A Planner for Exploratory Data Analysis

Robert St. Amant and Paul R. Cohen
Computer Science Department
University of Massachusetts
Amherst, MA 01003-4610
stamant@cs.umass.edu, cohen@cs.umass.edu

Abstract

Statistical exploratory data analysis (EDA) poses a difficult search problem. The EDA process lends itself however to a planning formulation. We have built a system, called AIDE, to help users explore data. AIDE relies on partial hierarchical planning, a form of planning appropriate for tasks in complex, uncertain environments. Our description of the EDA task and the AIDE system provides a case study of the application of planning to a novel domain.

Exploring data

Data exploration plays a central role in empirical scientific research. Sometimes we can build a model of complex phenomena based on theory alone; often, however, we need to explore the data. We need to identify suggestive features of the data, interpret the patterns these features indicate, and generate hypotheses to explain the patterns. Successive steps through the process lead us gradually to a better understanding of underlying structure in the data. Exploratory data analysis (EDA) gives us a powerful set of operations for this process: we fit linear and higher-order functions to relationships; we compose and transform variables with arithmetic functions; we separate relationships into partitions and clusters; we extract features through statistical summaries (Tukey 1977). Through the selective and often intuitive application of these operations we gradually build a description of a dataset.

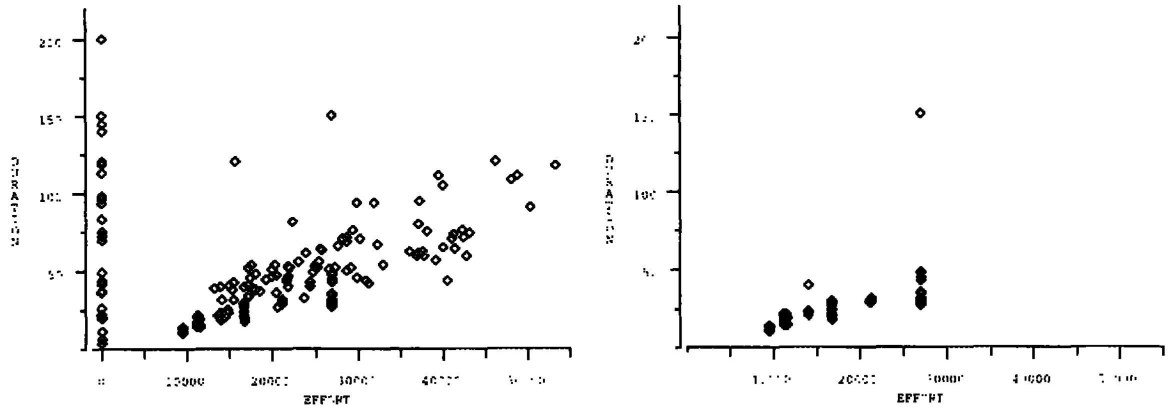
Exploration poses a difficult search problem. The flexibility of exploratory operators gives a large branching factor and an unbounded search space. If an exploratory analysis were driven purely by successive features discovered in data, the task would be impossible: Is a partitioning or a functional transformation appropriate? With what parameters? When should one stop? Though manageable in human hands, exploration is a difficult and painstaking task.

We have designed and implemented an Assistant for Intelligent Data Exploration, AIDE, to help users carry out their exploration (St. Amant & Cohen 1995). In AIDE, data-directed mechanisms extract simple observations and suggestive indications from the data. EDA operations then act in a goal-directed fashion to generate more extensive descriptions of the data. The system is mixed-initiative, autonomously pursuing its own goals while still allowing the user to guide or override its decisions. Our description of the planner and its task provides a case study of how domain characteristics can influence planner design.

In the remainder of the paper we discuss an EDA example, describing the types of operations and results involved in the process. We then discuss the planner and its plan representation in some detail. Returning to the EDA example, we show how it is solved by AIDE acting in concert with a user. We conclude by showing where AIDE fits in the context of approaches to planning.

An EDA example

We can best illustrate the EDA process with an example, 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 we collected forty variables 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 a comparison of the time it takes the planner to put out a fire (Duration) and the amount of fireline built during the trial (Effort). Figure 1a shows a scatter plot of these two variables.



(a) All data (b) Clusters

Figure 1: Patterns in Planner Effort and Trial Duration

We begin by observing that the relationship can be partitioned into two parts: a vertical partition at zero on the Duration axis and a separate, approximately linear partition. We call this an *indication*, a suggestive characteristic of the data. Examining other variables, we find that the vertical partition corresponds to trials in which the outcome was `Failure`. We note that the correlation is positive in the `Success` partition, as expected, but that there are two outliers from the general pattern.

We can be more precise about the "approximately linear" pattern in the `Success` partition: we can fit a regression line to the data. We then examine the residuals of the fit the degree to which the data are *not* explained by the description— by subtracting the actual value of Duration, for each value of Effort, from the value predicted by the regression line. We see no indications of further structure, such as curvature, that would render our description incorrect, and thus we tentatively accept the linear description.

Looking again at the `Success` partition, we notice small, vertical clusters in the lower range of Effort. In a histogram, or a kernel density estimate, these would appear as peaks. The clustered points are isolated in Figure 1b. We can describe the behavior of the clusters in terms of their central location, by reducing each cluster to its median Effort and Duration value. These medians are also linear, with approximately the same slope as the line fitting the entire partition.

We then try to explain why some observations fall into clusters while others do not. We find that the clustered data correspond to trials in which the planner did not need to replan: that is, observations fall into clusters only when #Replans = 0. If we associate each cluster with a unique identifier, we find that together the discrete variables WindSpeed and PlanType predict cluster membership almost perfectly. It further becomes clear that for the clustered data there is an interaction between the two predictive variables in their effect on Duration.

This brief account gives the flavor of EDA. A more detailed account given in *Empirical Methods in Artificial Intelligence* (Cohen 1995). The remainder of this paper describes the planner that lets AIDE, in cooperation with a user, generate these kinds of results.

Abstractions in exploration

In *Exploratory Data Analysis*, John Tukey describes EDA in this way:

> 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 through abstraction and description by hierarchical problem decomposition.

Abstraction is ubiquitous in exploration. Fitting a straight line to a relationship involves deciding that variance around the line, evidence of curvature, outlying values, and so forth may be ignored at an appropriate level of abstraction. One fits a simple description, a line, before attempting to describe the residuals, i.e., those data that don't fit the abstraction well. The effect is of moving from higher to lower levels of abstraction. A more subtle example can be seen in the Phoenix analysis. To describe the behavior of the vertical clusters in Effort and Duration, we summarize each cluster in terms of its central location. In other words, we deal with a simplification of each cluster, in which spread around the central location has been abstracted away.

Hierarchical problem decomposition plays a large part in exploration as well. The Phoenix example gives a good illustration: we begin by fitting a partition to the relationship, and then pursue the description of each component independently. Much of exploration can be viewed as the incremental decomposition of data into simple descriptions, which are then combined into a more comprehensive result.

Exploratory procedures furthermore impose top-down structure on the exploration process. In other words, when we execute an exploratory operation we generally have a good notion of which operation, of many possible, to execute next. Common procedures often fall into a few basic families that process data in similar ways. It is easy to see, for example, 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 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.

Knowledge of abstraction, problem decomposition, and common combinations of operations lets us restructure the EDA search space, to make it more manageable. These elements identify exploration as a planning problem (Korf 1987). Still, there are many different approaches to planning. Other characteristics of the domain help us refine our understanding of the type of planning involved.

Exploratory procedures require control structures more complex than simple sequences of operations. It is hard to see, for example, how one can iteratively improve a resistant fit or search through a space of model extensions given only the ability to chain together single operations. Many procedures are more naturally formulated in terms of tests of generated values, iteration, recursion, and other forms of control.

Exploration is opportunistic. Though procedures often specify a course of action, there can be a great deal of uncertainty in carrying out the details. Each operation is simultaneously an effective action and an information-gathering action. In the Phoenix example we could not have predicted that there would be vertical clusters in the relationship. Once we noticed the clusters, we were able to deal with them by reducing them to their medians and trying to describe the result. We could not have predicted that these points would be approximately linear, but this new information let us extend the exploration further. At each point during 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.

Finally, the results of an exploratory session are not simply the p-values, tables, graphs, and so forth that have been computed. Exploration is constructive, in that the interpretation of these individual 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 the knowledge that some operation has been applied and has failed can influence our interpretation of a related result. The result of an exploration, then, must include an annotated trace of the process itself.

The planner

A form of reactive planning called partial hierarchical planning (Georgeff & Lansky 1986) turns out to be a good match for the task. Systems that use the approach include PRS (Georgeff & Lansky 1987), the Phoenix planner (Cohen *et al.* 1989) and the RESUN system (Carver & Lesser 1993). Our design of the AIDE planner is largely based on experience with Phoenix and RESUN.

The AIDE planner operates by manipulating a stack of *control units*. The planner is essentially a high-level language interpreter, in which the active stack stores the current execution context. The planner executes the control unit at the top of the planning stack, by calling its `execute` method. If this generates a new control unit, 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 its status changes to `:succeeded` or `:failed`, then the control unit is not executed, but rather popped off the stack, its `complete` method being called at that point. The behavior of a control unit thus depends on the specialized definition of its execution and completion methods.

In this simple representation a variety of control structures can be defined. Consider a `:sequence` control unit, which executes a sequence of subordinate control units in order. A `:sequence` unit maintains an internal list of subordinate units. Each call to its execution method pops off and returns the next remaining element on the list. When the list is exhausted, or a subordinate fails, the `:sequence` unit's completion status is set appropriately and its completion method is called.

```
(define-plan explore-by-incremental-modeling ()
  :satisfies   (explore-by :modeling ?model-type ?structure ?model)
  :constraints ((?structure ((:dataset-type dataset))))
  :body        (:SEQUENCE
                (:WHEN (null ?model)
                  (:SUBGOAL generate (generate-initial-model ?structure ?model-type ?model)))
               (:SUBGOAL elaborate (elaborate-model ?model-type ?activity ?structure ?model))))

(define-action generate-initial-generic-model
  :satisfies (generate-initial-model ?description ?structure :generic ?model)
  :action (values t (return-bindings ?model (make-generic-model. . .))))
```

Figure 2: Plan and action definitions

Other control constructs can be defined similarly for conditionalization, iteration, and more specialized processing. Goals, plans, and actions are also specialized forms of control units.

A plan has a name, a specification of a goal that the plan can potentially satisfy, constraints on its bindings, 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. An action is similar to a plan, except that its body contains arbitrary code, rather than a control schema. The plan in Figure 2 is instantiated in the exploration of a dataset. It generates an initial model of an appropriate type and then establishes the goal of elaborating the model. With the plans in the AIDE library, elaboration will involve incrementally adding relationships to the model. One of the plans that matches the `elaborate` subgoal recursively establishes an identical subgoal, with `?model` bound to the incrementally extended model.

A plan instance, as a type of control unit, executes by instantiating the control units represented in its body. On completion a plan instance sends to its parent goal a completion status, `:succeeded` or `:failed`, along with a set of variable bindings. An action instance generates no new control units in its execution, but simply returns a completion status and a set of bindings for its matching goal. A goal instance executes by generating a new plan instance searching through the plan library and finding a matching (unifying) plan.

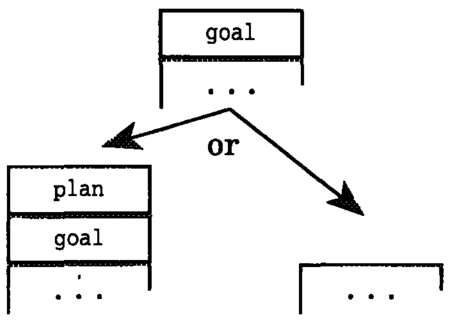
We must complicate this account somewhat. There are often several plans that satisfy a given goal, and sometimes an unlimited number of possible bindings for its plan variables. To manage these situations we have three mechanisms: evaluation rules, focus points, and meta-level plans. Selection of plans is similar to the selection of plan variable bindings, so we will only discuss plan selection.

When more than one plan in the plan library matches a goal, AIDE must decide which to instantiate. *Control rules* inform its decision. There are three steps involved in executing a plan to satisfy a goal: matching, activation, and preference. The matching step, already described, 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 a curvature indication. The preference step involves running another set of rules that impose an ordering on the active plans. In the presence of the curvature indication, the `power-transform` plan is preferred to the `linear-regression` plan.

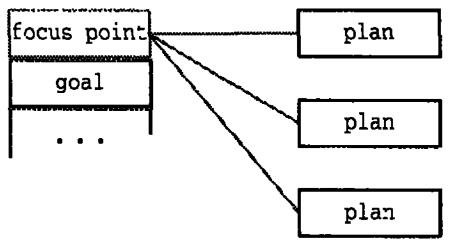
Candidate plans are maintained by *focus points*. A focus point manages branch points in the planning stack. Suppose that a goal instance is the top entry on the stack. If only a single plan matches this goal, then it is pushed onto the stack directly, as described earlier (shown here in Figure 3a.) If more than one plan matches the goal, a *plan focus point* is generated and pushed onto the stack. The focus point manages a set of newly created execution stacks, each rooted at a different instance of each plan that matches the goal. This is shown in Figure 3b. The planning process continues when one of these new stacks is selected, based on information generated by the evaluation rules, and the stack processed. When any stack becomes exhausted, the focus point can either select another of the stacks to proceed, or can return to the stack that originally generated the focus point.

For example, in the Phoenix data we saw that one subset of Duration and Effort was approximately linear, but with outliers in the relationship. We have several options: we can fit a regression line directly to the data; we can remove the outliers and then fit the line; we can let the outliers remain and fit a resistant line; we can fit a smooth to the data. Each of these options is implemented by a different plan. Each plan is instantiated to provide the root of a new execution stack, and the set of new stacks is then associated with the plan focus point. One stack is selected to continue the exploration.

As plans progress, a network of focus points is



(a) The plan stack



(b) A focus point

Figure 3: The planning process

generated. Only one is active at any time. Determining which focus point should be active at any given time is the responsibility of *meta-level plans*. A meta-level planner, identical in design to the base-level planner, handles focus point selection. The meta-level behavior of the system is similar to what would be provided by rule-based activation of plans; however, incorporating a planner at the meta-level lets us maintain the nessary execution context to switch dynamically between plans in progress.

In the Phoenix data, a vertical clustering indication in the relationship (Effort, Duration) activates several distinct plans. One plan searches for a way to distinguish between clustered and non-clustered points by using other variables in the dataset. Another plan tries to predict which cluster each data point belongs to. Other plans search for similar clustering patterns in other variables and relationships. Each of these plans leads to further exploration. Because the default behavior of focus points causes the network to expand in depth-first fashion, the exploration of just the first plan at this decision point could potentially go on indefinitely. A meta-level plan, however, can cause the planner to switch between active, partially-expanded plans. The plan expansion often leads to actions that generate new information, which can be relevant to the planner's behavior. In other words, meta-level plans in this and comparable situations capture a general huristic: When more than one plan exists to extend or deepen a description (e.g., a clustering or a regression) and these plans compute intermediate results that may provide useful indications, then alternate between partial expansions of the plans. In this case, the final decision about which plan is most appropriate is usually made by the user.

To summarize briefly, evaluation rules decide which plans are relevant in trying to satisfy a goal. A focus point selects one plan from those possibilities and executes it; the focus point may switch dynamically between possible plans as knowledge about the problem changes. Meta plans decide which focus point is currently active, i.e., which goal the system should be trying to satisfy at the current time. These mechanisms give us local and non-local control over the planning process.

User interaction

The correspondence between planning and EDA lets us take two different views of the interaction between AIDE and the user. We can say that AIDE is in the business of building and executing plans, in an environment that lets the user review and modify the system's plan selection and variable binding decisions. We can just as well say that the system is a statistical computing environment in which a user carries out EDA procedures; AIDE's job is to follow the process the best it can, making suggestions and supplying relevant information at each decision point. To show how this works, we return to the Phoenix example. The following dialog represents the information exchanged between AIDE and the user, though the interaction is actually through menu selections, user gestures, and the usual interactions common to statistical user interfaces. Some of the points are marked with a number in parentheses for later discussion.

USER *(1)*: Select relationship (Effort, Duration).
AIDE: (Effort, Duration) indications: high correlation ($r = 0.93$); outliers in Effort; outliers in Duration; clustering. Suggested courses of action are (a) generating a linear fit and (b) 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 *(2)*: (Generates linear fit.) A three-group resistant line fits the data with a slope of 0.0017

and intercept of −0.96. (Presents graphical display of linear fit.) Indications: outliers in residuals; clustering.

USER (3): Show recent decisions.

AIDE: In reverse order: (a) With the goal of fitting a line, I decided without consultation on a resistant fit. The alternative is a regression fit. (b) With the goal of exploring (Effort, Duration), you accepted the linear fit suggestion. The alternative is to explore clusters...

USER: Back up a step.

AIDE: With the goal of fitting a line to (Effort, Duration), the possibilities are a resistant fit, which is in progress, or a regression fit, which has not started.

USER: Execute the regression fit.

AIDE: (Generates regression fit.) The regression line has a slope of 0.0019 and an intercept of −2.73; $r^2 = 0.57$. (Presents graphical display of linear fit.) Indications: outliers in the residuals: 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 (Effort, Duration) is linear, despite indications of outliers and clustering. I prefer the regression fit.

AIDE: (Accepts note.)

USER: Okay.

AIDE: (Considers linear fit phase to be complete.) The possibilities remaining to explore (Effort, Duration) are (a) the linear fit, which is in progress, and (b) cluster exploration, which has not started.

USER: Okay.

AIDE: (Begins cluster exploration.) Here are the clustering possibilities for (Effort, Duration). (Displays identifiers for potential clustering criteria: single-linkage clusters in Effort, regions of constant values in Effort, etc. Also displays similarity between resulting clusters.)

...

A single step in this exchange often represents a great deal of underlying processing. The steps marked with numbers are discussed in more detail below:

1. When the user loads a dataset, AIDE's top level plan is activated, which incrementally explores the variables and relationships of the dataset. Because any variable or relationship might be relevant, a focus point is created to manage the possibilities. Evaluation rules rank the possibilities by their features and indications, computed dynamically. When the evaluation is complete, AIDE presents a data selection menu to the user, along with its own preference and supporting indications. AIDE thus generates a great deal of information to support its internal decisions before interacting with the user. In this example the user overrides AIDE's preference by selecting (Effort, Duration) for exploration.
2. Here the user accepts one of AIDE's suggestions, that of fitting a line. The user could potentially have chosen a specific linear fitting procedure (through a menu selection) but instead lets AIDE maintain control over the exploration. AIDE makes an internal decision for a resistant line, generates the fit, and elaborates the description by generating and examining the residuals of the fit.
3. When the user asks for a listing of recent decisions, AIDE displays all focus points leading to the current decision point. Some of these may have been handled internally, while others may have been directed by the user. The list of focus points can be read directly from the plan execution network. The user can select any of these decisions, to be reconsidered and perhaps changed, as happens in the next few steps.

The network of focus points is central to AIDE's processing. It lets the user "navigate" through the exploration space at the strategic decision level, rather than the level of primitive EDA operations. Further, the network gives AIDE a way of interpreting user actions in context, and of following along when the user takes control of the exploration.

Related Work in Planning

Partial hierarchical planning was introduced by Georgeff and Lansky (Georgeff & Lansky 1986). Several characteristics distinguish it from classical planning. In the classical formulation a plan is a partially-ordered sequence of actions, often with annotated links between actions. In partial hierarchical planners, a plan is a procedural specification of a set of subgoals to be achieved. A plan may specify that subgoals must be satisfied sequentially, or conditionally on some test, or iteratively. Control constructs may also provide for parallel satisfaction of subgoals, mapping over lists of subgoals, recursion, and domain-specific processing.

A partial hierarchical planner executes a plan before it is completely elaborated. In a sense, these planners do not generate plans at all, but simply execute them. This behavior has advantages over 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. A disadvantage is the uncertainty about whether a given partial plan will succeed.

As with case-based planning, plan definitions are stored in a library. AIDE's definitions are not the fully-elaborated sequences of actions that are usually stored in a case library, however, but are partial specifications as described above. AIDE constructs plans by searching through the library, its set of partial solutions, for appropriate matches to established goals.

This lack of emphasis on constructing plans from scratch is balanced by a greater concentration on the meta-level problem of which plan to invoke when several match the current situation. Meta-level processing can be handled in different ways. PRS uses meta-level "knowledge areas" that function something like blackboard knowledge sources for control (Georgeff & Lansky 1987). The Phoenix planner maintains a time line of subgoals and pending plans, and gives each plan a degree of meta-level control over the actions remaining to be executed. The RESUN planner establishes focus points during its plan expansion to allow suspension and resumption of in-progress plans, as a way of focusing attention in its search (Carver & Lesser 1993). These mechanisms let the planners behave opportunistically.

Reactive planning techniques provide a good match for the EDA problem. Though there are no hard time constraints on the process, the space of exploration is highly dynamic, in that each action can provide potentially significant information. All the aspects of partial hierarchical planning mentioned above contribute to solving the problem.

AIDE is also an example of a mixed-initiative planning system. By making an analogy to dialog behavior, James Allen has identified three distinguishing characteristics of mixed-initiative planning: flexible, opportunistic control of initiative: the ability to change focus of attention: mechanisms for maintaining shared, implicit knowledge (Allen 1994). 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.

AIDE's control of initiative changes with context. That is, whether a decision is presented to the user or is settled internally depends on situation-dependent factors. For example, if AIDE determines that only one plan is able to satisfy a given goal (i.e., all others rendered inactive by evaluation rules), then this decision point will not be presented to the user. An exception is made in the case where a plan is being selected for the initial exploration of a variable or relationship. Choosing an initial plan for a relationship is often a more important decision than deciding how to bind its plan variables or how to satisfy its subgoals; thus the decision about how to proceed from the new point is presented to the user even if only one course of action seems appropriate.

AIDE changes its focus of attention to follow the user. In the processing for the Phoenix analysis, for example, AIDE interprets the selection of a new relationship as a shift of focus from exploration of its current relationship to a new point in the search space. 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 OK 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.

AIDE also provides ways for the user to follow its planning process. The user can view the data under consideration, results constructed, commands carried out, and planning structures leading to the current point. These views also act as navigation mechanisms, giving the user more explicit control over AIDE's focus of attention. AIDE does not ask for clarification of user actions, however, which could clearly be beneficial in some situations.

The mixed-initiative approach has become interesting to researchers in both planning and statistical expert systems. An extension of PRODIGY, for example, provides autonomous planning with consultation (Stone & Veloso 1995), by letting the user review each decision the planner makes. The early regression expert system, REX, gave users similar abilities when reaching difficult decision points in its analysis (Gale 1986). TESS, a sophisticated interactive system for data analysis, took the approach further by letting users opportunistically modify its search strategies and decisions (Lubinsky & Pregibon 1988). AIDE takes steps toward an even fuller cooperation between user and machine.

In summary, we have described the task of EDA and how it can (and should) be cast as a planning problem. We have presented the AIDE planner as part of a solution to the problem. We have described a representative example in the domain of EDA, and shown how AIDE solves the problem. A series of experiments is currently in progress to evaluate the performance of users working with and without assistance from AIDE.

Acknowledgments

Thanks to Tim Oates for useful discussion and comments about this work. This research is supported by ARPA/Rome Laboratory under contract #F30602-93-0100, and by the Dept. of the Army, Army Research Office under contract #DAAH04-95-1-0466. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon.

References

Allen, J. F. 1994. Mixed initiative planning: Position paper. Presented at the ARPA/Rome Labs Planning Initiative Workshop. See URL http://www.cs.rochester.edu/research/trains/.

Carver, N., and Lesser, V. 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).

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

Cohen, P. R. 1995. *Empirical Methods in Artificial Intelligence*. MIT Press.

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

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

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 677–682. American Association for Artificial Intelligence.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

Lubinsky, D., and Pregibon, D. 1988. Data analysis as search. *Journal of Econometrics* 38:247–268.

St. Amant, R., and Cohen, P. R. 1995. Control representation in an EDA assistant. In Fisher, D., and Lenz, H., eds., *Learning from Data: AI and Statistics V*. Springer. To appear.

Stone, P., and Veloso, M. 1995. User-guided interleaving of planning and execution. In *European Workshop on Planning*.

Tukey, J. W. 1977. *Exploratory Data Analysis*. Addison-Wesley.