A Planner for Exploratory Data Analysis

Robert St. Amant and Paul R. Cohen
Computer Science Dept., LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
stamant@cs.umass.edu, cohen@cs.umass.edu
(413) 545-3616, fax (413) 545-1249

Abstract

Statistical exploratory data analysis (EDA) poses a difficult search problem. However, the EDA process lends itself 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 successful application of planning to a novel domain.

Keywords: reactive planning, partial hierarchical planning, statistics, exploratory data analysis

1 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 [11, 6]. Exploratory data analysis (EDA) [16] 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. Through the selective and often intuitive application of these operations we gradually build a description of a dataset.

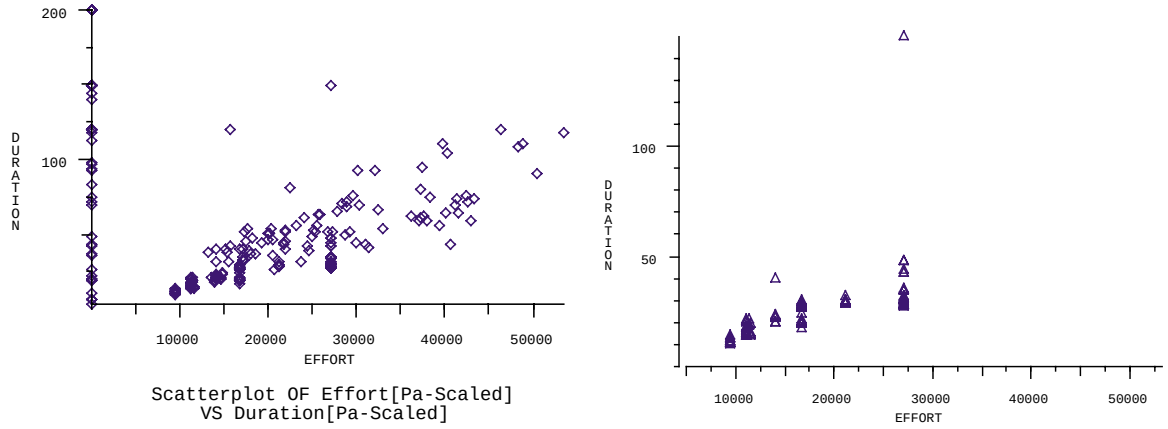
Viewed as search, exploration is a difficult problem. The flexibility of exploratory operators gives a large branching factor and an unbounded search space. If an exploratory analysis were driven purely by successive features discovered in data, the task would be impossible: Is a partitioning or a functional 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 [15]. 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 discuss the planner and its plan representation in some detail. We then return to the EDA example, to 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.

2 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 [3]. 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 2(a) shows a scatter plot of these



(a) Planner Effort vs Trial Duration (b) Effort Clusters

two variables.

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. Cross-tabulation with other variables shows that the vertical partition corresponds to trials in which the outcome was **`Failure`**. Concentrating on the **`Success`** partition, we note that the correlation is positive, as expected, but that there are two outliers from the general pattern. We put these two points aside, to examine them more closely later.

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 small peaks. The clustered points are isolated in Figure 2(b). 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 create a binary variable that encodes this difference. By cross-tabulating this variable with other discrete experiment variables, we find that the clustered data correspond to trials in which the planner did not need to replan. This is shown in Table 1: observations fall into clusters only when #Replans = 0.

If we further associate each cluster with a unique identifier, we find that WindSpeed and PlanType predict cluster membership. We can summarize this relationship as shown in Table 2, and extract a yet more refined description of the behavior of the planner as wind speed and plan type change.

This brief account gives the flavor of EDA. Note our use of *indications* in the analysis. Indications are suggestive characteristics of the data, most often involving evaluation of a statistic or descriptive structure [10]. The gap in the distribution of Effort indicates

# Replans =	0	1	2	3	...
clustered Effort	113	0	0	0	...
non-clustered Effort	49	95	45	14	...

Table 1: Clustered and non-clustered Effort for successful trials

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

Table 2: Median Duration by WindSpeed and PlanType for clustered data

that a partition is an appropriate description; the clusters in Effort 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. The process is constructive and opportunistic. A more detailed account of the application of these procedures to the Phoenix data is given in *Empirical Methods in Artificial Intelligence* [2].

3 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 [16, 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. An 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 often 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. Further, 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 [15]. 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 [7]. There are many different approaches to planning, however. 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.

Exploratory procedures are opportunistic. Though they often specify in general terms how one proceeds, 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.

These characteristics lead us to cast exploration as a planning problem, though not in the classical STRIPS formulation. A planner for exploration should be able to represent goals, to encode potentially complex control procedures, to adapt opportunistically to new findings, and to generate a structured set of justifications for its actions and support for its results.

4 The Planner

A form of reactive planning called partial hierarchical planning [4] turns out to be a good match for the task. Systems that use the approach include PRS [5], the Phoenix planner [3], the RESUN system [1], and to some extent languages for reactive control such as PROPEL [8]. Our design of the AIDE planner is largely based on experience with Phoenix and RESUN.

In abstract terms, the AIDE planner does little more than manipulate 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 top level planning loop is simple enough to present in pseudocode, as shown below:

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

In words, the planner executes the control unit at the top of the planning stack. If this generates a new control unit, then it is pushed onto the stack to be executed in turn. This process continues as long as the top 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. The two functions `execute-stack-element` and `complete-stack-element` allow the behavior of control units to be appropriately specialized.

4.1 Plan Representation

We turn now to the representation of plans. An example of a plan specification is given below. 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 specification is similar to a plan specification, except that its body contains arbitrary code, rather than a control schema.

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

The plan above is instantiated in the exploration of a dataset. It generates an initial model of the dataset (unless one is already available) of an appropriate type. It then establishes the goal of elaborating the model. With the plans in the AIDE library, elaboration will involve adding relationships, one at a time, to the model. One of the plans that matches the `elaborate-model` subgoal recursively establishes an identical subgoal, with `?model` bound to the incrementally extended model.

Plan, goal, and action instances are examples of control units. A control unit is specialized in two ways, through its execution and its completion. A goal instance executes by generating a new plan instance—searching through the plan library and finding a matching (unifying) plan. When a goal instance completes, it sends to its parent plan a completion status, `:succeeded` or `:failed`, along with a set of variable bindings. A plan instance executes by instantiating the control units represented in its body. It completes by informing its matching goal of its completion status. 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.

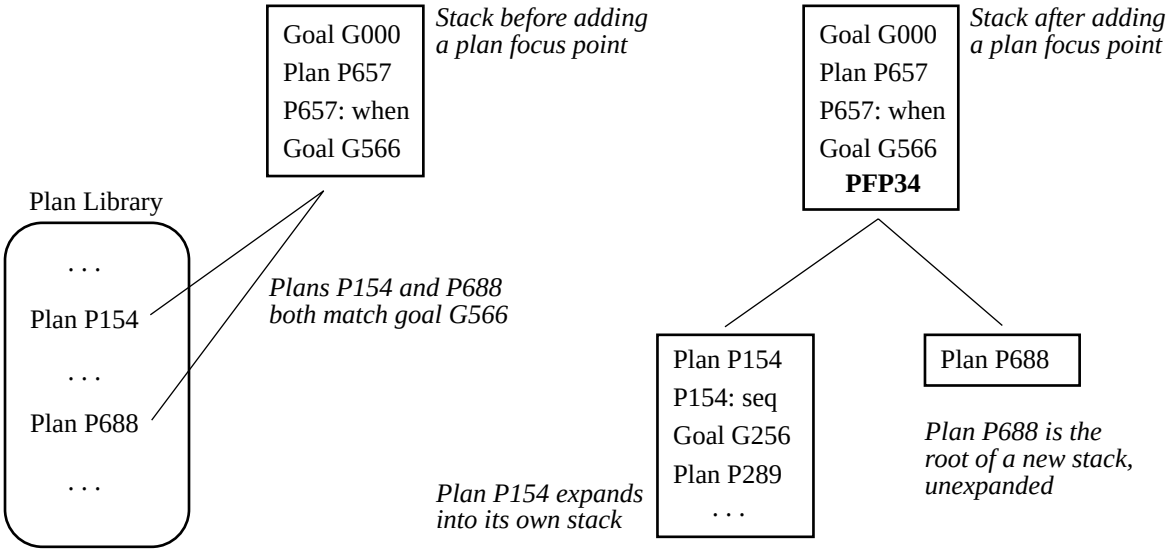
Schema control constructs, such as `:sequence`, `:when`, and `:while`, are control units as well. A `:sequence` construct generates a series of new control units, one at a time, each time it is executed. If any of these control units fail, then it fails as well; otherwise it succeeds. A `:when` construct generates a new control unit, its body, only if its executable test form returns non-nil. A `:while` form is similar to a `:when` form, but iterates its body as long as its test form returns non-nil. Other control constructs are defined for more specialized processing.

4.2 Branches in the Search Space

This account would be adequate only if the plan library contained a single matching plan for each goal and a single value for each established variable binding. There are often several plans that may satisfy a given goal, and sometimes an essentially 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.

When more than one plan in the plan library matches a goal, AIDE must decide which to instantiate. *Evaluation 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 curvature indications. The preference step involves running another set of rules that apply preferences to the active plans. In the presence of the curvature indication, the **`power-transform`** plan is preferred to the **`linear-regression`** plan. Evaluation of the bindings of plan variables follows a similar procedure.

Candidate plans and plan variable bindings 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. If more than one plan matches the goal, a *plan focus point* is generated and pushed onto the stack. The figure below illustrates the process. The focus point manages a set of newly created execution stacks, each rooted at a different instance of each plan that matches the goal. 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 locally-weighted, smoothed regression curve 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. The set of new stacks is then associated with the plan focus point. One stack is selected to continue the exploration.

Another type of focus point is a *variable focus point.* When an action satisfies a goal, it

returns a set of data structures that are bound to the plan variables specified in the goal. An action may return multiple bindings for a single variable, indicating that there are competing alternative ways to satisfy the goal. When this happens, a new branch point is generated, managed by a variable focus point. Its behavior is similar to that of a plan focus point. For example, in the Phoenix data we saw that some of the points fell into small clusters. Depending on our clustering criterion, we might treat these points as a set of two clusters, or nine, or fourteen, or some other number. Each of these possibilities is instantiated as a different possible binding of a `?description` plan variable, the different bindings maintained by the variable focus point.

As plans progress, a tree of focus points is generated. Only a single 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 switch dynamically between plans in progress. This gives us a finer level of control over the system's focus of attention than can be provided by a rule-based approach.

5 The Example Revisited

AIDE can run without assistance from the user. Its rules let it evaluate candidate data structures to explore; its library provides the plans that perform the exploration. Not surprisingly, however, AIDE's lack of contextual knowledge will take the exploration in directions an informed human analyst would not need to go. In the Phoenix experiment, for example, we were interested in a specific issue: ability of the planner to solve problems of varying difficulty under different degrees of time pressure. We collected a wide range of information to test our hypotheses, as well as subsidiary information for metering and other purposes. In its exploration of the Phoenix dataset AIDE would spend time exploring relationships that are completely uninteresting to us, or establishing descriptions of factors we are already familiar with. Human knowledge of context can be essential in focusing on interesting or significant areas in the exploration search space [9, 12]. To take advantage of this knowledge, AIDE pursues a mixed-initiative approach to exploration.

Exploration is mixed-initiative in the following sense. AIDE explores a dataset by elaborating and executing a hierarchy of plans. Branch points in the hierarchy are associated with explicit focus points. The user guides exploration by changing the ordering of candidate plans or variable bindings at a single focus point, or by selecting a different focus point as the currently active one. This guidance includes selecting appropriate dataset variables and subsets to be explored as well as applying any available statistical manipulations to the selected data. The arrangement gives the user most of the flexibility of the usual statistical interface, while still remaining within the planning framework.

Each focus point gives the user the opportunity to view the decision—and the data—currently under consideration. The user can *step* through the process, letting the planner execute only until another focus point is reached. The user can *continue* execution until the planner reaches a focus point that involves a descriptive result, at which point the user can decide how to continue. The user can also *jump* between different focus points, using

knowlege of context to explore the most promising areas. At each point the user can view descriptions of the data being explored, the justifications for individual candidate actions at a focus point, ordering constraints, and descriptions of the plans under consideration.

We begin our analysis by loading the Phoenix dataset. AIDE computes indications for each variable: evidence of clustering, gaps in variable distribution, outliers, and other heuristic evaluations. Without further guidance, AIDE will build a model one variable at a time, in forward-selection fashion, to describe and summarize the patterns it finds. We provide guidance at the outset, however, by specifying that exploration should begin with a subset of the original dataset, the relationship involving Effort and Duration.

Stepping through AIDE's focus points gives the following sequence of decisions. We indicate whether AIDE handles the decision on its own (default) or stops to ask the user for optional guidance (interactive). This sequence is taken from an execution trace of the running system.

Default. Different types of model-building plans match the top level goal, including plans to build regression and cluster models. Without guidance from the user, AIDE selects `generate-opportunistic-model`, a plan that builds a model based on heuristic forward selection of variables. The model is initialized to contain the variables Effort and Duration.

Default. Selection of a plan to explore the relationship between Effort and Duration. Plan selection is based on indications calculated for the relationship. A partition indication is active, triggering a `partition-dataset` plan. Ordering preferences, given the lack of a high correlation indication, put this plan first.

Default. An `initiate-exploration` goal is established for the partitioned dataset. A single plan is activated, `explore-partitions`.

Default. In `explore-partitions` each partition is explored in turn, managed by a variable focus point FP_v. By default the vertical line partition is explored first, with the establishment of an `initiate-exploration` goal.

Default. A zero variance indication for Effort in this partition activates a plan (and deactivates all others) that describes the partition in terms of this indication and features of Duration. This description is returned to satisfy the `initiate-exploration` goal.

Interactive. At this point the planner revisits the variable focus point FP_v, and stops to present its (partial) result. Because each description generated can potentially influence later decisions, AIDE presents each to the user before proceeding. We simply proceed with the single remaining choice, which is to establish an `initiate-exploration` goal for the other partition.

Default. A high correlation indication activates a set of linear fit plans. An indication of outliers imposes a preference for a resistant fit, rather than a least-squares fit. There is also evidence of clustering, as in the earlier description. The activated plans are managed by the creation of a plan focus point FP_p. There are no preferences about whether to pursue the clusters or the linear fit first. By default, the resistant linear fit plan is selected.

Interactive. AIDE generates the fit and presents it to the user. This presentation is again in the context of a focus point, this time to decide whether to examine the residuals of the fit. The line looks like an adequate description, and so we suspend all plans to examine residuals (i.e., lack of fit.) This returns us to the plan focus point FP_p.

Interactive. Because we are satisfied with the resistant fit, we deactivate the regression fit plan. Instead, we now look at the clusters, by selecting the `examine-clusters` plan.

Interactive. There are several distinct ways in which the data fall into clusters. These correspond to choices at a new focus point: we can derive clusters from recurring values in Effort; we can generate clusters using a single linkage method on the bivariate relationship, or on Effort or Duration separately; we can generate clusters by interactively assigning points. Without guidance, AIDE would try each. We select the first option.

Default. A `describe-cluster-behavior` plan is activated. It entails reducing each cluster to a single, representative value (or set of values) and looking for some pattern in the newly generated data. We eventually find that the cluster locations are approximately linear.

The rest of the analysis follows in a similar way. For the most part the user simply guides the analysis in appropriate directions, bringing to bear knowledge of context and superior pattern-matching where appropriate. The results are as described in Section 2.

This example comes from a single analysis, one already performed largely by hand. AIDE can reproduce this analysis, and has been applied to other datasets to generate cluster and regression models and in an earlier version causal models in the course of its exploration [14, 13]. We are currently designing a full set of experiments to evaluate its performance.

6 Related Work in Planning

Partial hierarchical planning was introduced by Georgeff and Lansky [4]. 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 [5]. 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 [1]. 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.

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.

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. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

References

[1] Norman Carver and Victor Lesser. 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), November 1993.

[2] Paul R. Cohen. *Empirical Methods in Artificial Intelligence.* MIT Press, 1995.

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

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

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

[6] David C. Hoaglin, Frederick Mosteller, and John W. Tukey. *Understanding robust and exploratory data analysis.* Wiley, 1983.

[7] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[8] Richard Levinson. A general programming language for unified planning and control. To appear in the Artificial Intelligence Journal's special issue on Planning and Scheduling.

[9] David Lubinsky and Daryl Pregibon. Data analysis as search. *Journal of Econometrics*, 38:247–268, 1988.

[10] Frederick Mosteller and John W. Tukey. *Data Analysis and Regression.* Addison-Wesley, 1977.

[11] Stanley A. Mulaik. Exploratory statistics and empiricism. *Philosophy of Science*, 52:410–430, 1985.

[12] Cullen Schaffer. Domain-independent scientific function finding. Department of Computer Science Technical Report LCSR-TR-149, Rutgers University, New Brunswick, NJ, 1990. PhD Thesis.

[13] Robert St. Amant and Paul R. Cohen. Toward the integration of exploration and modeling in a planning framework. In *Proceedings of the AAAI-94 Workshop in Knowledge Discovery in Databases*, 1994.

[14] Robert St. Amant and Paul R. Cohen. A case study in planning for exploratory data analysis. In *Proceedings of the First International Symposium on Intelligent Data Analysis*, 1995.

[15] Robert St. Amant and Paul R. Cohen. Control representation in an EDA assistant. In Douglas Fisher and Hans Lenz, editors, *Learning from Data: AI and Statistics V.* Springer, 1995. To appear.

[16] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.