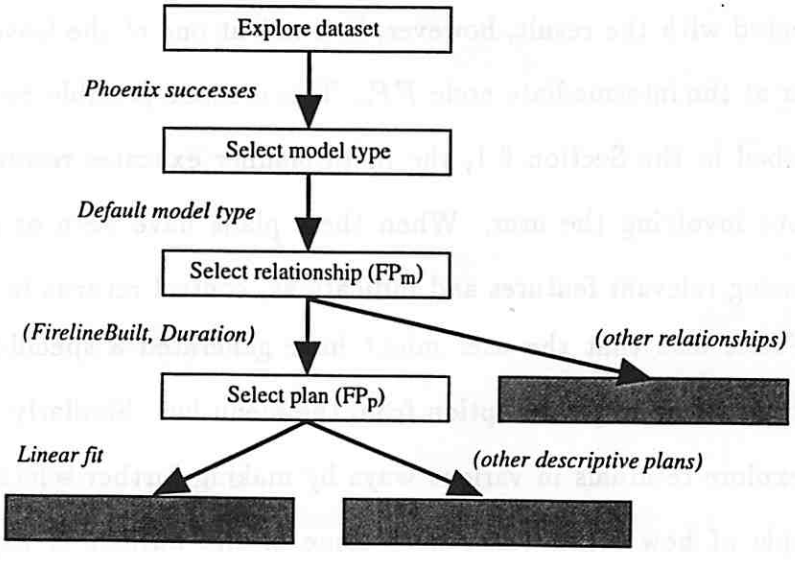


Figure 7.4. Focus point network state at plan selection time

AIDE generates a resistant line and displays it in scatter plot form, superimposed over the data points, in the result pane. The documentation pane contains the numerical parameters of the fitted line, plus additional indications present in the residuals. The selection pane contains the relevant residual exploration actions, displayed in bold typeface to indicate that they have been partially executed. Each is further annotated with partial results, such as residual outliers or indications of clustering.

Many different types of procedures can generate a linear fit for a bivariate relationship. AIDE plans include a simple ordinary regression, a weighted regression, and a resistant fit. A new focus point FP_f is generated to manage these possibilities. AIDE makes an internal decision on one of the plans, based on indications in the data, without consulting the user. Once the fit has been generated, the residuals are extracted. AIDE pursues the exploration of the residuals to a deeper level, then returns to this decision point to present the fit and any further information it has derived from its residual examination. This results in the focus point network shown in Figure 7.5.

The focus point network shows the branching at the fit focus point FP_f and its child focus point, FP_r, which handles the residual analysis. When the user is presented with the result, however, it is not at one of the leaves of the network, but rather at the intermediate node FP_r. This is made possible by the meta-planner. As described in the Section 6.1, the meta-planner executes residual examination plans without involving the user. When these plans have been executed to the point of producing relevant features and indications, control returns to the focus point FP_r.

Notice also that the user might have generated a specific type of linear fit by selecting the appropriate option from the menu bar. Similarly, the user can generate and explore residuals in various ways by making further selections. This is a simple example of how AIDE takes over some of the burden of exploration, by making these decisions internally, following plans that become applicable, and evaluating the

results. Such results are often presented as a matter of course in commercial statistics packages. In contrast, AIDE makes explicit decisions to generate these related results, in an attempt to focus on only what is relevant.

User: Show recent decisions.

Aide: (1) With the goal of exploring (Fireline-Built, Duration), we decided to fit a line. The active alternative is to explore clusters. (2) With the goal of fitting a line, I decided without consultation on a resistant fit. Alternative is a regression fit.

User: Select previous decision.

Aide: In fitting a line to the relationship, the possibilities are (1) a resistant fit, which is in progress, or (2) a regression fit, which has not started.

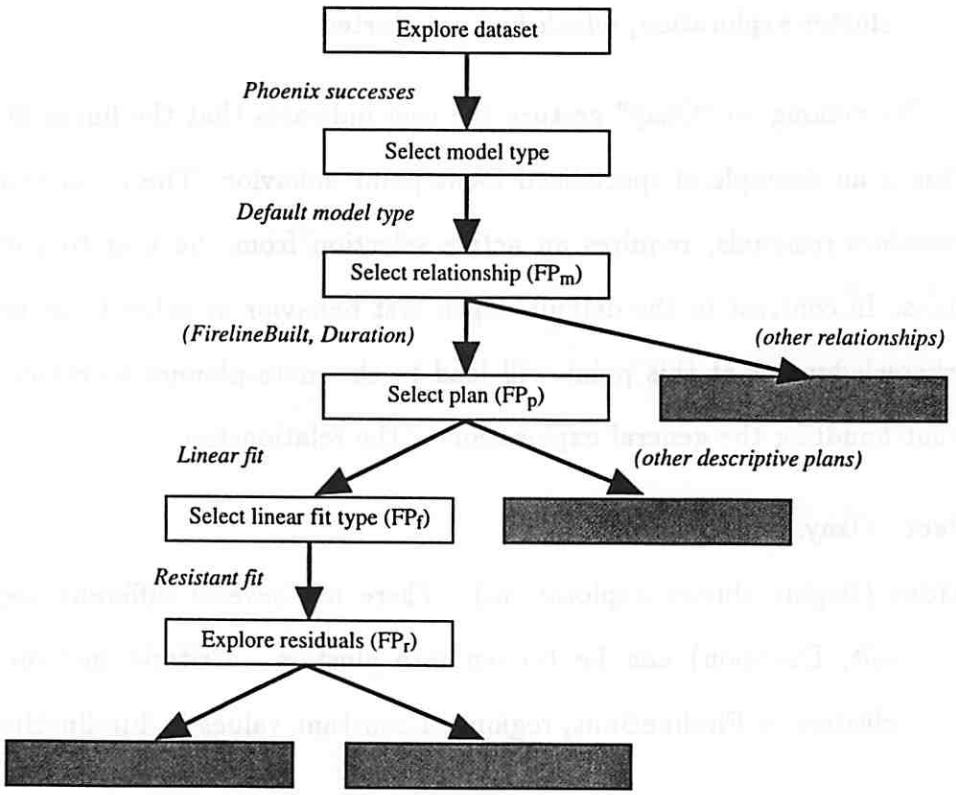


Figure 7.5. Focus point network state at residual exploration time

Here the user diverges from AIDE's suggested course of action. By making a "Recent" gesture, the user can choose from a list of previous decisions. In this case the user is reviewing the decision that led to the selection of the resistant line.

User: Execute the regression fit.

Aide: (Generates regression fit.) The regression line has an intercept of -2.73 and a slope of 0.0019; $r^2 = 0.57$. There are outliers in the residuals, as well as evidence of clustering. The slope of the regression line is 1.13 times the slope of the resistant line.

User: Make a note: I am confident that the relationship (FirelineBuilt, Duration) is linear, though there are outliers. I prefer the regression fit.

User: Okay.

Aide: (Considers linear fit phase to be complete.) The plans remaining to explore (FirelineBuilt, Duration) are (1) the linear fit, which is in progress, and (2) cluster exploration, which has not started.

By making an "Okay" gesture the user indicates that the linear fit is acceptable. This is an example of specialized focus point behavior. The focus point FP_r, which examines residuals, requires an active selection from the user to pursue one of its plans. In contrast to the default depth-first behavior at other focus points, a simple acknowledgment at this point will lead to the meta-planner to refocus on the focus point handling the general exploration of the relationship.

User: Okay.

Aide: (Begins cluster exploration.) There are several different ways (FirelineBuilt, Duration) can be broken into clusters. Criteria include single-linkage clusters in FirelineBuilt, regions of constant values in FirelineBuilt, etc.

Criteria are grouped, based on the similarities by which they assign data points to different clusters. For example, one way of clustering points is to break them

into "outliers" and "non-outliers". Many outlier detection criteria may be applicable, and some of the results will be similar (e.g., sometimes outliers determined by a Gaussian test will be indistinguishable from outliers determined by a resistant fourth-spread test.) Groups of clustering criteria are presented for examination by the user. When a criterion is selected, either by the user's explicit choice or gesture of "Okay", exploration continues with the partitioned dataset.

7.4 Related Work

AIDE stands alone in its focus on mixed-initiative, general-purpose data exploration. A number of systems attack problems of comparable difficulty, however, emphasizing human interaction issues. A brief discussion of significant work in the areas of collaborative systems, intelligent interfaces, intelligent agents, and statistical interfaces is given below.

Silverman proposes a model of human-computer collaboration in which six factors are important: cognitive orientation, or the style of human-computer dialog; deep knowledge, or the extent to which each participant in the collaboration understands the other's reasoning; intention-sharing, or the process by which each participant makes intelligent guesses about the other's problem-solving representation; control plasticity, or sharing of initiative; adaptivity, or learning about the problem; and experience, or learning about the collaboration [Silverman, 1992]. These factors are presented as a starting point for further research, rather than as elements of a testable model. AIDE's design focuses on implicit intention-sharing and control plasticity.

In a discussion of the architecture of intelligent interfaces, Rouse et al. propose an alternative to the common, implicit goal of interface designers to automate as much as is technically feasible [Rouse *et al.*, 1987]. Rather, a design should be centered around the user and the problem to be solved, enhancing human abilities, overcoming human limitations, and complementing individual human preferences. "Furthermore," the authors state, "these capabilities [should be] provided in a manner that preserves

operators' discretion to act in innovative ways whenever appropriate." [Rouse *et al.*, 1987, p. 91] It is easy to see how these concerns are reflected in AIDE's design.

In a description of intelligent agents, Etzioni outlines a number of general characteristics [Etzioni and Weld, 1993]. Agents run continuously, rather than being "one-shot" programs. They have distinctive characters, they are communicative, they adapt to their human users, and they can move about. Most importantly, agents are autonomous, able to take initiative and exercise control over their own actions. Because AI systems are traditionally autonomous, AI researchers often describe the properties of intelligent agents in terms of modifications and extensions to this autonomy. In AIDE we have taken a more balanced view of the interaction between the user and the system.

Finally, statistical researchers have also addressed collaboration issues. An early view, due to Huber, is to some extent a reaction against batch systems and limited-functionality interactive systems: Huber proposes that the best statistical environment is a complete programming language with appropriate statistical extensions, in a Lisp-like, interactive development environment [Huber, 1986b]. Lubinsky and Pregibon's TESS takes a different approach, by acting autonomously when it can, but accommodating user preferences when necessary [Lubinsky and Pregibon, 1988].

AIDE is principally concerned with how a system can balance autonomy and accommodation. All of these accounts, either peripherally or directly, attend to this issue as well. In Silverman's model the control plasticity factor subsumes this issue; in Rouse et al.'s intelligent interfaces, automation stops at the point when it begins to threaten accommodation; in Etzioni's agents, accommodation is a constraint on autonomy, as well as an aspect of communication and adaptation; in TRAINS and TESS, the issue is addressed directly.

7.5 Summary

AIDE's design provides an adequate framework for an autonomous system. At any point in the exploration AIDE considers one decision—a specific node in its network of decisions—to be its current focus of attention. The system can make autonomous decisions based on its internal plans and control rules, without consulting the user. New findings are combined in the network with existing results, creating opportunities for further exploration.

However, complete autonomy is undesirable for AIDE; rather, the system must accommodate the user's knowledge about the data and the goals of the analysis. Maintaining the balance between autonomy and accommodation is harder than might appear at first glance. If the system can take actions without consulting the user, it can as a natural consequence drive the analysis into inappropriate areas, possibly losing the user in the process. Similarly, the user must not be constrained to take only those actions the system finds reasonable; on the contrary, with better visual pattern-detection abilities and knowledge of what variable values actually mean, the user should be able to take the analysis in whichever direction appears appropriate. The system must nevertheless be able to follow in the user's path, ready to contribute to the process once it again reaches a familiar area.

AIDE's design addresses these concerns as well. Plans attempt to implement common statistical practice, relying implicitly on the user's knowledge of what actions are appropriate in a given situation. Thus AIDE's autonomous actions often anticipate the user's goals. More significantly, the user can explicitly direct AIDE's activities, the interaction taking place within a conventional menu-based statistics package. Menu choices let the user load a dataset, compose variables into relationships, compute summary statistics, generate linear models, partition data, run statistical tests, and so forth. These menu operations are tied internally to the network of decisions, so that if the user tells AIDE to run a regression of y on x, the system will search through

the network to find a decision point associated with selecting relationships, then find or create an appropriate branch for (x, y), and then incrementally select a sequence of plans that run the regression, explore the residuals, evaluate the results, and so forth. The user can review both the results and an explicit model of the decision-making process that led to the results, and make modifications as necessary.

Interaction is thus a flexible trade-off between autonomy and accommodation. When the user gives AIDE explicit directions to perform some operation, the system can follow along, ready to give assistance once that result is reached. When the user gives AIDE free rein, by saying, "Okay" to one of its suggestions, the system proceeds autonomously, until it reaches a result it considers significant. At this point the user can review, extend, or redirect the analysis. The system behaves neither like a conventional statistics package, nor like an autonomous machine learning program, but somewhere in between.

CHAPTER 8

EVALUATION

Our evaluation boils down to two central questions. First, can users do a better job of exploring data with the AIDE's help than without it? Second, if AIDE does indeed help users explore data, what is the nature of its contribution? Our experiment design involves two conditions. In one condition, users explore a dataset with the help of AIDE's observations, suggestions, procedures, and evaluation of results; in the other, users explore a dataset given the same underlying functions but no additional assistance.

We can briefly summarize the results as follows. AIDE helps users to make decisions about the presence of structure in the data, better decisions than are made without AIDE's help. AIDE improved the performance of almost all the subjects in the experiment; its degree of influence depended somewhat on the subject's style of exploration. AIDE furthermore did not significantly reduce the confidence of subjects in the observations they made, a concern for any system that takes responsibility for significant portions of a complex task. Finally, we have suggestive evidence that human contextual knowledge helps to focus exploration on relevant parts of the search space, though the interaction between this knowledge and AIDE's assistance remains unclear.

Analyzing the exploratory sessions of individual subjects, we found that the subjects varied widely in the ways they explored the test datasets. Given a dataset and flexible statistical tools, different users can take very different paths in their analysis of the data. These differences can involve styles of analysis, preferences among statistical operations and procedures, familiarity with the domain of the data, and confidence

in results. We observe in general, however, that with AIDE, users perform a more extensive analysis of the data, taking good advantage of the system's navigation facilities to help move through the space of data and exploratory operations.

8.1 Experiment Design

It will be easiest to understand the experiment design if we first discuss what we can measure in AIDE and then walk through the steps in the evolution of the design. Our discussion will end with a set of hypotheses about AIDE's performance.

8.1.1 Measurements

As described in Section 7, the user may make the "Note" gesture at any point during the exploration process. The user's *observations*, based on templates that describe specific patterns, provide a high-level record of exploration activity. The set of default pattern types is as follows:

- *No relationship:* There is no direct relationship between the specified variables.
- *Unspecified relationship:* A direct relationship exists between the specified variables, but the exact nature of the pattern cannot be established.
- *Linear relationship:* A direct linear relationship exists between the specified variables.
- *Nonlinear relationship:* A direct polynomial or power relationship exists between the specified variables.
- *Clustered relationship:* The relationship (or variable) can be broken down into partitions or localized clusters. This type of pattern can point to indirect relationships; for example, a relationship (x, y) might be linear in one cluster and superlinear in another, the difference determined by a third variable z.

For each observation the user enters a confidence (high or low) in the pattern, and also specifies any features of the data that might throw doubt on whether the

given pattern is genuine. For example, a relationship might consist of two elongated clusters, making it unclear whether it should be described as a line, a higher-order function, or a set of clusters. The user can describe such a relationship as a line with indications of clustering, a polynomial with clustering, or as a set of clusters with an unspecified functional relationship between the variables. The user may also extend the observation by referring to other variables; for example, the clusters in a relationship may be due to the influence of another variable or set of variables. Text comments are also entered to give specific parameters of the descriptions, and to qualify patterns or relationships for which the default types are not an exact match.

These observations summarize the results of an exploratory session, the explicit conclusions of the user. We also record the activity leading to these conclusions, including every discrete action the subject makes, from selecting a menu choice to clicking on an indication for documentation. For each action, we record the following:

- *Type:* The general type of action, such as "Select plan" or "Back".
- *Time stamp:* A number giving the relative ordering of actions, plus the interval between successive actions.
- *Source:* The data under consideration at the time of the action.
- *Result:* The result, if any, produced by the action from the source data.
- *Context:* The planning context in which the action was executed.

Finally, all subjects were asked to write a post mortem assessment of their experience using AIDE.

8.1.2 Design Issues

Our experiment compares two conditions. In the USER+AIDE condition, subjects explore a dataset with AIDE's help. In the USERALONE condition, subjects explore a dataset in a similar statistical computing environment, but without active

interaction with AIDE. We determine AIDE's effectiveness by measuring differences in performance between the two conditions. This preliminary description immediately raises several issues.

Extraneous variability in conditions: In order to prevent usability issues from entering the picture, we must ensure that the system used in the USER+AIDE condition is as close as possible to the system in the USERALONE condition. That is, we cannot simply use AIDE in the USER+AIDE condition and, say, JMP [Sall and Lehman, 1996] or CLASP [Anderson *et al.*, 1993] in the USERALONE condition, because differences in performance would then not necessarily be attributable to AIDE, but perhaps to differences in user interface design. To avoid this problem, the USERALONE condition reproduces AIDE's environment, lacking only the intelligent interaction capabilities. For example, in the USER+AIDE condition, the system might suggest that the relationship (x, y) be explored, then propose and execute a linear fit procedure on the data, exploring residuals afterwards. In the USERALONE condition, the user receives no suggestions about which relationships might be worth exploring; once the user selects the relationship (x, y), the system gives no advice about how to describe the data; the user must explicitly select a regression or resistant fit for the data, and then explore the residuals with further explicit commands. In the USER+AIDE condition, AIDE takes over some of the control, with the accompanying possibility of leading the exploration astray, while in the USERALONE condition the user can (and must) select every action.

Variability in subjects: Suppose we divided subjects into two groups, Group A to explore a dataset in the USER+AIDE condition, Group B to explore the same dataset in the USERALONE condition. Suppose further that Group A subjects outperformed Group B subjects, on average, for some performance measure. Unfortunately, because different subjects have different facility with EDA techniques, the performance of Group A might be attributable to random variation in user ability.

A paired comparison design can remove this variability. Each subject is tested in both conditions. Our performance measures are then not biased by differences in ability between individual subjects, because these abilities will be represented in both conditions. We can compare performance differences for individuals, as well as aggregate measures of performance per condition.

Practice effects and variability in problem difficulty: If subjects are to be tested in both conditions, we immediately face a serious practice effect. A subject who has explored a dataset in one condition will certainly be able to take advantage of this knowledge in the other condition. Randomizing the presentation of conditions is no help in this case. Giving subjects two different datasets may also pose problems: how can we compare performance between them? The datasets might be very different in their structural characteristics and the patterns they contain.

Artificial data is the answer. We can generate datasets based on identical or nearly identical models that give us datasets with very similar characteristics. In doing this we need to be careful that the information gained from exploring one dataset does not help in exploring another. This turns out not to be a problem; knowing that a log relationship holds between x and y in dataset D_1, for example, does not help us to identify and describe a similar relationship in dataset D_2.

Ordering effects: In pilot runs of the experiment we found that each phase of each trial (i.e. testing in each condition) took from one to two hours. This potentially gives rise to an ordering effect: the subject may become more comfortable with the system during the first phase, and have less difficulty during the second.

A counterbalanced design controls for this effect. The order in which the subject is presented with the two conditions is randomized. Thus even if subjects *always* do better in the second phase, the improvements will apply to both the USER+AIDE and the USERALONE conditions. We have no reason to believe that an improvement will favor either condition, and so the effects should cancel each other out.

Variability in effort: We have designed AIDE to help the user explore a larger search space than might be explored alone. One might then expect a user, given sufficient time, to be able to find all the structure AIDE would find. Under practical experimental conditions, however, we can't give the subject unlimited time to perform the exploration. On the other hand, if we give subjects too little time, we may be biasing the experiment in favor of AIDE, in the same way computers have an advantage playing blitz chess.

We compromise by relying on the subject's judgment about when the task is complete. We have no reason to believe that this judgment would be different for the different conditions. The subject receives instructions to present all results he or she considers significant in summarizing or explaining significant structure in the dataset.

To summarize, the design involves two conditions. In the USER+AIDE condition, the subject explores a dataset with AIDE's help. In the USERALONE condition, the subject explores another dataset in the same environment and through the same user interface, but without AIDE's active interaction. The latter environment reproduces as closely as possible the functionality of the conventional statistical package CLASP [Anderson *et al.*, 1993]. The USERALONE condition provides CLASP's functionality *in the* AIDE *environment*, so that the different styles of interaction in AIDE and CLASP do not affect our results.

Subjects were graduate student volunteers who had taken a statistical methods course. They were generally familiar with statistical user interfaces, in particular with the package CLASP, but could not be considered expert data analysts.

8.1.3 Hypotheses

At this point we have laid the groundwork to pose a hypothesis:

Exploration is more effective with Aide than without.

We define two measures of performance. In each condition t a subject s makes some number of observations: $\text{Obs}_{ts} = o_{ts1}, \ldots, o_{tsk}$. We can classify each observation

as correct or incorrect. By "correctness" we mean that the subject has correctly described one of the *direct* relationships in the model that generated the data. The first measure, $\overline{p}$, is the mean number of correct observations made,

$$\overline{p} = \frac{\sum_{i=1}^{k} \text{Correct?}(o_{tsi})}{k},$$

where k is the number of observations the subject makes in a condition and "Correct?" is a function that returns 1 if an observation is correct, 0 otherwise. Informally, $\overline{p}$ for a given subject gives the probability that one of his or her observations is correct.

The $\overline{p}$ performance measure takes both correct and incorrect judgments into account, which may sometimes be deceptive. We would like to distinguish, for example, between a subject with a $\overline{p}$ of 0.5 for a large number of observations and another subject with the same $\overline{p}$ score for many fewer observations. Another measure, which we call $k\overline{p}$ for consistency, considers number of correct observations alone:

$$k\overline{p} = \sum_{i=1}^{k} \text{Correct?}(o_{tsi}).$$

Having two measures can help us to make distinctions between subjects with conservative and liberal strategies in making observations. We can test our hypothesis with a pairwise comparison of subject performance (both $\overline{p}$ and $k\overline{p}$) in each condition.

The second hypothesis addresses the human element. The danger in giving a system autonomy is that its less predictable behavior will cause users to mistrust its results. In other words, if AIDE is as successful as we hope, we should see that users are no less confident in results obtained through AIDE as they are in results obtained by hand. We hypothesize, then, that

Aide's assistance does not degrade user confidence in observations.

In other words, the null hypothesis is that there is no significant difference in confidence between conditions. If we cannot reject the null hypothesis, we may tentatively conclude that it is not a factor. Note that one never accepts the null hypothesis; there may be many reasons why a difference cannot be detected.

Much of the literature in statistical expert systems holds that an important factor in successful exploration is the contextual knowledge a user brings to bear [Huber, 1986a; Lubinsky and Pregibon, 1988; Terveen, 1993]. Because we are dealing with artificial datasets, we can quantify this factor. Quantification of the contribution of contextual knowledge is a serious issue for human-computer interaction researchers [Rouse *et al.*, 1987; Silverman, 1992], but this type of analysis has received little attention in the area of statistical expert systems. In our experiment we can manipulate the level of contextual knowledge available to users, by giving some variables plausible names that correspond to natural relationships with the other variables, and giving others anonymous names like "V1" and "V2".[1] This leads us to a third hypothesis:

Exploration is more effective in the presence of contextual knowledge.

This hypothesis is actually independent of AIDE. However, we have a chance to explore the interaction of both the presence of AIDE and the availability of contextual knowledge affecting performance. This may give us some clues about when and how AIDE provides benefits in exploration. We refrain from making another explicit hypothesis about this potential relationship, however, mainly because it is not clear how to interpret the results. It could happen that AIDE compensates for a lack of contextual knowledge, giving equal or better performance for relationships involving anonymous variables. Alternatively, it could happen that AIDE works even better when the user has contextual knowledge available, so that the reverse situation holds for performance.

Our results will not be limited to these three yes/no questions. Using these as a starting point we will be able to derive a good deal of information about how users

[1]Thanks to Paul Cohen for proposing this arrangement. Each subject spends between three and four hours with AIDE; without this arrangement we might have had to double this time to explore the same issue.

go about exploration and why AIDE works. Fittingly enough, we will obtain this information by exploring the experimental data.

8.1.4 Test Data

Subjects are to explore two artificial datasets that meet special criteria. They should contain patterns and structural relationships amenable to exploration, comparable to the ones discussed in the examples of Chapter 2. These include linear and clustering relationships between variables, functional relationships between variables that depend on the values of other variables, and so on. The two datasets should furthermore be similar, in the sense that exploring one should be no harder or easier than exploring the other. Nevertheless information gained in exploring one dataset should not help in exploring the other.

Paul Cohen designed MODEL A and MODEL B and generated datasets with these criteria in mind. The names of the variables were carefully chosen to appear meaningful, but also to give no indication of direct causal relationship or causal order. The specific functions that determine each variable's value are given in Tables 8.13 and 8.14 at the end of this chapter. Each model consists of two subgraphs (g_1 and g_2) connected at two nodes. The two models are essentially the same, except for the ordering of their subgraphs. To disguise the similarities between the models, variables in subgraph g_1 of MODEL A and subgraph g_2 of MODEL B are given anonymous names. Figures 8.1 and 8.2 give the graphs that were used to generate the data, along with the variable names as they appeared to the user. From a subject's point of view, each model consists of named and anonymous variables, and there is no overlap in naming between the models.

It is important to note that in no sense was AIDE tuned to the specific patterns in these datasets. I had no role in building the data generator and producing the datasets, I had no contact with the data before the experiment with AIDE began, and this section was written largely after the data collection was completed.

8.2 Results

Each subject went through three separate phases in the experiment, with about a ten minute break between each phase. All phases were open-ended, with the subject deciding when to stop. In the first phase, the subject was introduced to the system and trained in its use, with the NETWORTH dataset described in Chapter 2 serving as an example dataset. This phase required from one to one and a half hours; it continued until subjects had become relatively comfortable with AIDE (functioning in both conditions) and felt that they had reached a plateau of familiarity with the

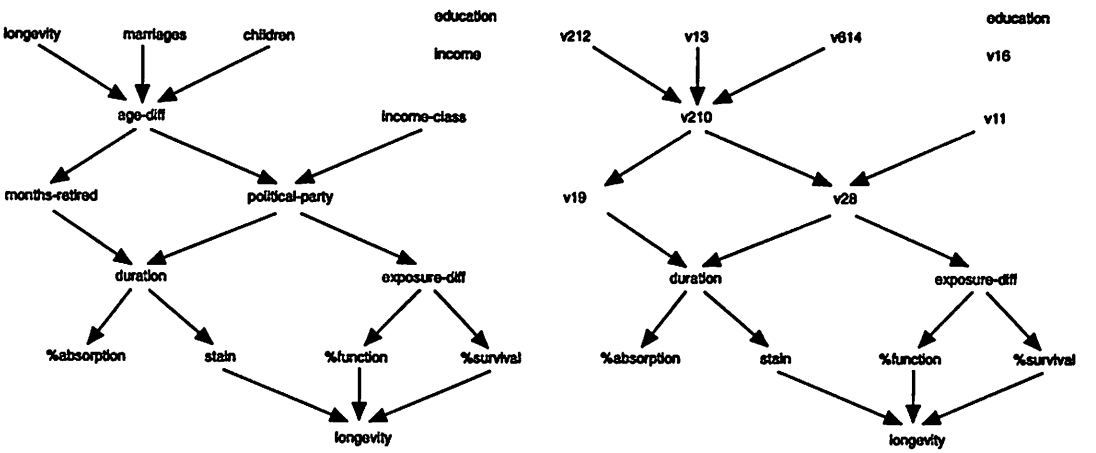


Figure 8.1. Relationships (actual and renamed) in the MODEL A dataset

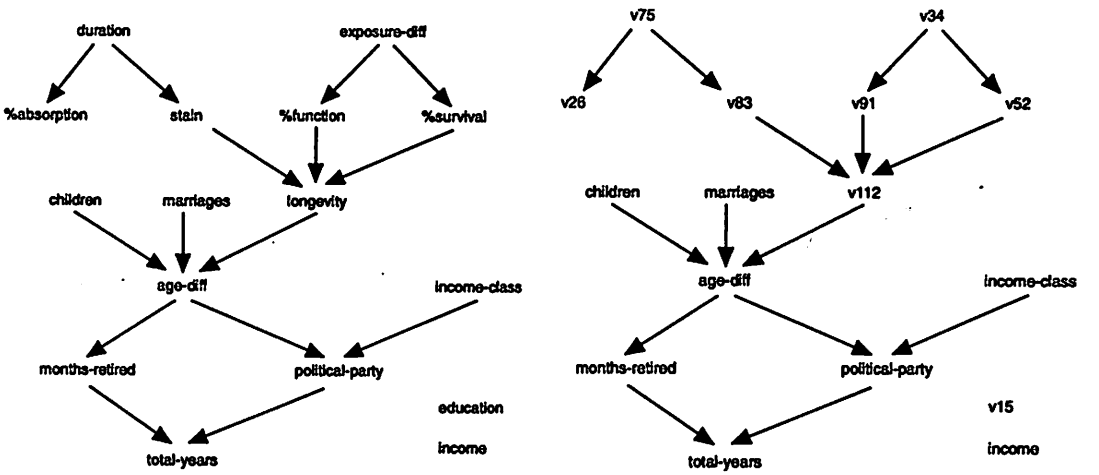


Figure 8.2. Relationships (actual and renamed) in the MODEL B dataset

system. Once familiar with the system, subjects followed one of four paths through the experiment:

- MODEL A with USER+AIDE, then MODEL B with USERALONE;
- MODEL A with USERALONE, then MODEL B with USER+AIDE;
- MODEL B with USER+AIDE, then MODEL A with USERALONE;
- MODEL B with USERALONE, then MODEL A with USER+AIDE.

Thus each subject explored both models, one in each condition. The path each subject took was determined randomly. Subjects spent about an hour in each condition, some up to about an hour and a half.

From the data collected we derived two types of result. The first concerns how well the subjects performed the exploration task. The second deals with how they went about the exploration.

8.2.1 Assessing Performance

We will consider these variables:

- *Subject:* An identifier for each participant;
- *Condition:* In which condition an observation is made;
- *Context:* Whether an observation involves an anonymous variable;
- $\overline{p}$ *and* $k\overline{p}$*:* Aggregate measures of correctness for multiple observations;
- C_M*:* Aggregate measure of confidence, defined below.

We begin by examining the difference in performance of subjects in the two conditions. $\overline{p}$, which computes the mean number of correct observations made, was recorded for each subject in each condition, giving the totals in Table 8.1. Because we are interested in whether performance in the USER+AIDE condition is better than in the USERALONE condition (rather than simply seeing a difference in either direction),

we use a one-tailed test. A matched-pair, one-tailed t-test tells us that $\overline{p}$ and $k\overline{p}$ are significantly higher for subjects in the USER+AIDE condition.

To put this comparison in perspective, we can dismiss a few plausible but trivial explanations for better performance in the USER+AIDE condition. First, subjects entered roughly the same number of observations in both conditions, with a median difference of 0.5 between the two conditions. Improved performance thus depends not only on making more correct observations, but also on making fewer incorrect observations. Further, subjects directly examined about the same number of variables and relationships in both conditions: 73 for USER+AIDE, 66 for USERALONE. This difference is not significant, so better performance is not due to subjects simply seeing more of the data in the USER+AIDE condition. It is also not the case that subjects in the USERALONE condition never happen upon the relationships and patterns suggested by AIDE in the USER+AIDE condition. Of all the correct suggestions AIDE made about each dataset, only one was not also tried by subjects in the USERALONE condition.

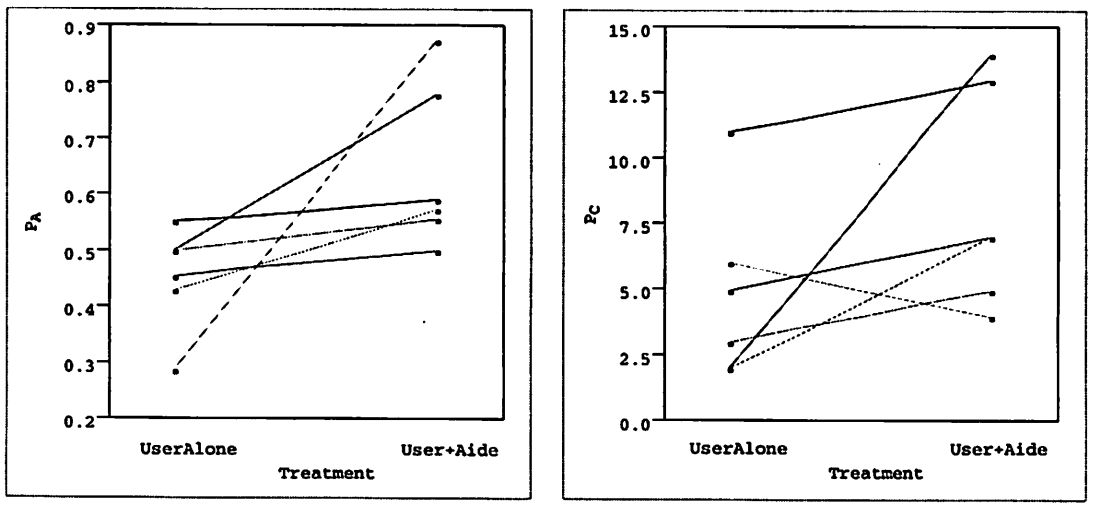
Table 8.1. $\overline{p}$ and $k\overline{p}$ per subject

	$\overline{p}$		$k\overline{p}$	
	USER+AIDE	USERALONE	USER+AIDE	USERALONE
Subject 1	0.29	0.34	4.0	5.5
Subject 2	0.39	0.29	3.5	3.5
Subject 3	0.50	0.21	3.0	1.5
Subject 4	0.56	0.37	10.0	7.0
Subject 5	0.44	0.29	4.0	2.0
Subject 6	0.34	0.50	4.5	5.5
Subject 7	0.50	0.07	3.0	1.0
Subject 8	0.59	0.36	6.5	1.5
Means	0.45	0.31	4.8	3.4

Results	t-statistic = 2.217 $Pr(> t) = 0.031$	t-statistic = 1.808 $Pr(> t) = 0.055$

The result of this comparison is central to this dissertation. It tells us that AIDE contributes significantly to the correctness of a given user's observations, on average, and that AIDE contributes to a higher total number of correct observations as well.

Knowing that data analysis strategies vary from individual to individual, we immediately ask whether this result holds across all subjects. Our experiment design does not allow us to examine interactions in an analysis of variance: we have only one case per Subject/Condition combination. Still, in a graphical display of these factors (Figure 8.3), we see that levels of improvement differ between the USERALONE condition and the USER+AIDE condition. Each line represents one subject. A good proportion of the lines are approximately parallel, which means that the improvement from the USERALONE condition to the USER+AIDE condition for some subjects was about the same. On the other hand, the slopes of some lines are quite different, indicating that the benefit varies between some subjects in in moving from the USERALONE condition to the USER+AIDE condition. These patterns hold for both $\overline{p}$ and $k\overline{p}$. We reach the plausible conclusion that Condition is not independent of Subject: while AIDE helped all the subjects, it helped some more than others.



$\overline{p}$ $k\overline{p}$

Figure 8.3. Performance of subjects in both conditions

Let's move to the second hypothesis. How does AIDE affect the confidence of subjects in the results they produce? If we combine confidence values (taking "high" as 1, "low" as 0) for all observations made by each subject, we arrive at a measure of the confidence of a subject during each condition. The mean confidence C_M of subjects in the USER+AIDE condition turns out to be very close to that of subjects in the USERALONE condition, with confidence in the USER+AIDE condition being slightly but not significantly lower. Table 8.2 gives a summary of the comparison. We use a one-tailed t-test because the alternative hypothesis is that AIDE will degrade confidence; the resulting value turns out not to be significant for either the one- or two-tailed test.

This raises the further question of whether subjects have different confidence in observations that turn out to be correct than they do for incorrect observations. In fact, this is an important point: we are happy if a system makes subjects confident in their activities, but not if their results turn out to be consistently wrong. Breaking the dataset down into correct and incorrect observations gives us no further information,

Table 8.2. C_M per subject

	C_M	
	USER+AIDE	USERALONE
Subject 1	0.750	0.750
Subject 2	0.667	0.333
Subject 3	0.375	0.571
Subject 4	0.455	0.696
Subject 5	0.600	0.750
Subject 6	0.733	0.471
Subject 7	0.833	0.706
Subject 8	0.381	0.750
Means	0.599	0.628

Results	t-statistic = 0.327 $Pr(> t) = 0.753$

however. As we might expect, confidence is generally higher for correct observations than for incorrect ones. Table 8.3 shows that the relationship between C_M and Condition remains unchanged for correct and incorrect observations.

This result, though weak, is also important. We have built a system that makes some decisions on its own, but not to an extent that users are reluctant to trust its results. Confidence levels are not significantly lower in the USER+AIDE condition than in the USERALONE condition, and separating observations by correctness only reduces the differences between the conditions. Degraded user confidence can affect performance [Rouse *et al.*, 1987]; in this experiment, at least, we avoided this potentially serious pitfall.

The third issue concerns the effect of contextual knowledge on performance. Here the interpretation of the performance measures $\overline{p}$ and $k\overline{p}$ will change slightly. So far $\overline{p}$ and $k\overline{p}$ have meant the performance of a subject in a single condition. Accounting for context requires a distinction between the constituent elements of these measures—some observations apply to relationships with anonymous variables while others do not. For conciseness we'll call the former type of observation "uninformed", the latter "informed," and we'll call the variable measuring this informed/uninformed factor "Context". Now rather than having a single value of $\overline{p}$ or $k\overline{p}$ for each condition, each subject has two such values, for informed and uninformed observations. These measures are weighted to account for the uneven distribution of observations into the informed and uninformed types. The weighting is intended to account for the

Table 8.3. C_M per subject, for correct and incorrect observations

	C_M	
	USER+AIDE	USERALONE
Correct Means	0.68	0.69
Incorrect Means	0.56	0.58

following possibility. Suppose that a subject makes nine informed observations and one uninformed observation, with $\overline{p}$ scores of 0.75 and 0.25 respectively. In testing the effect of Context on performance in the usual analysis of variance, these scores would make an equal contribution, unless the number of observations that contribute to a score is taken into account. Appropriate weighting adjusts for this, giving the informed score of 0.75 a weight of 0.9, the uninformed score of 0.25 a weight of 0.1.[2]

The effects of Condition and Context on performance are shown in Table 8.4. Though subjects perform better in the case where contextual information is available, the improvement can be attributed either to Condition alone (for the $\overline{p}$ measure of performance) or the combination of Condition and Subject (for $k\overline{p}$). Collapsing this table by aggregating subjects also shows no interaction between Condition and Context. This is a disappointing result. The literature on statistical expert systems and statistical user interfaces strongly emphasizes the importance of contextual knowledge on decision-making, and we had hoped to perform a quantitative evaluation of its influence on performance. It appears though that the cues in the data were not strong enough to lead subjects directly to correct descriptions, making our experiment insufficiently sensitive to capture the effects of context.

[2]We could have taken a similar "weighting" approach in discussing our second hypothesis about confidence levels, giving each subject two weighted C_M scores, one for correct and one for incorrect observations. Because of the natural asymmetry in confidence for correct and incorrect decisions, however, it seemed better to separate the two subsets.

Table 8.4. The effect of Condition and Context on $\overline{p}$ and $k\overline{p}$

	$\overline{p}$		$k\overline{p}$	
	F	$Pr(>F)$	F	$Pr(>F)$
Condition	5.00	0.036	4.12	0.054
Context	0.72	0.405	0.14	0.705
Subject	0.61	0.740	3.90	0.007

8.2.2 Extending the Results

Performance contains at least two components: identifying a significant variable or relationship and correctly describing it. Consider two new measures, $\bar{i}$ and $k\bar{i}$, which are comparable to $\bar{p}$ and $k\bar{p}$ but are more lenient. $\bar{i}$ and $k\bar{i}$ call an observation "correct" if a direct relationship is identified, ignoring its descriptive form (linear, cluster, nonlinear, etc.) The $\bar{i}$ variable measures the mean number of such correctly identified observations, $k\bar{i}$ the total number. Table 8.5 reproduces the analysis of the effects of Condition and Context on performance, measured here by $\bar{i}$ and $k\bar{i}$. Contextual knowledge does have an effect on $\bar{i}$, the mean identification component of performance, though it has no effect on the total number, $k\bar{i}$.

This result is suggestive and intuitively plausible. As established earlier, contextual cues in the data are not strong enough to lead subjects to correct descriptions of patterns in the data. They nevertheless point in the right direction, by drawing attention to those relationships worth pursuing, giving better performance (by $\bar{i}$) in the USER+AIDE condition.

Revisiting the first hypothesis, which dealt with the effect of Condition directly on performance, the more lenient measures of performance give the same result as the original ones. Performance as measured by $\bar{i}$ and $k\bar{i}$ in the USER+AIDE condition is superior to performance in the USERALONE condition, by a greater margin than with $\bar{p}$ and $k\bar{p}$. The results are shown in Table 8.6.

Table 8.5. The effect of Condition and Context on $\bar{i}$ and $k\bar{i}$

	$\bar{i}$		$k\bar{i}$	
	F	$Pr(>F)$	F	$Pr(>F)$
Condition	7.82	0.011	3.27	0.084
Context	3.72	0.067	0.50	0.485
Subject	0.36	0.915	2.72	0.034

Returning to the second hypothesis, we find a more striking result. The original relationship between Condition and confidence remains unchanged. However, breaking down the observations by the more lenient measure of correctness uncovers a much stronger effect in one area, as Table 8.7 shows. Confidence levels for correct observations are about the same in both conditions, but for incorrect observations confidence levels are lower in the USER+AIDE condition. That is, for one interesting subset of cases subjects have more appropriate confidence levels in the USER+AIDE condition than in the USERALONE condition.

These findings accord with our intuitions. They show that AIDE contributes to identifying interesting relationships as well as describing them. They also suggest that

Table 8.6. $\bar{i}$ and $k\bar{i}$ per subject

	$\bar{i}$		$k\bar{i}$	
	USERALONE	USER+AIDE	USERALONE	USER+AIDE
Subject 1	0.538	0.455	7	5
Subject 2	0.667	0.417	6	5
Subject 3	0.875	0.285	7	2
Subject 4	0.632	0.579	12	11
Subject 5	0.556	0.500	5	3
Subject 6	0.571	0.583	8	7
Subject 7	0.500	0.429	3	6
Subject 8	0.667	0.500	8	2
Means	0.626	0.468	7.00	5.13
Results	t-statistic = 2.489 $Pr(> t) = 0.021$		t-statistic = 1.957 $Pr(> t) = 0.045$	

Table 8.7. Revised C_M per subject, for correct and incorrect observations

	C_M	
	USER+AIDE	USERALONE
Correct Means	0.660	0.661
Incorrect Means	0.477	0.558

contextual information can be helpful by guiding a subject's search toward useful areas of the search space. This leads to better performance and more appropriate confidence levels in some situations.

We can make a last fortuitous connection between Context and confidence. As we might expect, observations for relationships for which contextual information is available are are associated with higher confidence than uninformed observations. This holds across both conditions. Table 8.8 shows that both Subject and Context have a significant effect on confidence. The mean value of C_M for informed observations is estimated to be 0.69, for uninformed observations 0.51.

8.2.3 Explaining Subject Performance

Up to this point we have concentrated on demonstrating that subjects perform better with AIDE than without. We now take on the more difficult problem of explaining how the interaction works. In this section we will explore connections between performance and other factors, to gain a a better understanding of AIDE's contribution to exploration. This will involve extracting patterns from the traces of subject behavior and relating the patterns to performance results.[3] Because we are interested in the fully functional system, we will concentrate on data from the USER+AIDE condition, giving results from the USERALONE condition only where a comparison appears interesting.

[3]Most of results in this and the following section were generated in the AIDE environment.

Table 8.8. C_M per subject, for informed and uninformed observations

	C_M	
	F	$Pr(> F)$
Subject	4.18	0.039
Context	19.29	0.003

Exploratory operations in AIDE can be divided roughly into three types. Some operations are concerned with local decision-making: selecting a variable or constructing a relationship for display, examining indications, or asking the system for documentation of proposed actions. These are what we will call LocalOperations. They involve decision-making at a single focus point: assessing information about which variables and relationships it would be worthwhile to describe; evaluating the applicability of different operations and procedures to describe a potential pattern. LocalOperations account for 40% of the operations in the USER+AIDE condition. NavigationOperations are such actions as initiating the exploration of a variable or relationship or going back after generating a result to select another relationship. In other words, these operations generate new focus points, or take the exploration from one focus point to another. Navigation is responsible for almost half (44%) of the operations in the USER+AIDE condition. Finally, ManipulationOperations are a specific type of navigation operation, involving selection of the reductions, transformations, and decompositions that make changes or additions to the data. Actual data manipulation accounts for only a small portion of the total number of operations. Table 8.9 gives a statistical summary of the operations made in each condition, per subject. Because the distributions are somewhat skewed, the table presents the median and interquartile range as well as the mean and standard deviation. Statistics for the USERALONE condition are given for comparison.

These summaries give us a rough idea of how subjects went about exploring a dataset. Much of the effort, in terms of the number of operations applied, involved examining the data from different angles and evaluating ways of building descriptions. Subjects showed a good deal of mobility, not just in moving from one data structure to the next, but also in moving from one point in the network of exploratory plans and actions to another. This point was also emphasized by most of the subjects in their assessments: a common theme was the importance of being able to navigate through

the exploration process. The summaries also show that data manipulation was secondary to other activities; we might even think of navigation and local evaluation of decisions as setting the stage for data manipulation.

These variables are related as shown in the model in Figure 8.4, produced by AIDE's causal modeling procedures. We have eliminated TotalOperations from this model, because, as we know, it is just a linear function of the other three variables. The relationship between NavigationOperations and LocalOperations is relatively strong ($r = 0.67$), as is the relationship between NavigationOperations and ManipulationOperations ($r = 0.54$). The variables ManipulationOperations and LocalOperations are weakly correlated to begin with ($r = 0.29$), and if we hold NavigationOperations constant the correlation drops to 0.12. Exploration of the relationships in this model shows no unusual patterns.

Causal direction in this model is unclear; there are a few plausible but conflicting interpretations. Data manipulation operations generate new data, which can then be explored; thus the number of ManipulationOperations influences NavigationOperations. Increased navigation leads in turn to the generation of new focus points, which requires more local decision-making, which means NavigationOperations influences LocalOperations. On the other hand, it could be argued that local decision-making

Table 8.9. Summary of operations selected, per subject

Command	Condition	% Total	Mean	SD	Median	IQR
TotalOperations	USER+AIDE		331	158	361	297
	USERALONE		191	83	180	144
LocalOperations	USER+AIDE	38%	127	84	118	134
	USERALONE	73%	140	81	143	122
NavigationOperations	USER+AIDE	44%	146	73	137	148
	USERALONE	12%	22	5	21	9
ManipulationOperations	USER+AIDE	13%	44	37	28	65
	USERALONE	9%	17	21	9	24

provides information necessary for navigation, and that this navigation turns up new opportunities for data manipulation. The direction of these associations remains ambiguous.

We can connect our modeling efforts with the results of the last section by bringing performance measures into the picture, as shown in Figure 8.5. As expected, the identification of a relationship influences its correct description (i.e. $\bar{i}$ influences $\bar{p}$.) Otherwise, $\bar{i}$ is influenced by NavigationOperations, while both $\bar{p}$ and $\bar{i}$ are influenced by ManipulationOperations. This latter set of influences is interesting, in that explicit data manipulation accounts for relatively little of the total effort a

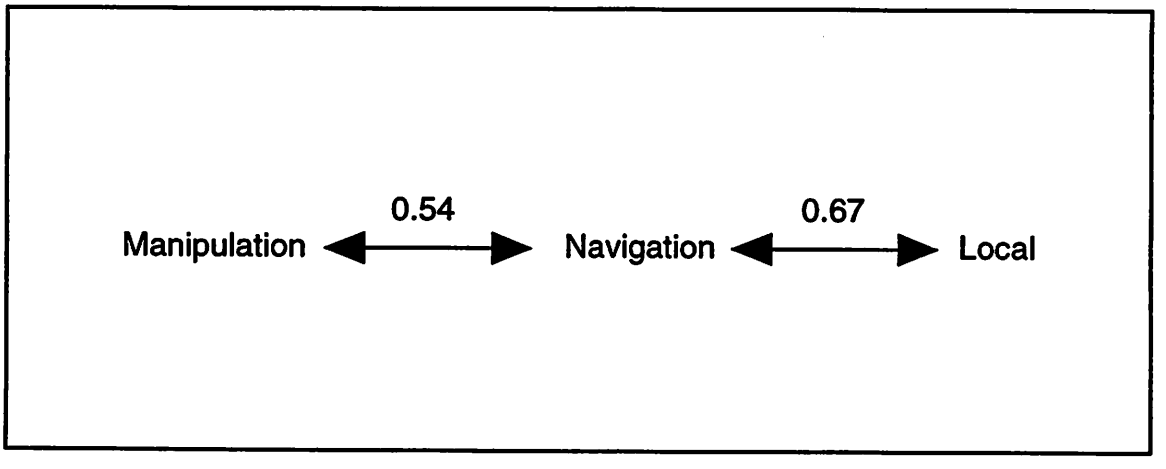


Figure 8.4. Model of operation counts

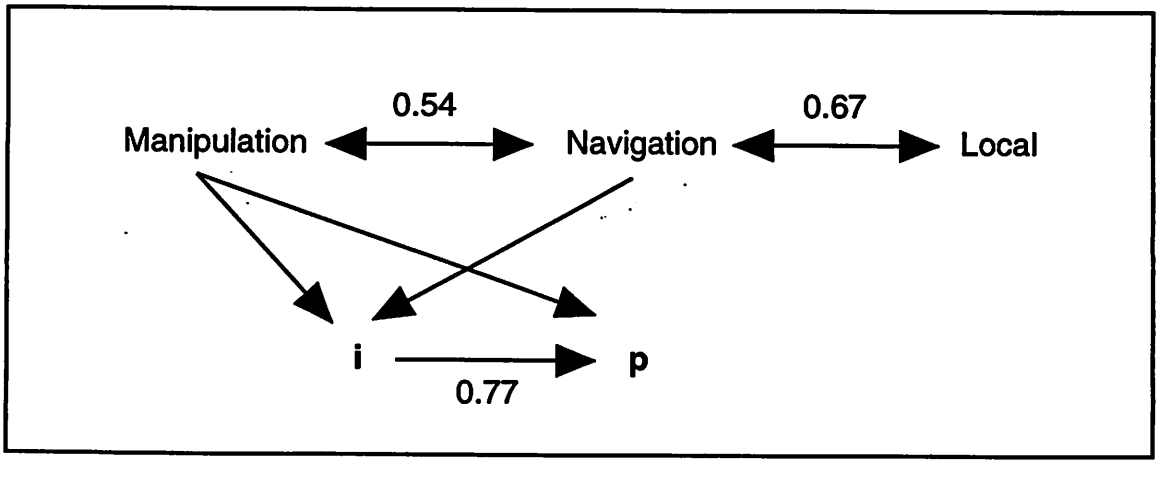


Figure 8.5. Model of operation counts, plus performance

subject puts into the exploration, but is a factor in determining both performance measures. A plausible explanation is that data manipulation operations are generally applied only when one perceives some kind of pattern. These operations give further information about the pattern, and thus a greater number of such operations leads to more accurate observations. This model gives us a set of plausible, tentative notions about how subject behavior, in terms of the types of operations applied, influences performance.

8.2.4 Explaining AIDE's Behavior

So far our analysis has concentrated on the behavior and performance of individual subjects. We would like to bring in other variables that influence performance, such as the type of pattern involved, or whether an observation applies to anonymous variables, but we run into a problem: these properties characterizes observations, not subjects. It makes little sense to compute some kind of context or pattern-type score for each subject, averaging over all observations.

Let's take a different tack. If we are willing to aggregate the results of all the subjects, we can build a tentative model of AIDE's behavior based on the properties of individual observations, rather than of individual subjects. These are the observations made during the USER+AIDE condition. This brings to light some interesting properties of AIDE.

In our analysis up to this point we have considered the individual subject as a single case. Thus, because we have eight subjects, our analysis is based on eight cases, over performance variables ($\overline{p}$, $k\overline{p}$), a confidence variable (C_M), and so forth. Now, because we are treating observations as cases, we have many more cases, and we can use properties of individual observations as variables, such as these:

- O_p: whether an observation correctly describes a variable or direct relationship in the data. O_p is 1 if correct, 0 otherwise.

- O_i: whether an observation identifies a direct relationship in the data. O_i is 1 if correct, 0 otherwise.
- *Suggestion:* whether AIDE suggested a variable or relationship for exploration. A common theme in subject assessments was the helpfulness of AIDE's suggestions in imposing structure on the exploration. For some subjects this even led to a difference in the distribution of observation times: in the USER+AIDE condition most observations were clustered toward the beginning of the trial, while in the USERALONE condition they were more evenly distributed.
- *Informed:* whether a variable had a real or anonymous name. As we saw earlier, context has a significant relationship with subject confidence and with the performance measure $\bar{i}$. Here we will examine the relationship between the presence of context and the accuracy of observations for AIDE as a whole.
- *Pattern type:* the type of pattern identified. Because AIDE has different plans to suggest and generate descriptions of different types of patterns, we expect to see some interaction between this variable and our performance measures.

Our procedure did not collect the specific suggestions AIDE made to the subjects. We can retrace exploratory sessions to some extent, however, to record AIDE's initial suggestions about which variables and relationships to examine. This lets us construct a binary variable, Suggestion, which records whether a variable or relationship was one of AIDE's suggestions.[4] Suggestion is thus related to the performance measures that deal with identifying variables and relationships, O_p and O_i.

Evaluated in aggregate, AIDE's initial suggestions perform reasonably well, averaging $\bar{i} = 0.583$ and $k\bar{i} = 7.0$ over both datasets. Both of these values are significantly greater than the performance of subjects in the USERALONE condition; they fall near the low end of performance values in the USER+AIDE condition.

[4]Unfortunately, our data collection was not detailed enough for us to reconstruct the suggestions AIDE made about which operations to apply, once a data structure has been selected. Exploration of issues in this area must await further experimentation.

However, performance cannot be so simply decomposed. Subjects often rejected AIDE's suggestions, and tried more possibilities on their own initiative. The resulting observations thus depend not only on AIDE's efforts, but also on the interpretation of the subjects. The relationship between Suggestion and O_i is given in Table 8.10, in which "1" means that a variable or relationship was suggested by AIDE, "0" that it was not. The relationship has a χ^2 of 3.298, with a significance of 0.069. AIDE's suggestions are in general effective.

Clearly there will be a strong relationship between O_i and O_p. Any relationship that is correctly described must first be correctly identified, which imposes a strong dependence between the two types of correct observations, as well as incorrect ones. The relationship between these two variables is shown in Table 8.11. The relationship has a χ^2 of 47.665, which has a significance level lower than 0.0001.

The model in Figure 8.6 reflects our initial understanding of relationships in the data. Now, AIDE's suggestions are made at variable focus points that manage decisions about which data structures to explore. At each focus point, activation and

Table 8.10. Counts of Suggestion and O_i

	Suggestion = 0	Suggestion = 1	Totals
$O_i = 0$	14	18	32
$O_i = 1$	14	42	56
Totals	28	60	88

Table 8.11. Counts of O_p and O_i

	$O_p = 0$	$O_p = 1$	Totals
$O_i = 0$	31	0	31
$O_i = 1$	12	42	54
Totals	43	42	85

preference rules consider the types of patterns present in the data when making their decisions. Thus we might expect to see a relationship between Suggestion and O_p, as shown in Table 8.12, but we find none. This is actually not surprising: suggestions at the variable focus points can only point toward data with potentially interesting patterns. These patterns are extracted through plans that are activated later, and the decisions at these later stages will have a much stronger influence on the correctness of the resulting descriptions. Thus we make no changes to our initial model.

Earlier we saw an effect of contextual knowledge on the performance measure $\bar{i}$, and thus we might wish to add the variable Context to our model, for its influence on O_p and O_i. However, we can find no relationship between Context and O_i. The explanation is that we are now considering only a subset of the data, from the USER+AIDE condition, and within this condition Context has no significant effect

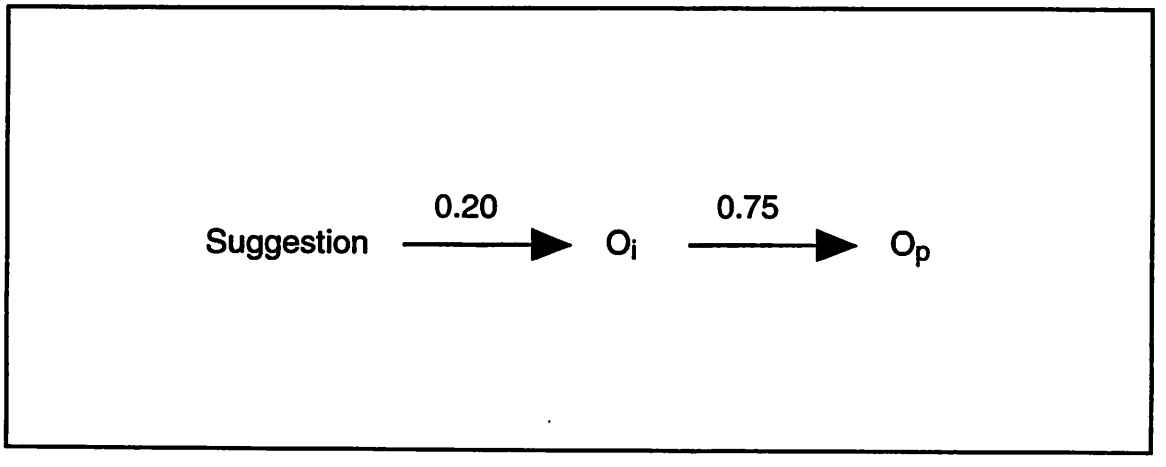


Figure 8.6. Initial model of performance relationships

Table 8.12. Counts of Suggestion and O_p

	Suggestion = 0	Suggestion = 1	Totals
$O_p = 0$	15	28	43
$O_p = 1$	13	29	42
Totals	28	57	85

on performance. Looking at other variables, though, we can extend the model in a different direction. The variable Pattern-Type interacts with both O_p and O_i, as shown in Figure 8.7. That is, the likelihood that a pattern is described correctly depends on its type. This is plausible, because different plans are used to generate different types of patterns. Thus Pattern-Type and O_p should be related. Because activation and preference rules depend on patterns in the data, Pattern-Type and O_i are also related, though to a slightly lesser extent. Further exploration of the data turned up no significant additional information.

8.2.5 Behavior Traces

Finally, our data collection included time series information: which operations subjects executed, and when. We examined the traces manually and ran them through a pattern detection algorithm called MSDD [Oates and Cohen, 1996]. MSDD performs a systematic search of potential dependencies in multivariate time series data. Our analysis turned up a great many correct but uninformative results, such as, "If a subject has selected some data structure for exploration, then the next operation is very likely to be a data manipulation operation," and, "If the current operation deals with relationship (x, y), then the next operation is very likely to involve (x, y)." The

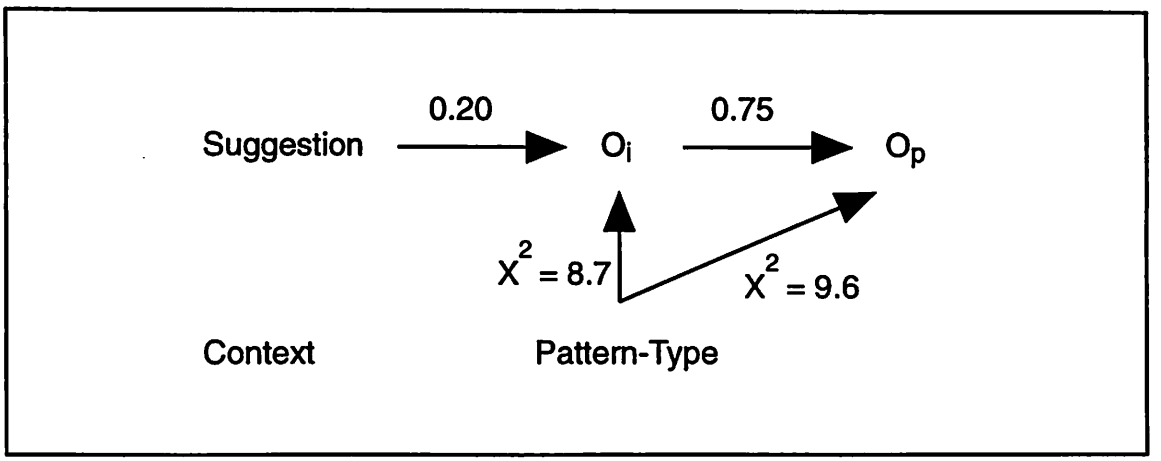


Figure 8.7. Revised model of performance relationships

results were sometimes ambiguous as well: it turned out that if a subject examines the same relationship for several operations in sequence, this is equally predictive of (a) continuing to examine the same relationship, and (b) switching to a different relationship. Unfortunately, we were unable to detect patterns that were both strong and not already obvious.

8.3 Discussion

Our experimental results support the claim that AIDE's mixed-initiative planning approach to exploration provides a significant benefit to users. One might raise questions, however, about the effectiveness of the demonstration.

An objection might be raised about the scale of the experiment. Only eight subjects were used. This number is slightly deceptive, in that we were able to collect a great deal of data for each subject. Each subject entered about fifteen observations about each dataset and generated well over five hundred nontrivial actions in carrying out the exploration. The real issue, though, is the strength of the effects. Differences in performance between the two conditions turned out to be so great that they were obvious even on the surface.

One might then argue that we have given subjects, in the USERALONE condition, tools that are too weak for effective exploration. Subjects are provided with tools comparable to those found in the CLASP package. In most cases the functionality provided by AIDE in the reduced condition is identical to CLASP functionality, the only necessary difference being in the style of interaction. Thus if CLASP provides an effective exploration environment, then AIDE in the USERALONE condition provides one as well. CLASP provides a set of sophisticated tools comparable to many packages, all of which compare favorably with the pencil-and-paper approach advocated by Tukey and others [Tukey, 1977] as being adequate for the task of EDA.

Finally, since the experiment involves subjects constructing and describing the details of a kind of causal model, we might wonder how an autonomous causal

modeling would fare on the data. There are two possible responses to this point. First, AIDE does include conditional independence heuristics in its preference rules. AIDE's selection of "interesting" relationships is thus based partly on whether it considers the variables involved to be directly related or not. The conditional independence heuristics alone, however, perform relatively poorly by the $\bar{i}$ and $k\bar{i}$ measures, significantly worse than subjects in the USER+AIDE condition and slightly worse than subjects in the USERALONE condition. We also note that the heuristics themselves cannot generate the types of descriptions evaluated by $\bar{p}$ and $k\bar{p}$. Second, we must remember that causal modeling algorithms are no more autonomous than multiple regression or clustering algorithms. That is, they may run without input from the user, but their performance depends strongly on their input data and parameter settings, which in turn depend strongly on the assumptions and observations one makes about the data.

To summarize, our experimental results are sound, within their clear limitations. Perhaps surprisingly, comparable experiments are rarely found in the literature of statistical expert systems and other exploration-related work. As simple as it might appear, testing a system like AIDE is an arduous task. There is no simple, unobtrusive way to run a controlled experiment to test such a system, which forces us to construct a set of explicit, artificial procedures. Testing must involve human participation, which introduces sources of variance that we would never need to consider in testing an autonomous system. Training procedures must be designed and tested; lengthy trials must be scheduled and overseen; large amounts of data must be filtered. It may be the difficulty of running such tests and of perhaps facing equivocal results that makes researchers shy away from the task. Experimental evaluation is however a necessary step. Our results, while preliminary, support the general claims.

We end with a few of the comments subjects made about their experience, comments that offer further insight into how and why AIDE works. The original text has been revised slightly to match the terminology of this chapter.

- "When I decided to stop exploring the dataset without AIDE, it was primarily because my search for structure had degenerated into random search. However, when I stopped exploring the dataset with AIDE, I was much more confident that I had found most (if not all) of the structure in the data. The reason is that there seemed to be a fairly smooth progression in the strength of the relationships that AIDE suggested that I explore."
- "In the USERALONE condition, I stopped when I couldn't find any more interesting correlations or any simple indications like gaps in a histogram. In the USER+AIDE condition, it was when AIDE ran out of suggestions and I had finished all the interesting tangents I saw on my way through the suggestions. In both cases, I felt like going any further would be like blind search."
- "For the most part, AIDE's first choice was the one I would have made (about 75% of the time). The only problems I see are user-interface related; a little work in this area and extension of the statistical capabilities would make it far better than any 'normal' statistical package."
- "In general, I vastly preferred the USER+AIDE treatment to the USERALONE treatment. First, AIDE provided an overall summary of the dataset that aided my exploration. Second, AIDE assisted with some low-level details that let me focus on the high-level ideas... Finally, AIDE gave me the sense of being "supported". That is, I felt more capable when using AIDE than when I was not."
- "I was surprised at how well AIDE worked in one regard. I expected to be frequently surprised by what AIDE did. Instead, I was surprised only in retrospect: when I hit the BACK button, I thought that AIDE has only made a single decision for me. I discovered, in general, that AIDE had made several inferences and decisions. This highlighted the ability of users to easily "jump ahead" when AIDE takes several steps."

- "I spent a lot of time (read: actions) lost in navigation due to being unfamiliar with the USER+AIDE and USERALONE conditions... Both conditions have fairly limited (statistical) functionality..."

In addition to giving us a better intuition about how AIDE works, the comments point out some clear directions for future research.

8.4 Summary

Evaluation focused on the simple hypothesis that exploration is more effective with AIDE than without. The experiment involved testing subjects under two conditions. In the USER+AIDE condition, subjects explored a dataset with AIDE's help, while in the USERALONE condition, subjects explored a dataset in a similar statistical computing environment but without active help from AIDE. AIDE's effectiveness was then determined by measuring differences in performance between the two conditions.

We defined several related measures of accuracy: the average number of direct relationships correctly identified and described, over all notations made; the total number of correct notations; the average number of direct relationships correctly identified, without regard to their correct description; the total number of correct identifications. Performance was significantly higher for subjects in the USER+AIDE condition for all of our measures. The comparison tells us that AIDE contributes significantly to the correctness of a user's observations, on average, and that AIDE contributes to a higher total number of correct observations as well. That is, given our experimental conditions, users can perform EDA better with the help of AIDE than they can alone, and better than AIDE acting alone.

We also made measurements concerning the confidence of subjects in their results and whether the presence of meaningful variable names in the datasets made a difference in their performance. Our results were suggestive but not conclusive in all cases. Briefly, AIDE did not significantly reduce the confidence of subjects in

the observations they made, a concern for any system that takes responsibility for significant portions of a complex task. It also appears that appropriately named variables help to focus exploration on relevant parts of the search space, though the interaction between this knowledge and AIDE's assistance remains unclear.

Finally, for a better view of AIDE's contribution to the exploration process, we divided subject actions into categories and built models of the relationships between the different types of actions. We found that most of the effort, in terms of the number of operations applied, involved examining the data from different angles and evaluating ways of building descriptions. Subjects showed a good deal of mobility, not just in moving from one data structure to the next, but also in moving from one point in the network of exploratory plans and actions to another. These preliminary activities dominate the exploration and set the stage for the actual data manipulation operations.

Table 8.13. Functional relationships in MODEL A

Variable	Name	Alias	Computation
v_1	longevity		$v_4 = \text{A} \rightarrow (v_2 + v_3 + U(25,30))/25$ $v_4 = \text{B} \rightarrow (v_2 + v_3 + U(50,60))/25$ $v_4 = \text{p} \rightarrow (v_2 + v_3 + U(75,80))/25$
v_2	%function		$\lvert v_5 \rvert^{1.9} + err(v_5/2)$
v_3	%survival		$5v_5 + err(v_5/2)$
v_4	stain		$v_6 < 2 \rightarrow \text{A}$ $v_6 < 3 \rightarrow \text{B}$ otherwise $\rightarrow \text{p}$
v_5	exposure-diff		$v_8 = \text{black} \rightarrow \lvert v_9/24 \rvert + err(1.5)$ $v_8 = \text{brown} \rightarrow \lvert v_9/20 \rvert + err(1.5)$ $v_8 = \text{grey} \rightarrow \lvert v_9/17 \rvert + err(1.5)$
v_6	duration		$U(1,4)$
v_7	%absorption		$2.7v_6 + err(2)$
v_8	political-party	V28	$v_{10} + v_{11} < 10 \rightarrow$ grey $v_{10} + v_{11} < 15 \rightarrow$ black otherwise $\rightarrow$ brown
v_9	months-retired	V19	$\lvert v_{10} \rvert^{1.5} + err(x)$
v_{10}	age-diff	V210	$2v_{12} + v_{13} + v_{14} + err(4)$
v_{11}	income-class	V11	$U(1,4)$
v_{12}	longevity*	V212	$U(1,4)$
v_{13}	marriages	V13	$U(1,4)$
v_{14}	children	V614	$U(1,4)$
v_{15}	education		$\Pr = 1/3 \rightarrow U(1,4)$ $\Pr = 1/3 \rightarrow U(9,15)$ $\Pr = 1/3 \rightarrow U(14,19)$
v_{16}	income	V16	$\Pr = 1/3 \rightarrow U(17,19)$ $\Pr = 1/3 \rightarrow U(31,35)$ $\Pr = 1/3 \rightarrow U(38,44)$

Table 8.14. Functional relationships in MODEL B

Variable	Name	Alias	Computation
v_1	%function	V91	$\lvert v_4\rvert^{1.9} + err(v_5/2)$
v_2	%survival	V52	$5v_4 + err(v_4/2)$
v_3	stain	V83	$v_5 < 2 \rightarrow$ A $v_5 < 3 \rightarrow$ B otherwise $\rightarrow$ p
v_4	exposure-diff	V34	$\lvert\gamma(3)\rvert$
v_5	duration	V75	$U(1,4)$
v_6	%absorption	V26	$2.7v_5 + err(2)$
v_7	total-years		$v_8 =$ black $\rightarrow \lvert v_9/24\rvert + err(1.5)$ $v_8 =$ brown $\rightarrow \lvert v_9/20\rvert + err(1.5)$ $v_8 =$ grey $\rightarrow \lvert v_9/17\rvert + err(1.5)$
v_8	political-party		$v_{10} + v_{11} < 10 \rightarrow$ grey $v_{10} + v_{11} < 15 \rightarrow$ black otherwise $\rightarrow$ brown
v_9	months-retired		$\lvert v_{10}\rvert^{1.5} + err(x)$
v_{10}	age-diff		$2v_{12} + v_{13} + v_{14} + err(4)$
v_{11}	income-class		$U(1,4)$
v_{12}	longevity	V212	$v_3 =$ A $\rightarrow (v_1 + v_2 + U(25,30))/25$ $v_3 =$ B $\rightarrow (v_1 + v_2 + U(50,60))/25$ $v_3 =$ p $\rightarrow (v_1 + v_2 + U(75,80))/25$
v_{13}	marriages		$U(1,4)$
v_{14}	children		$U(1,4)$
v_{15}	education	V15	Pr $= 1/3 \rightarrow U(1,10)$ Pr $= 1/3 \rightarrow U(9,14)$ Pr $= 1/3 \rightarrow U(16,33)$
v_{16}	income		Pr $= 1/3 \rightarrow U(7,10)$ Pr $= 1/3 \rightarrow U(21,35)$ Pr $= 1/3 \rightarrow U(38,40)$

CHAPTER 9

CONCLUSION

We set out with ambitious goals in building AIDE. We wanted a general-purpose assistant to help us with the difficult task of data exploration. The attempt has been successful in many ways, but a number of questions have been left open. In this chapter we'll review what has been accomplished, discuss the limitations of the research, and comment on what remains.

9.1 Revisiting the Contributions

AIDE's contributions fall into three areas: a recognition of the relationship between planning and EDA, the development of a planner-based system that exploits this relationship, and, most importantly, a demonstration that a mixed-initiative approach to data exploration can be effective.

AIDE provides a novel, effective approach to strategic EDA processing. Recall Hand's definition: a statistical strategy is a formal description of the actions and decisions involved in applying statistical tools to a problem [Hand, 1986]. Earlier approaches to statistical strategy have concentrated almost solely on the representation of decisions. For example, in the regression expert REX, a great deal of effort went into generating the rules that constitute the strategy, specifying the structures that represent relevant knowledge and data, and designing modules for user interaction. Actions, in contrast, are viewed as "convenient combinations[s] of operations from the data tending modules." [Gale, 1986, p. 202] With the exception of TESS [Lubinsky and Pregibon, 1988], exploratory systems tend to concentrate on the difficulty of making good decisions, rather than the representational power of actions and their strategic combination.

AIDE considers the other half of the picture, how the actions that constitute a strategy can be effectively represented and organized. AIDE's primitive operations provide a strong representational foundation for EDA. The operations are flexible and general, as we have seen in many real-world and artificial examples of data manipulation. The representation is comparable to that of existing systems (e.g., IMACS [Brachman *et al.*, 1992] and IDES [Goldstein and Roth, 1994]) but has closer ties to traditional EDA, capturing most common exploratory procedures and exposing strong similarities in their internal structure.

AIDE's plans represent procedural knowledge about exploration, something earlier attempts have largely neglected. Rather than generating procedures from scratch each time a specific type of analysis is called for, AIDE can retrieve and apply existing, well-tested plans. Flexibility is nonetheless maintained in the planner's ability to combine plan fragments and to switch dynamically between partially-executed plans.

AIDE's plans are furthermore designed to accommodate the superior knowledge of context a user brings to the analysis. A plan is not simply a compiled program that, once started, runs without interruption until complete. Rather, a plan relies on explicit decision-making rules for activating and imposing preferences on its choices between constituent actions. Its decision points remain active over the course of its execution, to be reviewed and modified by the user whenever desired. AIDE has autonomy, but within limits: the user is always free to take charge of the exploration.

Another of AIDE's contributions is its recognition that the EDA problem can be solved by a very specific type of planning, partial hierarchical planning. EDA involves not only the general properties of hierarchical problem decomposition, abstraction, and macro operators, but also interleaved plan generation and execution, explicit control structures, and flexibility in the representation of primitive objects and actions.

The design of the planner reflects these observations. It integrates a number of features from existing planners: control constructs for sequencing, iteration, and con-

ditionalization from the RESUN planner [Carver and Lesser, 1994]; focus points, again from RESUN; representation of plans and actions—in fact, all planning structures—as first class objects with inheritance, as in the PHOENIX planner [Howe, 1993]; a co-routining planner and meta-planner, a variation on the design of many other planners. AIDE's design incorporates sophisticated features while still remaining straightforward and extensible.

The partial hierarchical planning design is also well-suited to human-computer interaction, a novel area of application. The explicit procedural structures give strong contextual clues to help users locate themselves in the exploration process. This representation opens the doors to a very useful functionality. For example, in discussing the challenges of programming in the user interface, Ben Shneidermann includes requirements of sufficient computational generality (including conditionals and iteration), modularity, and reliance on existing libraries of functions [Shneiderman, 1992]. Such requirements, applied to a planning system, lead directly to the expressiveness of partial hierarchical plans.

Finally, our experiments show that a mixed-initiative approach to data analysis is feasible. In contrast to conventional systems that either make all the strategic decisions internally or no strategic decisions at all, AIDE attempts to strike a balance between the two extremes. With AIDE the user can deal with EDA at a more abstract level, without giving up the ability to control the details of the exploration whenever necessary. In the most general terms, the results give evidence that our planning approach can help to solve the "mind-reading" problem: the system can carry out direct instructions but can also anticipate the user's needs, providing intelligent, responsive assistance.

9.2 Limitations

This work has limitations in both what has been accomplished and what we have been able to demonstrate.

Our claims about the effectiveness of AIDE might lead one to ask why it should not immediately become a commercial product. One serious obstacle is its inefficiency. A general statistical user interface must provide broad coverage, and much of the code that implements AIDE's statistical processing needed to be written from scratch. While the CLASP implementation provided code for statistical tests, summary statistics, and a few modeling procedures, there were no existing sources for robust statistics, clustering, resistant line fitting, and various other specialized procedures. AIDE is a large system, containing almost 24,000 lines of code (8,500 lines in the interface) and relying on external systems such as CLASP and various machine learning sources for another 25,000 lines.[1] Given the natural limits on my programming efforts, I often had to sacrifice efficiency for added functionality. In practice AIDE can handle datasets of no more than about fifteen variables and a hundred or so observations. Some of its operations run in reasonable time, but others run very slowly. This problem is exacerbated by AIDE's semi-autonomy: when the user instructs AIDE to perform a simple operation, the system may execute many more in its efforts to anticipate the user's needs. While AIDE's plans were specifically designed to incorporate information about expected running time, accuracy of results, and related measures, the current implementation does not store this information. Efficiency may seem an unreasonable concern for a research prototype, but slow software puts practical limitations on the sophistication of the system.

The sophistication of the strategies in the plan library is another issue. They provide a sampling of common exploratory techniques, but many possibilities for expansion remain open. The regression strategies should be made much more detailed; evaluation of clustering results should be strengthened; opportunities for generalization of numerical patterns should be considered in more depth. Though it appears to be a very promising area of research, interaction between exploration and modeling

[1]These are rough maximum estimates, which should be reduced by perhaps 10% or 15% to account for comments.

is limited. Most importantly, AIDE's strategies are still at a relatively low level of abstraction, and they interact very little with one another. We can imagine strategies, however, that build more sophisticated models of the user's activities, perhaps making hypotheses about very high level user goals. Such strategies could potentially raise the level of interaction with AIDE to a much more effective level.

The third area deals with the interface. Interface design and implementation is a complex, time-consuming process. Because AIDE's development concentrated on what goes on beneath the surface, the interface has many faults. There are inconsistencies in the way choices are presented to the user; the style of interaction is overly modal; some useful features and behaviors are insufficiently documented; error handling is primitive. The subjects in our experiment found the interface adequate, but they were able to point out many small problems in addition to the ones mentioned.

Finally, our experiments are only a beginning in exploring the limits of AIDE's behavior. We have shown that AIDE's mixed-initiative approach to data exploration is feasible, a necessary first step. We have a grasp of the factors that make the arrangement effective. With this initial understanding we can now pose much more refined questions. How does the type of problem influence AIDE's effectiveness? Would experts perform differently from novices using AIDE? How would performance change if specific strategies were modified or even removed from the plan library? How does the "aggressiveness" with which AIDE takes the initiative affect user confidence? How much weight do AIDE's suggestions carry in different situations? We need to answer many more such questions if we are to gain a detailed understanding of why AIDE works.

Fortunately, addressing problems in these areas does not require revision of our basic concepts. Improving efficiency is a simple matter of programming. Adding to the plan library requires plan design skills and perhaps some knowledge engineering, but we could incorporate many procedures directly from the statistical and machine

learning literature. The interface needs a good deal of attention, but many of the improvements needed are independent of the underlying system design. Finally, further experimentation can only help us to build better versions of the system.

9.3 Future Work

We set out with ambitious goals for AIDE. Many of them remain just that: unachieved goals. Still, AIDE's development has provided better insight about how these goals can be reached.

We have thought of AIDE almost from the start as a system that should be able to learn from experience. The plan language is essentially a high level programming language, which makes learning no easy task. This is an open area of research in at both planning and human-computer interaction. In the planning community researchers have recently become interested in planners that, given a plan constructed by humans, can recapture the rationale used in its construction. HCI researchers have been interested in this problem for a much longer time, in the form of demonstrational programming. A demonstrational programming system induces a program from a sequence of operations executed by a user. While this might seem to be a simple matter of recording macros, there are difficult problems in determining the scope of variables, deciding which actions should be generalized and which should remain constant, and representing control.

In the domain of EDA, and specifically in AIDE's design, we have some advantages in solving the problem. HCI research often deals with drawing or text editing tasks, where the system has little hope of understanding contextual clues. Recapturing rationale, in the planning world, faces a different problem: the plans that must be analyzed are usually constructed off-line, rather than under the direct supervision of the planner. With AIDE we have workable plans designed to reflect common exploratory behavior; we also have specifications of the conditions in which these plans are applicable. Further, because users generate sequences of operations in AIDE's own

environment, the system has access to much of the contextual information that can affect their decisions. In this arrangement it seems plausible that AIDE could learn new plans and control knowledge by modifying its existing structures.

Another area for improvement in AIDE is its interactions with specific users. Interaction between human problem-solvers can go smoothly or badly, depending on the characteristics of their working relationship; an improved AIDE should be responsive to the exploration styles of individual users. This has close ties to learning new control knowledge. For example, AIDE might have a general preference rule that specifies Plan-A should be tried before Plan-B, but because of the exploratory style used by a particular user, a more specific rule specifies that the plans should run in the reverse order.

More generally, AIDE offers a promising approach to the representation and application of strategic knowledge in domains other than statistical data exploration. It is easy to think of computer-supported but (human) labor-intensive tasks, tasks in which competent users can capitalize on abstraction, problem decomposition, and procedural knowledge. Possible examples include experiment design, data visualization, information retrieval, knowledge discovery in databases, software verification and validation, and computer-aided design and modeling in various fields.[2] Given the chance to apply AIDE's techniques in other arenas, we will be able to generalize and refine our basic approach.

9.4 A Final Word

I will end this work with another of John Tukey's incisive observations about EDA [Tukey, 1982, p. 10]:

> Data analysis is like using an Erector or Meccano set, where we put together as many pieces as may be necessary for our present problem.

[2]These examples are the result of conversations with researchers in these areas who have shown interest in AIDE's general approach to problem-solving.

Tukey's comment emphasizes the flexibility of data exploration. We apply just those tools we need to the problem at hand, and we adapt our techniques as necessary to meet new problems. This characterization however incomplete. In addition to the knowledge of the specific tools, or building blocks, of exploration, we also rely strongly on knowledge about the complex ways in which the tools can be applied. Just as putting together building blocks can be guided by architectural principles, data exploration can be controlled and guided. This work is an attempt to identify such principles and to understand how they can be incorporated into an automated system.

APPENDIX A

AIDE TEST DATASETS

The development of AIDE involved a close interaction between design and testing. These are some of the datasets which played a role in AIDE's development:

age-height Age and height data for schoolchildren; resistant line fitting [Hoaglin *et al.*, 1985].

autos Automobile features (e.g., weight, length, engine displacement); clustering and linear modeling [Fowlkes *et al.*, 1987].

breast Classification of breast cancer patients [Murphy and Aha, 1993].

commitment-data Phoenix dataset; relationship between planner commitment, environment variability, and performance.

entomology Caterpillar habitat and behavior dataset; time series analysis.

acker Tests of human judgment of cohence; categorical data [Cohen, 1995].

nrc-rankings Computer science graduate school rankings; clustering.

population-data U.S. census data; function fitting [Tukey, 1977].

speed-brake Automobile speed and braking distance data; clustering and function fitting.

spiders Spider habitat and behavior dataset; time series analysis.

time-series TRANSSIM ship, port, and dock activity; time series analysis.

Other datasets from the machine learning repository [Murphy and Aha, 1993] were also used at times to test specific plans. Artificially generated data played a role as well.

APPENDIX B

AIDE PLANS

The plan describe-relationship-by-regression-fit establishes a subgoal to generate a regression fit for the relationship. It adds a layer of indirection so that other plans can use the same subordinate plan it triggers.

```
(define-plan describe-relationship-by-regression-fit
    (linear-fit-plan)
  ""
  :satisfies (linear-fit :regression-fit ?model
                         ?structure ?fit-relationship)
  :body (:SUBGOAL generate-fit-subgoal
                  (parameterized-description
                   :fit ?model ?structure regression-fit
                   ?fit-relationship))
  :translation "Linear regression fit."
  :operation-class :least-squares)
```

Goal classes are defined so that their execution can be specialized. The same is true of focus points.

```
(define-goal-class fit-evaluation-goal ()
  "Evaluating a fit."
  :roles ((?fit-relationship :source)
          (?deepening-result :result)))

(define-plan-focus-point-type
    fit-relationship-focus-point
    fit-relationship)
```

The fit-relationship plan simply generates a fit and evaluates it. Its definition is generic to provide for future expansion.

```
(define-plan fit-relationship ()
  "Describe a relationship by fitting a function to it.
   Evaluate the fit."
  :satisfies (parameterized-description
              :fit ?model ?structure
              ?fit-operation ?fit-relationship)
  :roles     ((?fit-relationship :result))
  :features  ((?structure ((:dataset-type relationship)
                           (:cardinality 2))))
  :body (:SEQUENCE
         (:SUBGOAL generate-fit
                   (generate-fit
                    ?structure ?fit-operation ?fit-relationship))
         (:SUBGOAL evaluate-fit
                   (explore-by
                    ?strategy ?activity ?model ?fit-relationship
                    ?deepening-result)
                   :class fit-evaluation-goal)))
```

The iteratively-fit-relationship plan is more involved than the fit-relationship plan but is similar in its use.

```
(define-plan-focus-point-type
    fit-relationship-focus-point
    iteratively-fit-relationship)

(define-plan iteratively-fit-relationship ()
  "Describe a relationship by iteratively
   fitting a function to it.  Evaluate the fit."
  :satisfies   (parameterized-description
                :iterative-fit ?model ?structure
                ?fit-operation ?fit-relationship)
  :features    ((?structure ((:dataset-type relationship)
                             (:cardinality 2))))
  :roles      ((?fit-relationship :result))
  :bindings    ((?residuals)
                (?residual-relationship)
                (?residual-fit-relationship)
                (?iteration-record)
                (?continue-iteration-p t)
                (?deepening-result))
  :body (:SEQUENCE
         (:SUBGOAL generate-fit
                   (generate-fit
```

```
          ?structure ?fit-operation ?fit-relationship))
(:SUBGOAL generate-iteration-record
          (generate-iteration-record
           ?fit-operation ?fit-relationship
           ?iteration-record))
(:WHILE (not (null ?continue-iteration-p))
        (:SEQUENCE
         (:SUBGOAL extract-residual-batch
                   (extract-residuals
                    ?fit-relationship :batch ?residuals))
         (:SUBGOAL compose-residual-relationship
                   (compose-residuals
                    ?fit-relationship :x
                    ?residuals ?residual-relationship))
         (:ACTION (recursively-generate-features
                   ?residual-relationship))
         (:SUBGOAL residual-fit
                   (generate-fit
                    ?residual-relationship ?fit-operation
                    ?residual-fit-relationship))
         (:SUBGOAL update-iteration-record
                   (update-iteration-record
                    ?structure ?fit-operation
                    ?residual-fit-relationship
                    ?fit-relationship ?iteration-record
                    ?continue-iteration-p))))
(:SUBGOAL evaluate-resistant-fit
          (explore-by
           ?strategy ?activity ?model
           ?fit-relationship ?deepening-result)
          :class fit-evaluation-goal)))
```

APPENDIX C

MENU COMMANDS

The user can explicitly direct AIDE to perform various tasks by selecting appropriate commands. The menu categories and the commands they contain are described below.

C.1 General Commands

These general-purpose commands are used to initiate and end an exploratory session.

- Load dataset: Interactively load a dataset from a file.
- Save dataset: Interactively save to a file one of the datasets AIDE has generated.
- Save session: Save a trace of the current session.
- Reset exploration: Discard all results and reset the status of the current session.
- Reset exploration (soft): Reset the status of the current session, but without discarding results.
- Toggle interaction mode: Change between USER and USER+AIDE modes.
- Quit: Exit AIDE.

C.2 Select Commands

These commands let the user select a data structure to explore.

- Select current data: Set context to exploration of the current structure.
- Select partition: Select one of the subsets of the initial dataset. Such subsets have been generated by AIDE or the user with partition commands.

- Select variable/relationship: Select a specific variable or relationship from the current partition.
- Invert selected relationship: If the relationship (x, y) is under consideration, view (y, x) instead.
- Data selection history: Explore a variable or relationship that was examined earlier.
- Model selection history: Explore a model that was generated earlier.

C.3 Partition Commands

These commands are associated with breaking the current variable or relationship into subsets.

- Partition relationship by discrete variable: Partition the current structure by an interactively selected discrete variable.
- Partition on clusters: Explore a partition of the current structure, based on an interactively selected clustering present in another variable or relationship in the dataset.
- Name clusters: Associate different names with different clusters.
- User defined partition: Interactively define a function to partition the current structure.

C.4 Transform Commands

These commands transform a variable or relationship by calling a function on each observation. Most are self-explanatory.

- Log 10.
- Natural log.
- Smooth: Smooth by 4253H.

- Difference: Generate a new variable or relationship whose elements are $a_{i+1} - a_i$.
- Power transformation: Generate a new variable or relationship whose values are a_c, where c is a user-supplied parameter.
- Interactive power transformation: Power transform a relationship through an interactive graphical interface.
- Add: Sum the variables in the current relationship; given (x1, x2, ..., xn), each observation in the resulting variable is $y_i = x1_i + x2_i + \ldots + xn_i$.
- Subtract: Subtract the variables in the current relationship; given (x1, x2, ..., xn), each observation in the resulting variable is $y_i = x1_i - x2_i - \ldots - xn_i$.
- Multiply: Take the product of the variables in the current relationship; given (x1, x2, ..., xn), each observation in the resulting variable is $y_i = x1_i * x2_i * \ldots * xn_i$.
- Divide: Divide the variables in the current relationship; given (x1, x2, ..., xn), each observation in the resulting variable is $y_i = x1_i / x2_i / \ldots / xn_i$.
- User defined transformation: Transform the current variable/relationship by an interactively defined Lisp function.

C.5 Describe Commands

Most of these items simply print out the relevant statistic. Statistics that are applicable to variables can also be applied to the component variables of a relationship.

- Data length: Display the number of rows (also known as cases or observations) in the current structure.
- Mean.
- Variance.
- Standard deviation.
- Minimum.

- Maximum.
- Range.
- Quantile: Display the quantiles of a variable, based on a user-supplied value.
- Median.
- Trimmed mean: Display a trimmed mean of a variable, based on a user-supplied proportion.
- Mode: Display the most commonly occurring value in a variable.
- Interquartile range.
- Covariance.
- Correlation.
- Partial correlation: Display the partial correlation between the variables in a relationship, with the variables to be held constant supplied by the user.
- Lisp function reduction: Display the result of calling an interactively defined function on the elements in the current variable or relationship.
- Correlation table: Display a table of pairwise correlations between variables in the current relationship.

C.6 Model Commands

These commands generate model-related data structures for exploration. They are currently limited to linear models.

- Simple regression: Given the currently selected structure (x, y), regress y on x.
- Resistant line: Given the currently selected structure (x, y), this command generates a resistant line for x and y.
- Residuals: This command generates the residuals from a linear fit, produced by either of the linear fit commands above.

- Leverage: This command generates a leverage relationship for a linear fit produced by the linear fit commands above.
- Multiple linear regression (current variables): Given the currently selected structure (x1, x2, ..., xn, y), regress y on x1 through xn.
- Multiple linear regression (any variables): Generate a regression based on a set of interactively selected variables.
- Linear causal model (current variables): Given the currently selected structure (x1, x2, ..., xn), generate a causal model of all variables, using a partial correlation heuristic for testing conditional independence.
- Linear causal model (any variables): Generate a causal model of interactively selected variables, using a partial correlation heuristic for testing conditional independence.
- Rank-correlation linear causal model (any variables): Generate a causal model of interactively selected variables, using a resistant rank partial correlation heuristic for testing conditional independence.

C.7 Tests Commands

These commands generate test results, which are presented and not explored further. Univariate tests act on individual variables. Multivariate t-tests treat relationships as (x1, ..., xn).

- Confidence interval (z statistic): Generate a confidence interval for the mean of the current variable, based on the normal distribution.
- Confidence interval (t statistic): Generate a confidence interval for the mean of the current variable, based on the t distribution.
- Confidence interval (proportion): Generate a confidence interval for the proportion of the variables in the current relationship.

- One sample t-test: Test whether the observations in the current variable were drawn from a population with a known mean.
- Two sample t-test: Test whether the observations in the variables of the current relationship were drawn from populations with the same mean.
- Matched pairs t-test: Test whether the paired differences between observations in the variables of the current relationship are different from zero.
- Chi square test: Test whether the variables in the discrete-valued relationship (x, y) are independent.
- G test: Test whether the variables in the discrete-valued relationship (x, y) are independent.
- Anova - one way: Test whether a discrete-valued variable x affects the values of a continuous variable y.
- Anova - two way: Test whether a set of discrete-valued variables (x1, ..., xn) affect the values of a continuous variable y.

C.8 Display Commands

These commands influence what gets displayed, but otherwise have no effect.

- Display dataset variables: Display a table of all dataset variables and their values.
- Set observation ID attribute: Select a variable whose values identify each row or observation. This can be helpful in matching outlying values in a scatter plot to unique identifiers of observations.
- Make subsidiary source display: Create a window that duplicates the source pane.
- Make subsidiary result display: Create a window that duplicates the result pane.

- Close subsidiary displays: Close all windows created by the above commands, leaving only the main AIDE frame.
- Clear display: Clear all AIDE frame panes.

REFERENCES

Abelson, Harold; Sussman, Gerald Jay; and Sussman, Julie 1985. *Structure and Interpretation of Computer Programs.* MIT Press.

Allen, James F. 1994. Mixed initiative planning: Position paper. [WWW document]. Presented at the ARPA/Rome Labs Planning Initiative Workshop. URL http://www.cs.rochester.edu/research/trains/mip/.

Anderson, Scott D.; Carlson, Adam; Westbrook, David L.; Hart, David M.; and Cohen, Paul R. 1993. CLASP/CLIP: Common Lisp Analytical Statistics Package/Common Lisp Instrumentation Package. Technical Report 93-55, University of Massachusetts at Amherst, Computer Science Department.

Biswas, Gautam; Weinberg, Jerry; Yang, Quin; and Koller, Glen R. 1991. Conceptual clustering and exploratory data analysis. In *Proceedings of the Eighth International Conference on Machine Learning.* 591–597.

Biswas, Gautam; Weinberg, Jerry; and Li, Cen 1994. ITERATE: A conceptual clustering method for knowledge discovery in databases. In Braunschweig, B. and Day, R., editors 1994, *Innovative Applications of Artificial Intelligence in the Oil and Gas Industry.* Editions Technip.

Brachman, Ronald J.; McGuinness, Deborah L.; Patel-Schneider, P. F.; Resnick, Lori Alperin; and Borgida, Alex 1990. Living with CLASSIC: When and how to use a KL-ONE-like language. In Sowa, J., editor 1990, *Formal Aspects of Semantic Networks.* Morgan Kaufmann Publishers, Inc.

Brachman, Ronald J.; Selfridge, Peter G.; Terveen, Loren G.; Altman, Boris; Borgida, Alex; Halper, Fern; Kirk, Thomas; Lazar, Alan; McGuinness, Deborah L.; and Resnick, Lori Alperin 1992. Knowledge representation support for data archaeology. In *Proceedings of the ISSM International Conference on Information and Knowledge Management (CIKM-92).* 457–464.

Carbonell, Jaime G.; Blythe, Jim; Etzioni, Oren; Gil, Yolanda; Joseph, Robert; Kahn, Dan; Knoblock, Craig; Minton, Steven; Perez, Alicia; Reilly, Scott; Veloso, Manuela; and Wang, Xuemei 1992. PRODIGY4.0: The manual and tutorial. School of Computer Science CMU-CS-92-150, Carnegie Mellon University.

Carver, Norman and Lesser, Victor 1993. A planner for the control of problem solving systems. *IEEE Transactions on Systems, Man, and Cybernetics, special issue on Planning, Scheduling, and Control* 23(6):1519–1536.

Carver, Norman and Lesser, Victor 1994. The evolution of blackboard control. *Expert Systems with Applications, special issue on the Blackboard Paradigm and its Applications* 7(1):1–30.

Chambers, John M. 1977. *Computational methods for data analysis.* John Wiley & Sons, Inc.

Chapman, David 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Cleveland, W. S.; Graedel, T. E.; Kleiner, B.; and Warner, J. L. 1974. Sunday and workday variations in photochemical air pollutants in New Jersey and New York. *Science* 186:1037–1038.

Cohen, Paul R.; Greenberg, Michael L.; Hart, David M.; and Howe, Adele E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3):32–48.

Cohen, Paul R.; Gregory, Dawn E.; Ballesteros, Lisa A.; and St. Amant, Robert 1996. Two algorithms for inducing structural equation models from data. In Fisher, Douglas and Lenz, Hans, editors 1996, *Learning from Data: AI and Statistics V.* Springer-Verlag.

Cohen, Paul R. 1995. *Empirical Methods for Artificial Intelligence.* MIT Press.

Currie, Ken and Tate, Austin 1991. O-Plan: The open planning architecture. *Artificial Intelligence* 52:49–86.

Daniel, Cuthbert and Wood, Fred S. 1980. *Fitting Equations to Data.* John Wiley & Sons, Inc.

Davis, Randall 1980. Meta-rules: Reasoning about control. *Artificial Intelligence* 15:179–222.

Diaconis, Persi 1985. Theories of data analysis: From magical thinking through classical statistics. In Hoaglin, David C.; Mosteller, Frederick; and Tukey, John W., editors 1985, *Exploring Data Tables, Trends, and Shapes.* John Wiley & Sons, Inc.

Duda, Richard O. and Hart, Peter E. 1973. *Pattern Recognition and Scene Analysis.* John Wiley & Sons, Inc.

Durfee, Edmund H. and Lesser, Victor R. 1986. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence.* MIT Press. 58–64.

Emerson, John D. and Hoaglin, David C. 1983. Resistant lines for y versus x. In Hoaglin, David C.; Mosteller, Frederick; and Tukey, John W., editors 1983, *Understanding Robust and Exploratory Data Analysis.* John Wiley & Sons, Inc.

Erol, Kutluhan; Hendler, James; and Nau, Dana S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.* 249–254.

Etzioni, Oren and Weld, Dan 1993. Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert* 8(4):44–49.

Everitt, Brian 1993. *Cluster Analysis.* John Wiley & Sons, Inc.

Falkenhainer, B. C. and Michalski, R. S. 1986. Integrating quantitative and qualitative discovery: the ABACUS system. *Machine Learning* 1(1):367–401.

Faraway, Julian J. 1992. On the cost of data analysis. *Journal of Computational and Graphical Statistics* 1(3):213–229.

Faraway, Julian J. 1994. Choice of order in regression strategy. In Cheeseman, P. and Oldford, R. W., editors 1994, *Building Models from Data: Artificial Intelligence and Statistics IV.* Springer-Verlag.

Fawcett, Tom E. 1993. Feature discovery for problem solving systems. Technical Report CMPSCI-93-49, University of Massachusetts, Amherst. PhD Thesis.

Fikes, R. E. and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

Fisher, Douglas and Langley, Pat 1986. Conceptual clustering and its relation to numerical taxonomy. In Gale, William, editor 1986, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company.

Fisher, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

Forester, Tom 1989. *Computers in the human context: information technology, productivity, and people.* MIT Press.

Fowlkes, Edward B.; Gnanadesikan, Ramanathan; and Kettenring, Jon R. 1987. Variable selection in clustering and other contexts. In Mallows, C. L., editor 1987, *Design, Data, and Analysis: by some friends of Cuthbert Daniel.* John Wiley & Sons, Inc.

Fox, John and Long, J. Scott 1990. *Modern Methods of Data Analysis.* Sage Publications.

Gale, W. A. 1986. REX review. In Gale, W. A., editor 1986, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company.

Gennari, John H.; Langley, Pat; and Fisher, Doug 1989. Models of incremental concept formation. *Artificial Intelligence* 40:11–62.

Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Science* 7:155–170.

Georgeff, Michael P. and Lansky, Amy L. 1986. Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation* 74(10):1383–1398.

Georgeff, Michael P. and Lansky, Amy L. 1987. Reactive reasoning and planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence.* MIT Press. 677–682.

Goldstein, J. and Roth, S.F. 1994. Using aggregation and dynamic queries for exploring large data sets. In *Proceedings of the CHI'94 Conference.* Association for Computing Machinery. 23–29.

Good, I. J. 1983. The philosophy of exploratory data analysis. *Philosophy of Science* 50:283–295.

Goodall, Colin 1983. Examining residuals. In Hoaglin, David C.; Mosteller, Frederick; and Tukey, John W., editors 1983, *Understanding Robust and Exploratory Data Analysis.* John Wiley & Sons, Inc.

Hammond, Kristian 1986. Case-based planning: An integrated theory of planning, learning and memory. Department of computer science technical report, Yale University, New Haven, CT. PhD Thesis.

Hand, D.J. 1986. Patterns in statistical strategy. In Gale, W.A., editor 1986, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company. 355–387.

Hoaglin, David C.; Mosteller, Frederick; and Tukey, John W. 1983. *Understanding Robust and Exploratory Data Analysis.* John Wiley & Sons, Inc.

Hoaglin, David C.; Mosteller, Frederick; and Tukey, John W. 1985. *Exploring Data Tables, Trends, and Shapes.* John Wiley & Sons, Inc.

Howe, Adele E. 1993. Accepting the inevitable: The role of failure recovery in the design of planners. Computer Science Department Technical Report 93-40, University of Massachusetts, Amherst, Massachusetts. PhD Thesis.

Huber, Peter J. 1986a. Data analysis implications for command language design. In Hopper, K. and Newman, I. A., editors 1986a, *Foundation for Human-Computer Communication.* Elsevier Science Publishers.

Huber, Peter J. 1986b. Environments for supporting statistical strategy. In Gale, W.A., editor 1986b, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company. 285–294.

Huber, Peter J. 1994. Languages for statistics and data analysis. In Dirschedl, Peter and Ostermann, Ruediger, editors 1994, *Computational Statistics.* Springer-Verlag.

Kellogg, C. and Livezey, B. 1992. Intelligent data exploration and analysis. In *Proceedings of the ISSM International Conference on Information and Knowledge Management (CIKM-92).* 257–264.

Kloesgen, Willi 1992. Problems of knowledge discovery in databases and their treatment in the statistics interpreter explora. *International Journal of Intelligent Systems* 7:649–673.

Kloesgen, Willi 1993. Some implementation aspects of a discovery system. In *Proceedings of the AAAI-93 Workshop in Knowledge Discovery in Databases.* 37–48.

Korf, Richard E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

Langley, Pat; Simon, Herbert A.; Bradshaw, Gary L.; and Zytkow, Jan M. 1987. *Scientific Discovery: Computational Explorations of the Creative Processes.* MIT Press.

Lansky, Amy L. and Philpot, Andrew G. 1993. AI-based planning for data analysis tasks. *IEEE Expert* 8(5).

Lesser, Victor R.; Nawab, S. Hamid; and Klassner, Frank I. 1995. IPUS: An architecture for the integrated processing and understanding of signals. *Artificial Intelligence* 77(1):129–171.

Lubinsky, David and Pregibon, Daryl 1988. Data analysis as search. *Journal of Econometrics* 38:247–268.

Mallows, C. L. 1973. Some comments on C_p. *Technometrics* 15:661–675.

Mallows, C. L. 1979. Robust methods–some examples of their use. *American Statistician* 33:179–184.

Mallows, C. L. 1983. Data description. In Box, G. E. P.; Leonard, T.; and Wu, C.-F., editors 1983, *Scientific Inference, Data Analysis, and Robustness.* Academic Press.

Matheus, Christopher J.; Chan, Philip K.; and Piatetsky-Shapiro, Gregory 1993. Systems for knowledge discovery in databases. *IEEE Transactions on Knowledge and Data Engineering* 5(6):903–913.

McAllester, D. and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence.* MIT Press. 634–639.

McDermott, Drew 1992. Transformational planning of reactive behavior. Department of Computer Science Technical Report 941, Yale University.

Michalski, R. S. and Stepp, R. E. 1983. Learning from observation: Conceptual clustering. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., editors 1983, *Machine Learning: An artificial intelligence approach.* Morgan Kaufmann Publishers, Inc.

Mosteller, Frederick and Tukey, John W. 1977. *Data Analysis and Regression.* Addison-Wesley Publishing Company.

Murphy, P.M. and Aha, D.W. 1993. UCI repository of machine learning databases [Machine-readable data repository]. Irvine, CA: University of California, Department of Information and Computer Science.

Nordhausen, Bernd and Langley, Pat 1990. A robust approach to numeric discovery. In *Proceedings of the Seventh International Conference on Machine Learning.* Morgan Kaufmann Publishers, Inc. 411–418.

Oates, Tim and Cohen, Paul R. 1996. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning.* Morgan Kaufmann Publishers, Inc. 1–10.

Oldford, R. Wayne and Peters, Stephen C. 1986. Implementation and study of statistical strategy. In Gale, W.A., editor 1986, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company. 335–349.

Pearl, J. and Verma, T. 1991. A theory of inferred causation. In *Proceedings of Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, Inc. 441–452.

Penberthy, J. and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, Inc. 103–114.

Pregibon, Daryl 1991. Incorporating statistical expertise into data analysis software. In *The Future of Statistical Software.* National Research Council, National Academy Press. 51–62.

Quinlan, J. R. 1993. *C4.5 : programs for machine learning.* Morgan Kaufmann Publishers, Inc.

Roth, S.F.; Kolojejchick, J.; Mattis, J.; and Goldstein, J. 1994. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the CHI'94 Conference.* Association for Computing Machinery. 112–117.

Rouse, William B.; Geddes, Norman D.; and Curry, Renwick E. 1987. An architecture for intelligent interfaces: Outline of an approach to supporting operators of complex systems. *Human-Computer Interaction* 3:87–122.

Russell, Stuart and Norvig, Peter 1995. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc.

Sall, John and Lehman, Ann 1996. *JMP Start Statistics: A Guide to Statistics and Data Analysis Using JMP and JMP IN Software.* Duxbury Press.

Schaffer, Cullen 1990. Domain-independent scientific function finding. Department of Computer Science Technical Report LCSR-TR-149, Rutgers University, New Brunswick, NJ. PhD Thesis.

Shneiderman, Ben 1992. *Designing the user interface: strategies for effective human-computer interaction.* Addison-Wesley Publishing Company.

Sibson, R. 1973. Slink: An optimally efficient algorithm for the single-link cluster method. *Computer Journal* 16(1):30–34.

Silverman, Barry G. 1992. Human-computer collaboration. *Human-Computer Interaction* 7:165–196.

Simoudis, Evangelos; Livezey, Brian; and Kerber, Randy 1994. Integrating inductive and deductive reasoning for database mining. In *Proceedings of the AAAI-94 Workshop in Knowledge Discovery in Databases.* 37–48.

Spirtes, Peter; Glymour, Clark; and Scheines, Richard 1993. *Causation, Prediction, and Search.* Springer-Verlag.

Stemple, David and Sheard, Tim 1989. Automatic verification of database transaction safety. *ACM Transactions on Database Systems* 14(3):322–368.

Tanur, Judith M.; Mosteller, Frederick; Kruskal, William H.; Link, Richard F.; Pieters, Richard S.; and Rising, Gerald R. 1972. *Statistics: A guide to the unknown.* Holden-Day.

Terveen, L. G. 1993. Intelligent systems as cooperative systems. *International Journal of Intelligent Systems* 3(2–4):217–250.

Thisted, Ronald A. 1986. Representing statistical knowledge. In Gale, W.A., editor 1986, *Artificial Intelligence and Statistics I.* Addison-Wesley Publishing Company. 389–399.

Tufte, Edward R. 1983. *The Visual Display of Quantitative Information.* Graphics Press.

Tukey, P. A. and Tukey, J. W. 1981. Graphical display of data sets in 3 or more dimensions. In Barnett, Vic, editor 1981, *Interpreting Multivariate Data.* John Wiley & Sons, Inc.

Tukey, John W. 1977. *Exploratory Data Analysis.* Addison-Wesley Publishing Company.

Tukey, John W. 1982. Introduction to styles of data analysis techniques. In Launer, Robert L. and Siegel, Andrew F., editors 1982, *Modern Data Analysis.* Academic Press.

Tversky, A. 1977. Features of similarity. *Psychological Review* 84:327–352.

Velleman, Paul F. and Hoaglin, David C. 1981. *Applications, Basics, and Computing of Exploratory Data Analysis.* Duxbury Press.

Veloso, Manuela M. and Carbonell, Jaime G. 1993. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning* 10:249–278.

Weisberg, Sanford 1985. *Applied Linear Regression.* John Wiley & Sons, Inc.

Wickens, Thomas D. 1989. *Multiway contingency tables analysis for the social sciences.* Lawrence Erlbaum Associates.

Wilensky, Robert 1981. Meta-planning: representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science* 5:197–233.

Wilkins, David 1988. *Practical Planning.* Morgan Kaufmann Publishers, Inc.

Zytkow, Jan M. and Zembowicz, Robert 1993. Database exploration in search of regularities. *Journal of Intelligent Information Systems* 2:39–81.

Zytkow, Jan M. 1987. Combining many searches in the Fahrenheit discovery system. In *Proceedings of the Fourth International Conference on Machine Learning.* Morgan Kaufmann Publishers, Inc. 281–287.