StatSci Research Report 17

Brushing Histories to Compare Models

Russell G. Almond

StatSci (a division of MathSoft, Inc.)
1700 Westlake Ave, N., Suite 500
Seattle, WA 98109
almond@statsci.com

Revision 2
3/30/94

This research is supported by the GRAPHICAL-BELIEF project, NIH SBIR contract 1 R43 RR07749-01 and NASA SBIR contract NAS 9-18908.

Brushing Histories to Compare Models

Russell G. Almond, StatSci

ABSTRACT

A common high level task for the user of complex modelling software is making comparisons between models. The differences between the current and previous models guide future modelling efforts and model acceptance. This paper explores an interactive graphical paradigm to support this task. The computer displays the *state history*—the series of states the system takes one—corresponds closely to a series of formal models in the mind of the user. The user can *brush*—temporarily select—previous models through this display. During this brushing, *probes*—graphical displays of model statistics—track the corresponding models. This interaction with previous states simplifies the task of comparing models. The paper illustrates these ideas with applications to decision analysis (graphical models) and statistical model fitting.

1.0 Introduction

Consider a decision analyst who is uncertain about the value of a particular parameter in the model. The analyst would build several similar models which differ in the value of that parameter and compare the results. If the value of the parameter makes little impact on the final decision, no further work is necessary. If the parameter makes a large impact, the analyst must refine the model. Such *sensitivity analysis* is an important part of the modelling process, but is often poorly supported by decision analysis software models.

To build a more formal knowledge of this process, let $M_0, \ldots M_n$ be a sequence of models built by the analyst. These models correspond to a sequence of states $S_0, \ldots, S_n$ of the computer program. In order to validate the current model, or to suggest improvements in it, the analyst must compare models. In particular, the user wishes to compare the value of some *statistic* (function of the model) for two models. Thus the basic task the computer must support is comparing $t(M_i)$ with $t(M_j)$.

If the statistic $t(\cdot)$ is a single valued function, the computer could present the comparisons as a table of values of the statistic, however, the analyst must still break that table down into the individual comparisons. This paper instead suggests presenting the analyst with a graphical display of the model history and allows the analyst to select the comparisons she wishes to make. *Probes* display the values of the important statistics for the currently selected models, dynamically tracking the analyst's selections. This dynamic interaction is related to the idea of scatterplot brushing (Becker and Chambers[1987]) and is expected to have many of the same improvements in productivity.

The traditional approach to history focus on the use of meta-commands (like *undo*, *skip* and *redo*) for error recovery; Archer, Conway and Schneider[1984], Vitter[1984], and Yang[1988] provide some formal descriptions. They define history in terms of a *script* or series of commands, $c_1, \ldots, c_n$. The meta-commands manipulate this script, allowing the user to bring the system into the desired state. Many word processing, text and graphical editing programs use variations on these ideas for error recovery; these applications are the typical examples for work on command histories. The script history approach is the dual of the model history approach outlined here; section 4 integrates ideas from this literature with the current approach.

The remainder of this section describes the two motivating examples, one from decision analysis (section 1.1) and one from statistical model fitting (section 1.2). Section 2 introduces the *History Stack* a formal representation of the history of states. Section 3 describes some probes and ways of manipulating history stacks to compare states. Section 4 describes the principle meta-commands for manipulating history stacks. Section 5 explores the limit of the linear model of history and the possibility of tree shaped alternatives.

1.1 Graphical Models for Prediction and Decision Support

Graphical belief models (Almond[1993], Shenoy and Shafer[1990], Lauritzen and Speigelhalter[1988], Pearl[1988]) are distributed models for uncertain reasoning. A model graph, such as the one shown in Figure 1, is a representation of the factorization of the model into a series of independent submodels which could be probabilities, belief functions or other representations of information. A “fusion and propagation algorithm” (described in the above references) allows global information to be calculated about a small set of variables without ever directly representing the full joint distribution of all variables in the model.

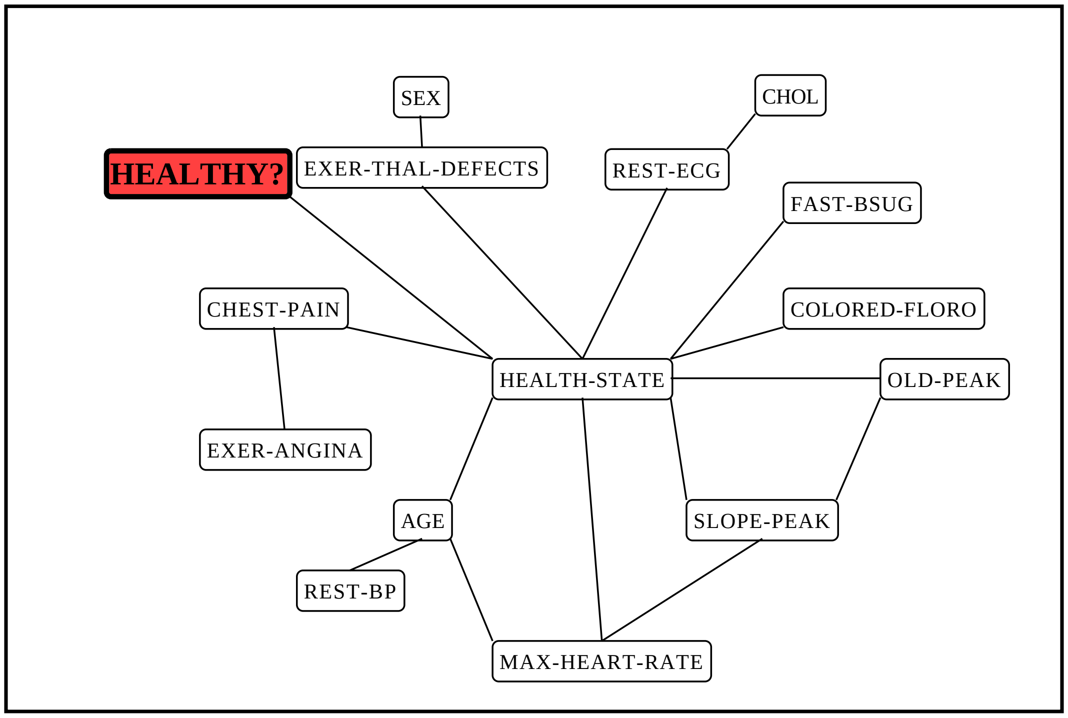


Figure 1. Coronary Artery Disease Model
Graphical model display in the program GRAPHICAL-BELIEF *of the model fit to the Coronary Artery Disease data of Detrano* et al.*[1989]. The analysts interacts with this model by setting values for the variables (nodes in the graph). The node* Healthy? *is highlighted because there is an active* probe *associated with it (Figure 3).*

The example in Figure 1 is a model fit to the Coronary Artery Disease Data of Detrano *et al.*[1989] (from the Murphy and Aha[1992] repository). First a statistical model was fit to the historical data (typically graphical models are formed from a combination of historical data and expert opinion encoded as probabilities or belief functions). This was used as input to the program `GRAPHICAL-BELIEF` (Almond[1992]) which uses the model as a sort of medical expert system, predicting medical risk for various variables in the model given certain combinations of other variables.

A typical scenario of use might begin with entering specific background data about the patient (or fetching it from medical records). This would set the value of variables such as *Age*, *Sex* and *Rest Blood Pressure*. Next, information pertaining to recent patient specific data (such as current complaints) would be entered; this might effect variables such as *Chest-Pain* and *Cholesterol Level*. In order to assess the importance of test results for an expensive or invasive test, we can hypothesize the result of the test; for example, hypothesizing the value for the *Rest-Ecg* could reveal whether the information obtained would have much impact on the result of the computation. A more sophisticated user might temporarily set *distributions* which link a number of different variables in the model, or a *parameter* (numeric value in that distribution) to study sensitivity of the conclusions to modelling assumptions.

This scenario of use involves a number of global states of the system; each state (S_j) corresponds to different patterns of instantiation among the variables of the model (*Sex*=`Male`, *Age*=`60+`, *Rest-Ecg*= *Unknown*, ...). Many of these model states are hypothetical models meant only to be temporary entities, to be compared to other models based on either real data or other hypotheses. The example above only uses two operations: `set-variable`, and `unset-variable`, although GRAPHICAL-BELIEF also supports `set-distribution`, `unset-distribution`, `set-parameter` and `unset-parameter` operations. The GRAPHICAL-BELIEF implementation contains two other commands `set-parameter` and `set-law` plus the corresponding unsetting commands.) GRAPHICAL-BELIEF propagates each change to the model far enough to support any queries the analyst might make about the model.

Note that many important operations in this domain (such as calculating the value of information, or assessing the impact of influential observations) involve comparing the models before and after changes. History meta-commands designed to support error recovery are typically not sufficient for this application.

1.2 Model Selection

Fitting such a graphical model to a set of data involves a very general statistical operation known as model selection. In model selection a data analyst picks a model which is best under some sort of heuristic measure of goodness of fit. Typically, a data analyst performs model selection by a manual stepwise search (Chapter 5 of Daniel and Wood[1980] provides a good annotated example of a typical search pattern); this enables the analyst to balance information from goodness of fit measures, graphical diagnostic displays and domain knowledge about the problem not encoded into the computer.

One early language to explicitly support model selection is `GLIM` (Healy[1988]). In `GLIM` models are represented through formulas using a language developed in Wilkenson and Rogers[1973] plus some extra parameters determining the distribution of errors about the expected value. Thus the model `Health ~Max-Heart-Rate * Age + Chest-Pain` would indicate that predictions of the variable *Health* should by modelled with a term for *Chest-Pain* a term for *Max-Heart-Rate*, a term for *Age* and a term for the interaction between *Age* and *Max-Heart-Rate*. Chambers and Hastie[1982] describe a more recent implementation of the Wilkenson and Rogers language in the S-PLUS environment along with several examples of the model fitting process.

Model selection is essentially a search through the space of all possible models. The search is usually performed by stepping through model space, adding or subtracting one interaction at a time. Thus the effective history is the list of models which have been considered. Models are compared on the basis of heuristic measures of goodness of fit, such as the deviance, often modified by adding a reward for parsimony (the number of variables and interactions in the model, or parameters). Bayesian model selection uses the posterior probability of the model (the probability of the model given the observed data) and contains a built in penalty for additional variables.

Measures of goodness of fit are not always numbers. Plots of the predicted values or the residuals (differences between predictions and observations) are robust indicators of lack of fit, especially in regression theory (most recent texts on regression cover this topic, see, for example, Daniel and Wood[1980]). In particular, plots often depict trends not expressed in the model (for example curves when the model is linear) and suggest transformation of one of more of the variables. These transformation can also be expressed as stepwise changes in the model.

Because model fitting software typically forces the user to work with one model at a time (and doesn't support comparisons) there is a dangerous tendency to accept a single best model and then make predictions as if that model were the truth. In this case, the analyst ignores the uncertainty in the predictions due to uncertainty in the model. Several authors (*e.g.* Breslow[1991], Madigan *et al.*[1993]) have recently pointed out that averaging over several models can produce better calibrated predictions than relying on just one model, however, even the simple sensitivity analysis described in this paper will at least alert the analyst to a potential problem.

It is also important for the data analyst to be able to interact with the model selection process. Understanding which models fit best produces greater understanding of the physical system being modelled. Scientific progress is usually a function of overthrowing an older model for a newer more descriptive one, rather than simply fitting a single best model.

All of these factors indicate a single style of interaction between the data analyst and the computer. The user creates a series of models (usually by stepwise differences between the current and new models). These models are evaluated using various metric. Although the best model is selected at every step, the analyst will frequently backtrack to compare the current model with previous models. History meta-commands designed only to support error recovery are not only insufficient, they encourage the dangerous single correct model theory. The challenge is to create an interaction paradigm which will support this type of analysis and support the consideration of many models simultaneously.

2.0 The History Stack

In both of the applications described above, the system takes a series of states, $S_0, S_1, \ldots$, corresponding to a series of mathematical models, $M_0, M_1, \ldots$. This series of states is the *history stack.* The state history stack can be represented as an object in the user's environment. It corresponds closely to the model history stack which resides in the user's mental model of the system.

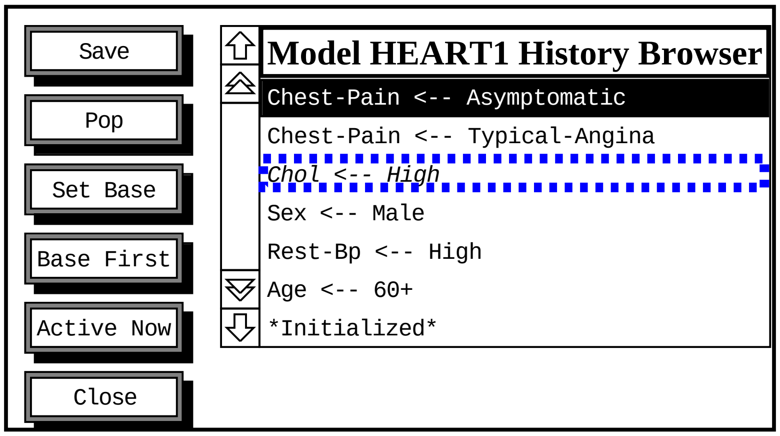
A series of commands (or command scripts) cause the transitions between states. For example, executing the command c_1 causes the transition between states S_0 and S_1. The sequence of commands forms a script view of history, such as the one used in Archer, Conway and Schneider[1984]. Note that the state history view: $S_0, S_1, \ldots$ and the script view $c_1, c_2, \ldots$ are duals. In the model manipulation and fitting applications (described in section 1),the analyst uses (explicitly or implicitly) the sequence of mathematical models $M_0, M_1, \ldots$ to perform analysis tasks.

Each state S_j is assigned a name, d_j which is known as the *date.* The date does not need to be a timestamp; it can be any sort of unique identifier. In the model selection problem (section 1.2), the Wilkenson and Rogers[1973] language provides a natural set of labels. In the graphical model domain (section 1.1), GRAPHICAL-BELIEF uses a simplified form of the command c_j as the date, as this serves as a useful reminder to the user.

Adopting the state history model, as opposed to the command history model has important implications in the decision of which commands to include in the command history. Data analysts interacting with modelling software typically issue three kinds of commands: (1) commands which change the model, (2) commands which provide information about the current model and (3) commands which provide information or modify aspects of the system state unrelated to the model. In the state history model, only the first type of command is included in the command history. The second type of command is typically repeated for each model; section 3 develops the concept of *probes* which automatically track statistics related to model state. The third type of command covers all sorts of miscellaneous manipulations, including escapes into the operating system for unrelated tasks. It also covers some important meta-commands such as "save" and "undo" which are discussed in section 4.

The program GRAPHICAL-BELIEF provides a visual display of the history stack which the user can manipulate (figure 2). Some meta-commands are available through this display, but its primary purpose is to simplify comparisons between current and past states. The user can make temporary selections (brushing) on this display and special displays of model statistics *probes* react to those selection. Section 3 describes this process in more detail.

Maintaining complete a complete history of states can use a large amount of system memory. Two different memory saving devices are available: distributing the state over many local states (section 2.1) and maintaining only partial histories at many sites (section 2.2).



```
Chest-Pain <-- Asymptomatic
Chest-Pain <-- Typical-Angina
Chol <-- High
Sex <-- Male
Rest-Bp <-- High
Age <-- 60+
*Initialized*
```

Figure 2. History Browser Gadget from GRAPHICAL-BELIEF
The scrolling list is a complete record of all states of the model in Figure 1. Using the mouse, the analyst can select two distinguished states for comparison (see Figure 3 for the results of that comparison). The state in reverse video is the active *state, the state in the dashed box is the* baseline *state. The baseline state corresponds to the information available from the patient's chart (the "*Initialized*" state corresponds to a generic patient). The buttons along the side perform meta-commands (manipulations of the history); "pop" corresponds to "undo." This figure is typical of how the user could use the system to assess the impact of the variable* Chest-Pain *on the patients overall health measurement.*

2.1 Distributed Histories

The graphical models for decision support (section 1.1) naturally store the state in a distributed fashion. In the fusion and propagation algorithm, the model graph is reformulated into a *junction tree* which smooths out the loops of the model graph. Associated with each node in the junction tree, is a collection of variables in the model. Each node of the junction tree stores information in the space associated with those variables.

Each node of the junction tree stores the following information: (a) a local factor of the total belief function or probability model, (b) incoming messages from each of its neighbors describing their contribution to the global model and (c) a projection of the global model (the combination of all local factors) onto the space of the variables in that node (the *local margin*). All of the manipulations of the model described in section 1.1 amount to changes in one or more of the local factors (a) associated with a given node. Most of the user's queries relate to questions about the local margins (c). Queries about the messages (b) are useful for explaining unusually properties of the model and debugging model specifications.

To store the histories in a distributed fashion, GRAPHICAL-BELIEF uses both a global history stack associated with the full model and three local history stacks for each node, one for each of the three kinds of information. The global history stack is a list of pairs (d_i, c_i) where d_i is the *date* and c_i is the command issued at that date. The local history stacks are pairs of the form $(d_i, B_{\mathbf{N},i})$ where $B_{\mathbf{N}}$ is the appropriate kind of information for node $\mathbf{N}$.

The dates d_i can be used as key into the database associated with each node. One can then call for local information associated with any node in the model at any date. This drives the probe system described in section 3.

2.2 Partial Histories

The user will not want information from all sites in a distributed model. Label those sites (nodes in the junction tree model) from which the user is currently interested in information as *interesting* (these are the sites with probes attached). Only the interesting sites need complete history information. Other sites simply need partial histories which track changes.

In the current GRAPHICAL-BELIEF implementation, GRAPHICAL-BELIEF maintains only a partial history for the changes to the local factor (a) and for the new incoming messages (b). If a node is interesting, then GRAPHICAL-BELIEF maintains a complete local history for the margin of the global distribution (c). If a node changes status from uninteresting to interesting, then GRAPHICAL-BELIEF builds the local history for the margin of the global distribution (c) from the partial histories for the local factor (a) and messages (b).

3.0 Probes

The previous section identified three different kinds of command: commands which changed the model, commands which provided information about the model, and unrelated commands (primarily session management). The history stack is only concerned with the first kind of command. This section takes up the second kind of command, and looks at ways to provide information about the model's state.

Often, the analyst issues similar informational commands for each model. The principle purpose for this repetition is to compare models on various features. Making comparisons across model is often a tedious process requiring the analyst to scroll backwards through output transcript both looking for the previous values of important statistics and cutting and pasting commands to be re-issued. If the transcript buffer is too small, the analyst must refit the model and re-issue the informational command. Clearly, an intelligently designed interface could provide better support for these between model comparisons.

Probes are a way of dynamically tracking changes to the state or model. Formally a probe is a function which takes a state and returns a value (a number, a series of numbers or a more complex entity such as a plot or other picture). Simple probes track the current state of the system. For example, a word processing program might provide a probe which shows the number of lines or pages in the finished document.

In order to facilitate the between model comparisons, the user should be able to associate probes with any state in the history stack, not just the current state. In particular, the user should be able to select the states currently associated with the probes on the display. Section 3.1 describes a simple schema for this involving two distinguished states: the *baseline* (B) and the *active* (A) state. Section 3.2 generalizes this idea to multiple distinguished states.

Probe functions do not need to be simple. The program GRAPHICAL-BELIEF allows the user to specify complex questions for a probe to answer. The Figure 3 shows a probe which answers the question "What is the probability that the value of the variable *Healthy?* will be no?". This probe can either be specified through a sequence of graphical dialogs or by a command.

Probes can also report more than a single numeric statistic. Pairs of values (upper and lower bounds) or predictions with error bars are simple generalizations. Another possibility is a more complex graphical display. When fitting linear models (a special case of the application described in section 1.2), it is common to plot the predicted value against the residuals (the difference between the predicted and observed values). A systematic pattern in this plot can indicate that the relationship is non-linear. This non-linearity can often be corrected by a transformation. Figure 4 shows the value of the residual plot before and after the transformation. Clearly displaying such plots side by side helps the analyst understand the effect of the transformation.

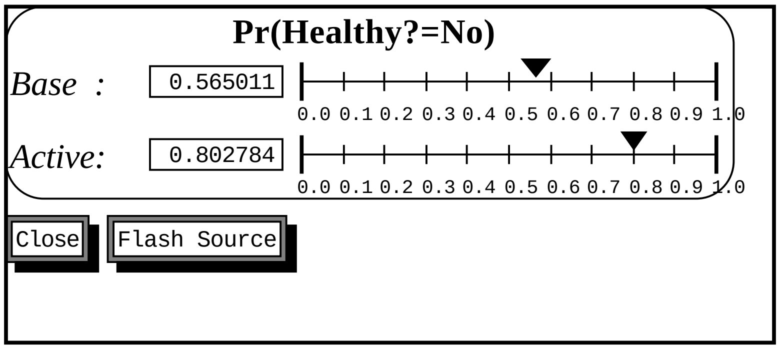
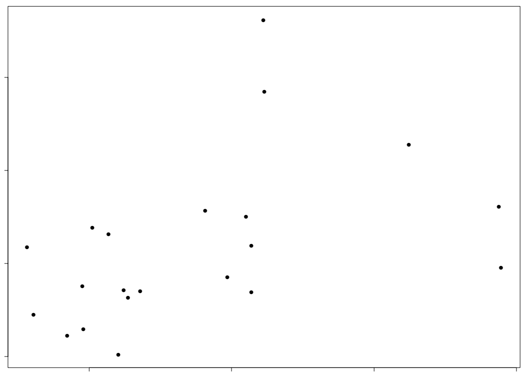


Figure 3. Probe for the probability of Healthy? = `no`
This probe assesses the overall health risk to the patient from coronary artery disease. The "Flash Source" button highlights the corresponding node in the original graph (the node Healthy? *in Figure 1). This probe indicates that the* `Asympotmatic` Chest-Pain *increases the patient's risk of coronary artery disease from .56 to .80.*



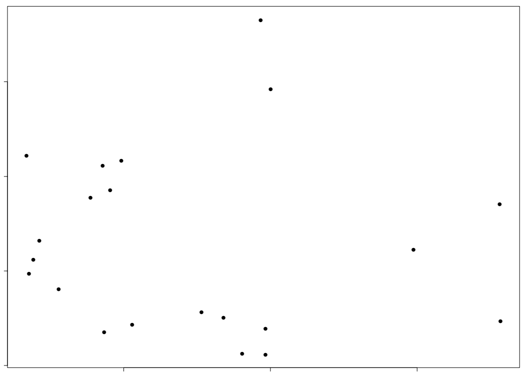


Figure 4a. Predicted vs Residual Before *Figure 4b. Predicted vs Residual After*
This example is taken from the analysis of the "Stack Loss" data in chapter 5 of Daniel and Wood[1980]. The first model (on the left) shows an increasing trend in absolute residual size as the fitted value increases, indicating that perhaps the log of the stack loss should be used as a response. The second model (on the right) incorporates that change. Note that both plots show large outliers, and potentially influential values, so neither model is adequate.

3.1 Baseline and Active States

In the both the examples described in section 1, many user tasks involve assessing the effect of a modification to the model. A *BA probe* shows two different states in the history stack: the *Baseline* (or *Before*) state labeled *Base* and the *Active* (or *After*) state labeled *Active*. The *probe* (figure 3) shows the statistics associated with these states and the *history browser* (figure 2) shows the history stack and any distinguished states. The history browser indicates the *active* state with reverse video and the *baseline* state with a dashed border (and italic font).

The user can change the baseline and active states in two different ways: (1) The *active* state automatically tracks the current state (all changes are made to the current state). Therefore, issuing any command which changes the model will cause the active state to revert to the new current state. The baseline state remains constant through these activities. (2) The user can select a different active or baseline state with the mouse (*brushing* history). (GRAPHICAL-BELIEF uses the left mouse button to select the active state and the middle mouse button to select the baseline state. As a baseline state should intuitively come before an active state, GRAPHICAL-BELIEF requests confirmation before allowing the user to set the baseline after the active state or the active state before the baseline.)

The baseline and active state model fits well with the kind of manipulations required in the two applications. In the decision support model (section 1.1), the analyst frequently want to answer questions of the kind "How valuable would knowing this piece of evidence be?" or "How much impact did this quanta of knowledge have on the conclusions?" To answer either of these questions, the analyst selects the current state as the baseline. The analyst then makes appropriate hypothetical changes (adding new evidence or backing old evidence out). The difference between the B and A probes provides the answer to the questions.

In the model fitting application (section 1.2), the analyst is trying to find the best fitting model according to a number of criteria. The baseline model is the best model found so far and the active model is the alternative currently under consideration. Comparing the B and A probes shows the differences between the models.

3.2 Multiple Reference Points

Although the *BA-prove* model is relatively powerful, it is easy to envision situations in which it is not sufficient. Unfortunately, it is not simple to generalize these situations. Two possibilities are using multiple baselines and using using multiple active states.

The principle difficulty with providing the user with more than one baseline, is providing the user with a way to distinguish the various baselines. Perhaps the easiest way is to allow the user to add a label to a state, making that state behave like the baseline. Associating the labels with colors is a simple mechanism for making the conceptual linkage between the history stack and the probe (the shape of the glyph used on the probe is another possibility). One possibility is to link the labeling operation with checkpointing or saving (section 4.2).

Note that the *BA-probe* confounds the notion of the active state and the current state. The system might be simpler to use if there was a distinct *current* state which always reflected the most recent state on the history stack. In some cases this extra state would be redundant, but in cases where the user is trying to scroll back through recent history, it may be helpful to compare to both the current and baseline states. The extra state would also prevent the user from confusing the current and active states; a danger with the *BA-probe* interface.

4.0 Operations on Histories

Although the primary purpose of the graphical display of the history stack (Figure 2) is to allow brushing the history stack, it should also help clarify meta-commands such as *undo* (section 4.1) and *save* (section 4.2).

4.1 Undo, Redo and related Meta-Commands

Archer, Conway and Schneider[1984], Vitter[1984], and Yang[1988] all introduce formal command history models in order to describe implementation choices for *undo* and related commands. In the command history model, one primary design question is whether the *undo* command undoes another *undo*.

This question simply does not arise in the state history stack model. The model for history is a series of states $S_0, S_1, \ldots, S_{n-1}, S_n$ (or in the user's mind a series of models $M_0, M_1, \ldots, M_{n-1}, M_n$). Undo simply pops the most recent state off the top of the stack. Thus after undo the model stack is $S_0, S_1, \ldots, S_{n-1}$. Multiple undos are also simple, in particular, popping all states after the active or baseline state fits well with the interaction paradigm already established for controlling the probes.

In both of the applications described in section 1, all of the changes made to the models are invertible. This provides another way of selectively undoing. This type of interaction is quite common in the model fitting application, where a variable which is selected early for its high explanatory power has much lower explanatory power after other variables are added to the model.

The Undo, Skip and Redo model (Vitter[1984]) suggests that as commands are undone, they be put into a *redo* buffer. The user can then *redo* a command or *skip* over a command the user does not want to redo. Thus if the current state is S_6 and the user reverts the state to S_3, the commands c_4, c_5 and c_6 would be put into the redo buffer. The user could then *skip* c_4 and *redo* c_5 and c_6, thus selectively undoing the change c_4. Yang[1988] extends this model by making the buffer circular, thus allowing the user to reuse the commands in an arbitrary pattern.

Redo and skip commands could be added to the graphical display of the history in a separate scrolling list. Thus the user could select to either undo an existing command, redo a command from the redo display or perform a new command. Note that the redo command buffer could keep more than just undone commands, it could keep a list of all commands thus making repeated commands simpler.

As Vitter[1984] points out, adding the *skip* and *redo* commands turns the linear history structure into a directed tree. In the example of the previous paragraph, S_3 would become a branch point, one child being the state S_4 (created by executing state c_4) and one being the state $S_{4'}$ (created by skipping c_4 and redoing c_5). Here the linear display of the history stack described by Figure 2, will no longer be adequate. Section 5 describes extending the linear history stack model to tree shaped histories.

4.2 Checkpointing and Saving

Typical users do not save their work randomly; instead they usually choose a time when the system is in some distinguished state (for example, they have just finished an important subtask). Saving at a distinguished state simplifies the resumption of their work after an interruption. The saved distinguished state also serves as a *checkpoint*, that is, if the system receives too many erroneous commands (either via user or computer error), the checkpointed state is a good restarting point. For example, the baseline state in Figure 2 represents information which comes from the patients chart; this is a natural checkpoint.

To facilitate this kind of interaction, the history stack mechanism could be augmented with a *checkpoint* meta-command. This command would simply label a state on the stack as a checkpoint, a point to which the user may want to return. Often the user would name this checkpoint. This checkpoint would often correspond to a state saved to disk or other long term memory. These checkpointed states could correspond to the labeled states of the multiple baseline model of section 3.2.

After checkpointing or saving the state, the user is often no longer interested in the states between the current and previous saved states. For example, in the history $S_0, S_1, \ldots, S_6$, let S_1 represent the previous saved state, and S_6 is the state that is currently being saved. The user may no longer be interested in the states $S_2, \ldots, S_5$. Colapsing the history discards the intervening states, leaving the simpler history stack S_0, S_1, S_6. Note that the "command" to get from S_1 to S_6 is now the script c_2, c_3, c_4, c_5, c_6.

Cowley and Whiting[1986] look at the issues of managing data analysis through save states more closely. In particular, they note that the analyst frequently returns to a checkpoint state and proceeds down another analysis path. The resulting history is inherently tree shaped, branching out in different directions from the saved state. The next section looks at the implications for history brushing.

5.0 Tree-Shaped Histories

Both the Undo, Skip and Redo model of Vitter[1984] and the checkpointing model of Cowley and Whiting[1986] produce a tree-shaped rather than linear model of history. This suggests replacing the model history stack with a model history tree (Figure 5). The user would select the baseline and active states by selecting nodes in the tree. Commands which change the model state would create a difference from the active state. Thus changing the active state would be an effective *undo*, although because the system captures the tree shaped history, all *undo*s would be temporary. (To compensate for making *undo* temporary, the system would require a new *delete-state* meta-command for getting rid of states which represent mistakes or irrelevant information).

This tree shaped model manipulation corresponds to the typical behavior patterns of analysts in both of the example applications. In particular, in the model selection application (section 1.2), the analyst will look

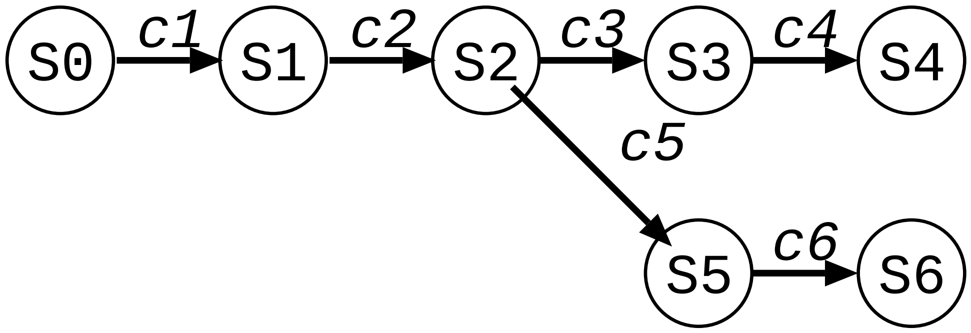


Figure 5. Sample History Tree

This non-linear history shows both states and commands. It could form the basis of an interface to a non-linear history model.

at a large number of changes from a given state and then select the best one to proceed along. The analyst also often wants to compare leaves of the history tree to find the best model obtained along two different search paths. Documenting the paths to the best model (including the dead ends) can be an important part of the model selection process.

Producing a state tree is more difficult than producing a history tree (in the style of Vitter[1984]). In particular, the state tree can reconverge producing a state dag. For example, if operations commute then first adding X then Y to a model should be equivalent to first adding Y and then X. Ideally, the system should recognize those points of reconvergence and form them into a common node in the graph. This will make the history a directed acyclic graph (Figure 6).

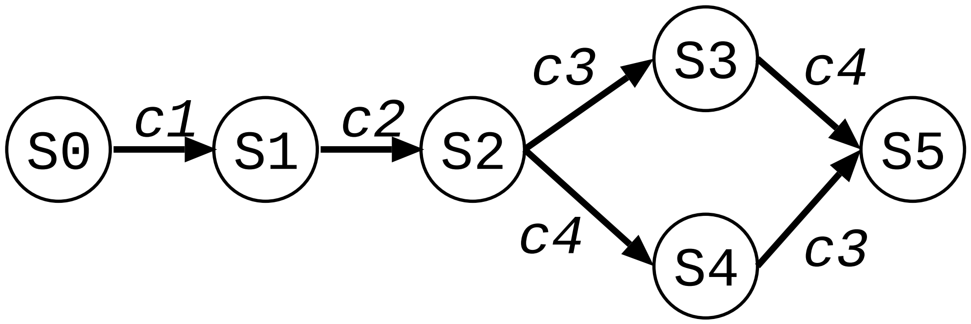


Figure 6. Reconvergence of states

If the commands c_3 and c_4 are commutative, then the two scripts c_1, c_2, c_3, c_4 and c_1, c_2, c_4, c_3 are identical and should both lead to the same state S_5. The user may want to compare state S_5 to both S_3 and S_4; that would be a less intuitive operation in either a linear or tree shaped model.

6.0 Conclusions

In both of the applications discussed in section 1, the challenge is to raise the level of dialog of the analyst with the computer. Without the history mechanism there is little support for comparisons between models, much of the dialog would be commands to change the model alternating with commands to display statistics about the current model. This forces the user to focus on the level of mechanical manipulation of the mode. With the state history mechanism, the user can directly compare the models. This allows the user to focus on the conceptual difference between the models.

The second important feature of the history brushing mechanism is that it is interactive. In scatterplot matrix brushing, the interactivity has increased the complexity of the tasks the analyst can do, enabling the analyst to uncover complex dependencies. Brushing histories should similarly increase the complexity of the model comparisons an analyst can perform. Furthermore, brushing interactions with scatterplots are fun, encouraging the analyst to spend more time exploring the data; brushing histories should similarly encourage the analyst to spend more time exploring possible models.

The ideas in this paper are based on the design of the user interface for the GRAPHICAL-BELIEF package, as well as extending these ideas to cover the model fitting domain. Prototypes for some of the concepts were implemented in Garnet (Myers *et al.*[1992]), in particular Figures 1, 2 and 3 are screen-dumps from Garnet gadgets. Preliminary experience with the GRAPHICAL-BELIEF history mechanism suggests that the history stack really does allow the user to perform higher level model comparison tasks. This in turn, has made GRAPHICAL-BELIEF much easier to use.

7.0 References

Almond, Russell G.[1992]. "GRAPHICAL-BELIEF: Project Overview and Review" Statistical Science Research Report 14. StatSci, 1700 Westlake Ave, N., Suite 500, Seattle, WA 98109.

Almond, Russell G. [1993]. *Graphical Belief Models: Algorithms and Examples.* To be published as a monograph from Van Nostrand Reinhold. (Previous version was Ph.D. dissertation and Harvard University, Department of Statistics Technical Report S-130.)

Archer, Jr. J. E., Conway, R. and Schneider, F.B.[1984]. "User recovery and reversal in interactive systems." *ACM Transactions on Programming Languages and Systems,* (**6**), 1–19.

Becker, Richard A. and Chambers, John M.[1987]. "Brushing Scatterplots." *Technometrics,* **29**, 127-142.

Breslow, N. [1991]. "Biostatistics and Bayes," *Statistical Science,* (**5**), 269–298.

Chambers, John M. and Hastie, Trevor J. [1992]. *Statistical Models in* S. Wadsworth & Brooks / Cole, Pacific Grove, CA.

Cowley, Paula J. and Whiting, Mark A. [1986]. "Managing Data Analysis through Save-States." *Computer Science and Statistics: 17th Symposisum on the Interface* Allen, D.M. (ed.), Elservier Science Publishers, pp 121–127.

Daniel, Cuthbert and Wood, Fred S. [1980]. *Fitting Equations to Data: Computer Analysis of Multifactor Data,* 2nd Edition, John Wiley and Sons.

Robert Detrano, Andras Janosi, Walter Steinbrunn, Matthias Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern H. Guppy, Stella Lee, and Victor Froelicher [1989]. "International Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease." *American Jouran of Cardiology,* (**64**) 304–310.

Healy, M.J.R. [1988]. GLIM*: An Introduction.* Claredon Press, Oxford.

Madigan, D., Raftery, A. E., York, J. C., Bradshaw, J. M. and Almond, R. G. [1993]. "Strategies for Graphical Model Selection." Poster presented at *4th International Workshop of Artificial Intelligence and Statistics,* Ft. Lauderdale, FL. Paper to appear in proceedings.

Murphy, P.M. and Aha, D.W.[1992]. *UCI Repository of Machine Learning Databases.* Online database maintained at the Department of Information and Computer Science, University of California, Irvine, CA.

Myers, Brad A., Guise, Dario A., Dannenberg, Roger B., Vander Zanden, Brad, Marchal, Phillipe, Pevin Ed, Mickish, Andrew, Landay, James A., McDaniel, Richard, and Gupta, Vivek [1992]. "Garnet: The Garnet Reference Manuals, Revised for Version 2.1" Carnegie Mellon, School of Computer Science, Technical Report, CMU-CS-90-117-R3.

Vitter, Jeffrey S.[1984]. "US&R: A New Framework for Redoing." *IEEE Software,* (**1**), 39–52.

Wilkinson, G. N. and Rogers, C. E. [1973]., "Symbolic Description of Factorial Models for Analysis of Variance." *Applied Statistics,*, **22**, 392–399.

Yang, Yita [1988]. "Undo Support Models." *International Journal of Man–Machine Studies,* (**28**), 457–481.