A PLANNING REPRESENTATION FOR AUTOMATED EXPLORATORY DATA ANALYSIS[*]

Robert St. Amant and Paul R. Cohen

Computer Science Technical Report 94-18

Experimental Knowledge Systems Laboratory
Department of Computer Science, Box 34610
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610

Abstract

Igor is a knowledge-based system for exploratory statistical analysis of complex systems and environments. Igor has two related goals: to help automate the search for interesting patterns in data sets, and to help develop models that capture significant relationships in the data. We outline a language for Igor, based on techniques of opportunistic planning, which balances control and opportunism. We describe the application of Igor to the the analysis of the behavior of Phoenix, an artificial intelligence planning system.

[*] This research is supported by the Advanced Research Projects Agency and Rome Laboratory under contracts F30602-91-C-0076 and F30602-93-C-0100.

1. INTRODUCTION

Igor is a knowledge-based system designed for exploratory statistical analysis of complex systems and environments. Igor has two related goals: to help automate the search for interesting patterns in data sets, and to help develop models that capture significant relationships in the data. Igor supports and complements the efforts of a human analyst in examining large or complex data sets.[1,2]

Igor explores a data set with the statistical operations of exploratory data analysis (EDA). Examples of EDA operations are univariate power transforms, bivariate relationship partitioning to distinguish separate effects, and analysis of patterns in residuals. The results of these operations are incrementally combined in the framework of a domain model. When promising directions are indicated by domain knowledge, or when interesting patterns are observed the data, Igor selects appropriate operations to explore the new area. As the analysis proceeds, a more complete picture of the data set gradually emerges.

We must consider many different tradeoffs in designing a system for statistical reasoning: operator generality versus power, language expressiveness versus efficiency, opportunism versus control. Of these the key tradeoff is between opportunism and control. EDA operations are powerful: they can be applied in many situations in different combinations, producing results whose proper interpretation depends on context. Effective exploration takes advantage of general and domain-specific heuristics to control their application, and opportunistically incorporates new results into the search process.[3]

We can approach this task with the techniques of opportunistic planning. We have developed in Igor a planning language that balances control and opportunism. By explicitly representing the goals of exploratory analysis, we can take advantage of strategic knowledge about data analysis to structure and reduce search. Explicit control structures monitor the search for 'interesting' results and select appropriate context-dependent sequences of operations during the analysis.

Our work has dealt mainly with analyzing the behavior of artificial intelligence programs. Because an AI system may react in complex ways to the influences of its environment, the reasons for its behavior may not be obvious from execution traces. The enormous search space of possible explanations poses a challenging problem even for human analysts. In Igor we have a prototype that performs useful exploration in this domain while still controlling the complexity of the analysis.

In the next section we discuss elementary strategies for exploring data. In Section 3 we outline the planning representation, the relevance of planning to statistical analysis and how the representation supports the process. Section 4 describes an example of Igor in practice. Section 5 concludes with a discussion of the benefits of the general approach and plans for future work.

2. EXPLORATORY DATA ANALYSIS

Exploratory studies are the informal prelude to experiments, in which questions and procedures are refined. Exploration is like working in a test kitchen: before one writes down the final version of a recipe, one first tries out possible alternative procedures and evaluates the results. Exploratory results influence confirmatory studies in a cycle of successively more refined exploration and confirmation.[4,5,6,7]

We apply two general strategies in exploring data: one generates simplifying descriptions of data, the other extends and refines surface descriptions of data. We simplify data by constructing partial descriptions and models that capture particular characteristics of the data. These descriptions range from simple summary statistics, such as means and medians, to complex domain models. We make descriptions more effective by looking beyond the surface at what is left unexplained. For example, a regression line may be a good description of the general trend in a relationship, but an analysis of residuals can give an entirely different picture at a lower level of detail. Exploratory strategies generate increasingly detailed, complementary descriptions of data.

When exploring a set of data variables, we can combine our results explicitly in a domain model or implicitly in an unstructured set of relationships between the variables. In Igor we begin with an initial causal model, produced by a path analysis algorithm.[8,9] The algorithm generates a directed graph in which nodes represent variables and edges represent direct causal influences between variables. The model provides a framework in which exploratory results can be combined and interpreted. We thus begin with a qualitative model of the data, in terms of causal relationships, and through exploration elaborate the model by generating more detailed quantitative descriptions of these relationships.

We can illustrate the procedure best by example, adapted from Tukey.[7] We begin with a dataset containing variables X and Y, among others. Our initial model indicates that X causes Y ($X \longrightarrow Y$). The relationship is shown in Figure 1.

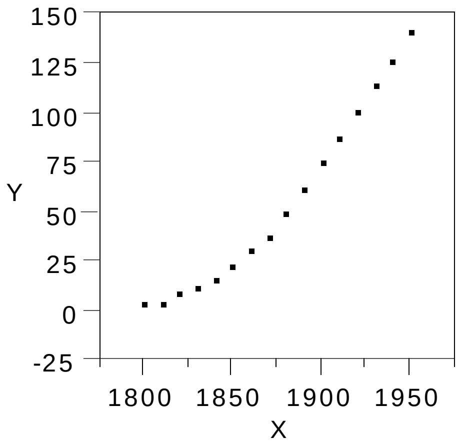


Figure 1: Relationship of X and Y

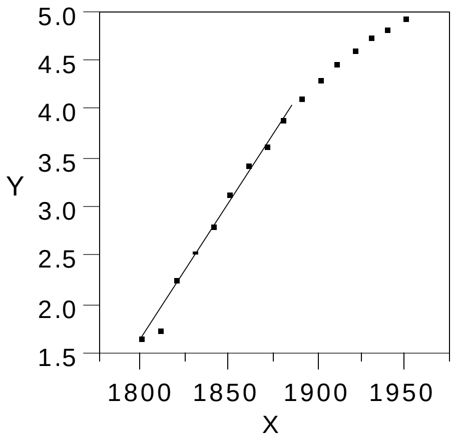


Figure 2: Relationship of X and log Y

We first notice that the relationship between X and Y curves upward. We hypothesize that the relationship is exponential, and apply a transformation by taking the log of each Y value. This kind of transformation serves to straighten the data points so that further structure may be observed. We find, however, that the points are not collinear after the transformation. A pattern in the transformed relationship, shown in Figure 2, tells us that there is further structure to be elucidated. We return to the original data scale to try a different approach.

Looking more closely we see that the curve in the relationship is more pronounced for lower values of X; the slope of X and Y seems close to constant for higher values of X. We thus partition the relationship and analyze each partition separately. This approach turns out to be more successful: while the lower partition is indeed represented well by an exponential curve, the upper partition may be fit just as well with a straight line.

Subtracting the effects of these curves gives us the residuals, a closer view of the data. We notice, for example, that one of the data points in the lower partition in Figure 2 lies far from our proposed description. If the point reflects a real effect on Y for this value of X, our linear fit of the lower partition is inadequate. It may be that the point is anomalous, however; further exploration, perhaps taking other variables into account, can help us decide.

Consideration of the fit and residuals, which we interpret as descriptions of trend and local detail, helps us build a relatively detailed description of the relationship between X and Y. In addition to the initial description $X \longrightarrow Y$, we can further say that Y increases exponentially for X values between 1800 and 1875, but slows to a linear rate thereafter. Local effects can be seen in the residuals of the trend descriptions. Our construction of this composite description was guided by observation of an inflection point in the data, and the realization that the first, simpler model was not sufficient to capture significant characteristics of the data.

Our description has neglected much of the complexity of the decisions involved. In general there are choices to be made about which techniques best suit the situation and which results are most interesting to pursue. When we explore data effectively we exploit a tension between opportunism and control. Opportunism lets us explore new interesting results that arise unpredictably. Control determines which results are promising, how they might best be evaluated, and when particular paths of reasoning might be abandoned. Without a proper balance between opportunism and control, an automated system may be unable to make the simplest discoveries, or may face an explosive search space.

3. PLANNING FOR DATA EXPLORATION

In the foregoing example we incrementally built a structured interpretation from raw data. We can view the process as a form of planning. In planning, sequences of simple operations are combined for complex effect. Planning may be incremental and opportunistic, based on constructing and revising plans according to information acquired during the process.

Using this analogy we have developed a planning language for Igor, based on the RESUN signal interpretation system.[10] RESUN is a blackboard-based control planner which supports goal satisfaction through flexible combination of scripts and actions. Opportunism is managed by mechanisms that monitor the state of plans and by heuristics that determine when and how particular goals are satisfied.

At the lowest level, the data structures manipulated in the planning representation are frames. A variable is a simple frame; a linear relationship between two variables is a slightly more complex hierarchy of frames; an annotated causal model is a highly interconnected hierarchy of frames. Igor allows specialization of two built-in data types – *basic sequence* and *basic relationship*. We call frames and hierarchies of all types *structures*.

The primitive operations provided by the representation are called *actions*. An action is a data transformation or decomposition of an input structure into an output structure. A log transform is a simple example of an action; it applies a log function to each element in a sequence and collects the results. More complex transformations include smoothing, outlier removal, and fit operations. Each action has an associated goal form and may be triggered by the establishment of a matching goal. The definition of the log transform action is shown in Figure 3.

```
(define-action (log-transform-action
          :goal  (log-transform-goal ?variable ?result)
          :input (variable)
          :output (result))
          :action (log-variable variable))
```

Figure 3: Action form for log transform

Actions are combined in *scripts*. A script is a sequence of subgoals whose associated actions transform one structure into a more concise, better parametrized, more descriptive structure. Scripts, like actions, have associated goal forms, and thus may be combined hierarchically to satisfy the goals of other scripts. Combination of subgoals in a script is governed by the *specification* of the script. A script specification defines how its subgoals must be satisfied in order for the top level goal to be satisfied. Specification constructs allow sequential combination of subgoals, iteration over sets of subgoals, conditionalization on tests of variable values, and activation of subgoals in parallel.

The example script in Figure 4 combines a conditional test of a relationship with a sequence of subgoals. Acting on a relationship X—Y, this script first determines that X has discrete values. This means that we can partition the relationship by X to see how the values of Y behave in each partition. We do this by mapping a statistic, the mean, over each partition. The result is a more abstract relationship than the orginal, in which much of the detail has been stripped away. Such a transformation is often useful in deriving simplified descriptions of data.

```
(define-script  build-discrete-vs-mean-script
    :goal   (explore relationship ?relationship ?context)
    :input  (relationship context)
    :bind   ((x-variable (component ?relationship 0))
             (y-variable (component ?relationship 1)))
    :output  (transformed-relationship)
    :spec  (:COND (x-variable) (discrete-valued-variable-p x-variable)
                (:SEQ  partition-y-by-x
                       relationship->list
                       reduce-x
                       (:MAP (partitioned-y)
                             map-mean-over-partitioned-y)
                             list->sequence
                             relate-results
                             post-results)))
```

Figure 4: Script form for partitioning a variable

While the script representation addresses the main tradeoff between control and opportunism, we must address other issues as well: how scripts should be triggered, in which order they should run, and how results may be shared between different threads of exploration. For this we rely on an explicit representation of *context*, and two mechanisms that depend on context, *monitoring* and *focusing*.

A script matches a goal not in isolation but in an environment containing the goals and scripts that have been expanded beforehand. We make this environment explicit in a *context* structure associated with each plan. A context structure is simply the sequence of intermediate and end results produced by execution of a script. Because script subgoals may be satisfied by other scripts, a hierarchy of contexts is built up during exploration. This hierarchy provides contextual information to monitoring and focusing mechanisms.

As intermediate and final results are generated, they become available in the context of the script that produced them. When a script matches a goal, posts its subgoals, and eventually runs to completion, its results propagate to the context of the higher level script that posted its goal. Results propagate from low levels up to the more abstract levels, where they can be evaluated for their relevance to other areas of exploration.

Focusing heuristics guide the exploration process based on information retrieved through the context hierarchy. Focusing heuristics are activated whenever there is a choice between which goals to pursue and which scripts to apply; they evaluate the precedence of active goals and the relevance of matching scripts. A focusing heuristic produces an ordering of the goals or scripts it manages, and is free to prune the set, temporarily or permanently. We use focusing heuristics to evaluate the usefulness of pursuing particular search paths.

A *monitor* is a special kind of focusing heuristic that is activated by the detection of a new result through the context hierarchy. A strictly goal-driven system can find it difficult to take new structures under consideration during the search process. Monitors establish *exploration goals* for new results. Monitors evaluate the 'interestingness' of results, to see which kinds of exploration, and hence goals, are relevant.

The context hierarchy serves to limit the amount of information that focusing heuristics, especially monitors, must consider in their evaluation. We have designed scripts in Igor, for example, to consider first sets of bivariate relationships, then the individual relationships in a set, and then individual relationships containing specific types of variables. This lets us define a monitor to detect variable dependence, using the χ^2 test, which will act only on relationships between categorical variables. It needs access only to information about individual relationships. In contrast, a monitor that detect similarities between relationships operates at a higher level context, in which information about many different relationships is stored.

The hierarchy is not rigid; results are stored on a global blackboard defined to make hierarchical access most efficient. Focusing heuristics that apply locally are thus inexpensive. Less context-specific heuristics may also run, but at a greater cost, because of the larger space of results they must search through. The more non-local information a heuristic uses, the greater the cost of its application.

In this planning framework we incorporate domain-specific knowledge as well as general heuristics to guide exploration. Domain knowledge enters in three ways. First, we can design structures that support the kinds of operations we wish to perform. In our case we specialize the basic-relationship to a weighted causal-relationship, and define a model as a set of these relationships. Second, we can design scripts that perform domain-specific operations on data. For example, if variable T contains completion times for a task that doesn't always finish, we may find it more convenient to deal with the reciprocal of T. Third, and most significantly, we can use domain information to guide search with focusing heuristics. We will discuss an example of this in the next section.

4. AN IGOR EXAMPLE

In this section we examine Igor's analysis of the results of an experiment with Phoenix, a simulated environment populated by autonomous agents.

Phoenix simulates forest fires in Yellowstone National Park and the agents that fight the fires. Agents include watchtowers, fuel trucks, helicopters, bulldozers, and, coordinating the efforts of all, a fireboss. Fires burn in unpredictable ways due to wind speed and direction, terrain and elevation, ground cover type and natural boundaries such as rivers, roads, and lakes. Agents behave unpredictably as well, because they instantiate plans as they proceed, reacting to immediate, local situation changes. Phoenix is a complex simulation; even when the agents are successful in containing a fire we may find it difficult to explain why particular behavior is effective. We thus run experiments in which we control and test specific aspects of Phoenix.

In one experiment we examined the relationship between the thinking speed of the fireboss and the rate of environmental change. Thinking speed is controlled by the Real-Time Knob, or RTK, which sets the ratio of CPU time in the fireboss to simulation time in the environment. Thus, for example, during the development of Phoenix the value of RTK was fixed at 1.0, which corresponds to five minutes of simulation time elapsed per one second of CPU time. By setting the value of RTK higher or lower we change how quickly the fireboss can react to external events and build plans to deal with them. Environmental change in this experiment is influenced by wind speed. When wind speed is low, fire spread is slow and predictable. At higher wind speeds fire spreads more quickly and takes less predictable paths over the ground cover.

We created a fire fighting scenario to be fought by the fireboss and four bulldozers. We ran 385 trials using the same basic scenario but setting wind speed and RTK to different values at the beginning of each trial. During each trial we collected some forty measurements, including Wind Speed (WS), RTK, Success, and Shutdown time (SD), as shown in Table 1.

Table 1: Phoenix parameters

Variable	**Abbr**	**Values**
Wind Speed	WS	3, 6, 9 km/hr
Real-Time Knob	RTK	0.333, 0.454, 0.555, 0.714, 1.000, 1.666
Success	Success	0, 1
Shutdown time	SD	continuous values

We first describe the steps Igor takes in analyzing this data set and then summarize with a broader picture of the exploration process. Igor begins its analysis with a single goal, explore-dataset. The top level script that matches this goal, explore-dataset-script, first generates an initial model with generate-model, and then expands into parallel subgoals, explore-variables and explore-relationships. These subgoals are satisfied by scripts that expand into goals for the exploration of individual variables and the model relationships between them.

Exploration of single variables involves scripts for straightening variables through power transforms, detecting discontinuities and regions of constancy in variable values, finding possible clusters, and calculating standard summary statistics. Exploration of relationships includes 'untilting' skewed relationships with power transforms, mapping categorical variable relationships into contingency tables, and applying linear fit functions and testing residuals. On detecting a particular characteristic in a structure, a script most often produces a transformation of the structure into a form in which the characteristic is easily seen and manipulated. These scripts are not applied in linear fashion, but are rather selected by goal matching, pruned and ordered by focusing heuristics, and re-applied by monitors.

One avenue of exploration concerns the relationship between the probability of success and thinking speed. This is interesting because it deals with a high level description of the relationship between two primary variables in the experiment, RTK and Success. One script activated by the explore-relationship goal is the discrete-vs-mean-transformation. The script partitions Success by the values of RTK, reducing RTK to six distinct values, and mapping the statistic mean over each of the partitions. (Because Success has binary values, 0 for failure and 1 for success, the mean of a partition is equivalent to the proportion of successes in that partition.) The resulting sequences, containing the reduced values of RTK and the mean values of Success per partition, are associated in a new relationship, RTK-pS. We can interpret this relationship as the probability of succeeding given a particular value of RTK. The relationship RTK-pS is shown in Figure 5.

The exploration process occurs in parallel for other relationships in the dataset as well. For example, the relationship between RTK and mean Shutdown time, RTK-mSD, is generated by a process similar to the one described above. The resulting relationship is shown in Figure 6.

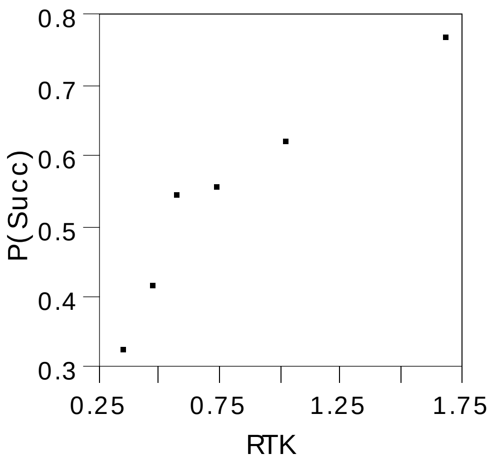


Figure 5: RTK-pS relationship

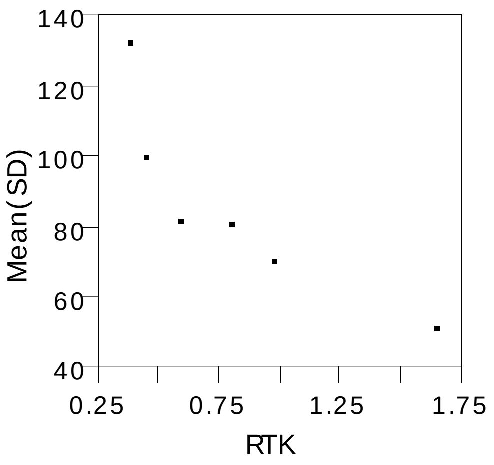


Figure 6: RTK-mSD relationship

The result of the discrete-vs-mean-transformation is a transformed relationship. A monitor at the explore-relationships level detects the creation of the such relationships, and posts exploration goals for each of them in turn.

One of the exploration goals generated for RTK-pS is the sequence-fit goal, which is matched by the linear-fit script. The linear-fit script fits a line to the relationship, generating both a simple regression equation and a sequence of residuals. The residuals sequence is detected by a monitor and examined for patterns. One of the simplest pattern-detecting scripts finds that the residuals of RTK-pS are bitonic, increasing then decreasing. A monitor makes this available in the context of the exploration goal of RTK-pS.

Now the activation of scripts to match the sequence-fit goal is controlled by a focusing heuristic at the explore-relationship level. If the linear-fit script produces a good enough fit (i.e., with enough variance accounted for) and there is no structure in the residuals, then the goal is satisfied. Here, however, because of the residuals, the focusing heuristic continues the search. Two further possibilities are available. The first is to produce a transform for RTK-pS and retry the linear-fit script. The second is to partition the relationship at or near the inflection point in the residuals, and generate a fit for each partition separately during exploration.

Both these approaches are plausible. Because we know that the mean of Success can never rise above 1.0, we could fit a sublinear function to the relationship, with asymptote at 1.0, as shown in Figure 7. This gives a better fit than a straight line, but still not a perfect one. (The marked point is considered an outlier under this interpretation.) Alternatively we recall that the initial setting of RTK is 1.0, the setting at which the system was designed and tested. We find that partitioning the relationship near this value gives us a good linear fit for each partition. Under this interpretation we see not a continuous relationship between the variables but two separate modes of operation, one at low (inadequate) thinking speed and the other at high (adequate) thinking speed. This is shown in Figure 8.

Neither of these competing interpretations clearly rules out the other. If we believe that the design of Phoenix provides smooth degradation in performance as thinking speed decreases, we might prefer the first interpretation. The second interpretation is also plausible, except that degradation is not smooth after some threshold value of RTK is reached. The additional evidence that this threshold occurs near the RTK setting of 1.0, possibly indicating that Phoenix plans rely implicitly on this setting, inclines us to credit the second interpretation.

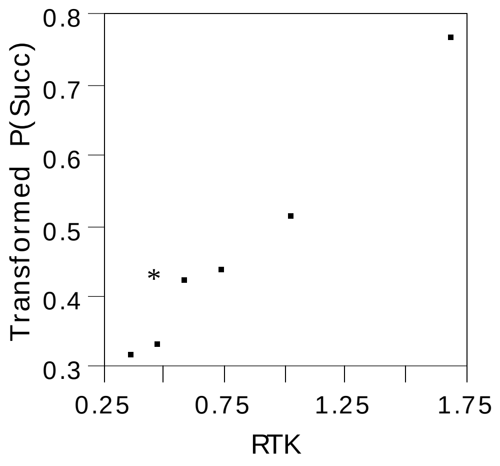


Figure 7: Straightened RTK-pS

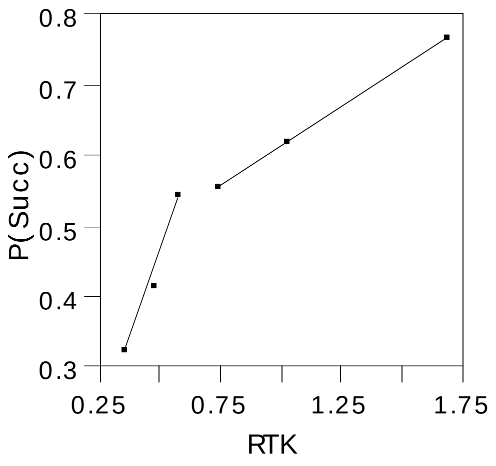
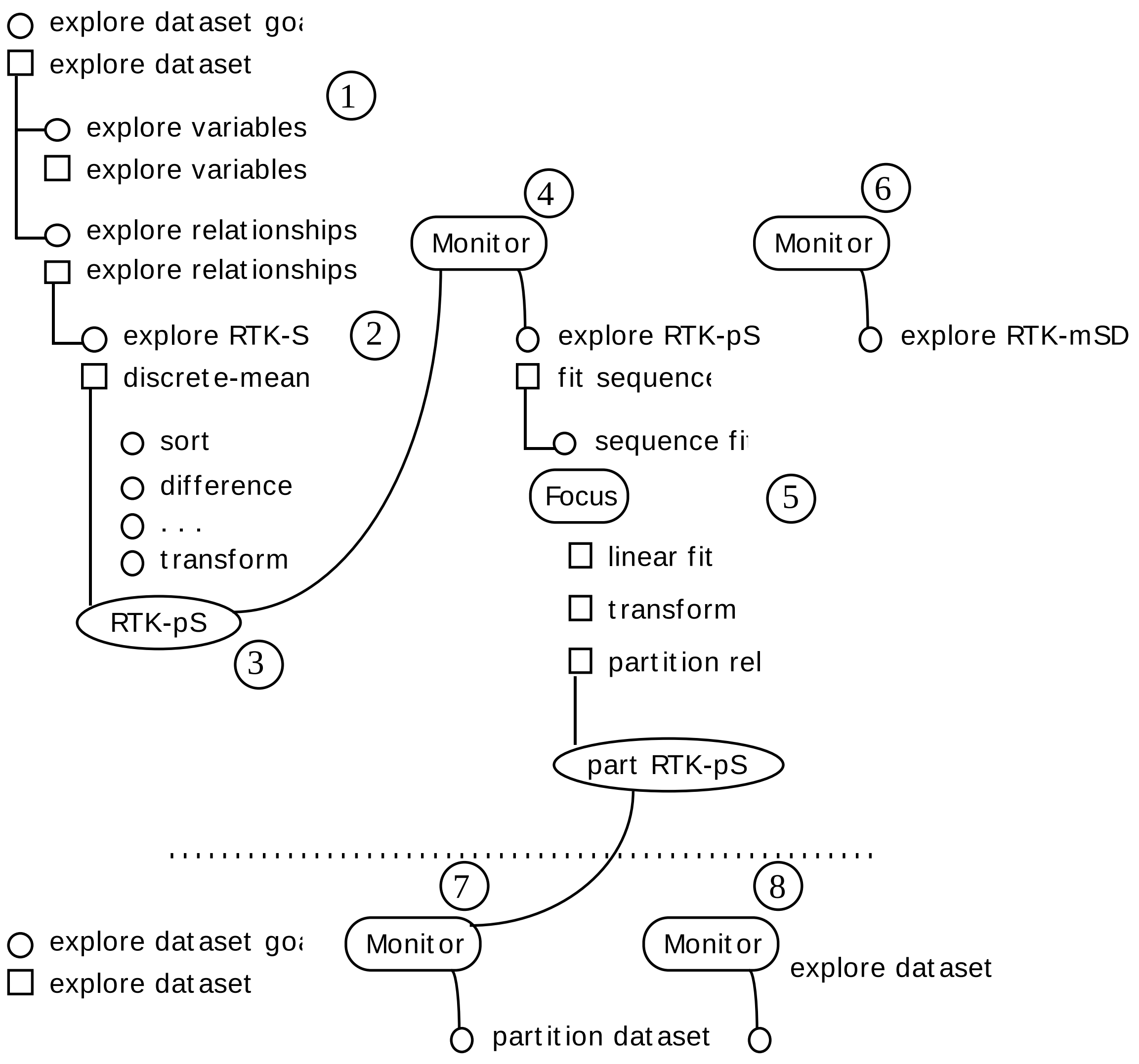


Figure 8: Partitioned model of RTK-pS

A third possibility now arises. If we consider the adequate/inadequate partitioning of RTK plausible, the effect may not be limited to the variable Success. In fact, we see a similar effect in the relationship between RTK and Shutdown time, as shown earlier in Figure 6. The relationship RTK-mSD shows a discontinuity at the same value of RTK. This prompts the partitioning of the entire dataset by adequate and inadequate RTK.

In the plan representation, these discontinuity effects are detected by monitors and recorded in the context structure at the `explore-relationships` level. The similarities of these effects is detected at the higher `explore-dataset` level, which activates a dataset partitioning goal. A new dataset is produced for each partition of RTK, and processing continues with the generation of initial models and exploration of their relationships and partitioned variables.

Finally we present an overview of Igor's processing. Figure 9 traces significant points in plan and goal instantiation during the exploration process.



1. Exploration goals are established for variables and relationships.
2. Relationship RTK-S is explored, triggering the `discrete-vs-mean-transformation`.
3. Relationship RTK-pS is generated.
4. A monitor detects RTK-pS and establishes an exploration goal for the relationship.
5. A `sequence-fit` goal is established for RTK-pS; a focusing heuristic tries different possible interpretations for RTK-pS.
6. Similar processing occurs for the RTK-mSD relationship.
7. A similarity between the partitioned relationships is detected at the `explore-dataset` level; the dataset is partitioned.
8. The partitioned datasets are explored in turn.

Figure 9: Exploration trace

5. CONCLUSIONS

We have examined some of the issues involved in automating exploratory data analysis, in particular the tradeoff between control and opportunism. We have proposed an opportunistic planning solution for this tradeoff, and have implemented a prototype, Igor, to test the approach. Our experience in developing Igor has been surprisingly smooth. In contrast to earlier versions that relied on rule representation, it has been straightforward to increment Igor's knowledge base without causing the search space to explode. The planning representation appears to be both general and powerful, with high level strategic knowledge provided by goals and plans, and hooks for domain-specific knowledge provided by monitors and focusing heuristics.

Our future plans include incorporating detailed domain-specific knowledge into Igor, which will let us explore the interaction between general strategic knowledge and domain-specific control knowledge.[11,12] We also wish to continue our examination of the relationship between exploratory data analysis and automated causal modeling. Some of the results produced by EDA appear to be useful cues to humans in developing causal models. We have concentrated on showing how EDA operations can augment a causal model; a causal model should conversely be able to give search guidance to Igor, for example in distinguishing between proximal and distal causes of an observed effect.

Our final concern is with evaluation. How sensitive is Igor to its parameter settings? How much information does Igor require to draw conclusions about a particular effect? Are the chains of statistical reasoning Igor produces coherent to humans? We will explore these and other questions in future work.

6. ACKNOWLEDGEMENTS

This research is supported by the Advanced Research Projects Agency and Rome Laboratory under contracts F30602-91-C-0076 and F30602-93-C-0100. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

7. REFERENCES

[1] St. Amant, R., and Cohen, P.R., 1993. Automated Analysis of Complex Data. Computer Science Department Technical Report. University of Massachusetts, Amherst.

[2] Silvey, P., 1993. IGOR: The MAD Scientist's Assistant or Building Models from Data. Computer Science Department Technical Report. University of Massachusetts, Amherst.

[3] Kulkarni, D., & Simon, H.A., 1990. Experimentation in Machine Discovery. In Schrager, J., & Langley, P., (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann. 255-274.

[4] Cohen, P.R., 1994. *Empirical Methods for Artificial Intelligence*. MIT Press. Forthcoming.

[5] Gale, W.A., 1986. REX Review. In Gale, W.A. (Ed.), *Artificial Intelligence and Statistics*. Addison-Wesley. 173-224.

[6] Hand, D.J., 1986. Patterns in Statistical Strategy. In Gale, W.A. (Ed.), *Artificial Intelligence and Statistics*. Addison-Wesley. 335-338.

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

[8] Cohen, P.R., Carlson, A., Ballesteros, L., St. Amant, R., 1993. Automating Path Analysis for Building Causal Models from Data. *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann. 57-64.

[9] Cohen, P.R & Hart, D.M., 1993. Path analysis models of an autonomous agent in a complex environment. *Proceedings of The Fourth International Workshop on AI and Statistics*. Ft. Lauderdale, FL.

[10] Carver, N., & Lesser, V., 1993. A Planner for the Control of Problem-Solving Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, vol 23, no. 6.

[11] Langley, P., Bradshaw, G., & Simon, H.A., 1983. Rediscovering Chemistry with the BACON System. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning An Artificial Intelligence Approach*. Morgan Kaufmann. 307-330.

[12] Schrager, J., & Langley, P., 1990. Computational Approaches to Scientific Discovery. In Schrager, J., & Langley, P., (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann. 1-26.