Constrained Functionality:

Application of the ACT-R Cognitive Architecture to the AMBR Modeling Comparison

Christian Lebiere

The ACT-R Cognitive Architecture

Symbolic Level

ACT-R is a production system theory that models the steps of cognition by a sequence of production rules that fire to coordinate retrieval of information from the environment and from memory. It is a cognitive architecture that can be used to model a wide range of human cognition. It has been used to model tasks from memory retrieval (Anderson, Bothell, Lebiere & Matessa, 1998) to visual search (Anderson, Matessa & Lebiere, 1997). The range of models developed, from those purely concerned with internal cognition to those focused on perception and action, makes ACT-R a plausible candidate to model a task like the air traffic control simulation described previously because the task includes all of these various components. In all domains, ACT-R is distinguished by the detail and fidelity with which it models human cognition. It makes claims about what

happens cognitively every few hundred milliseconds in performance of a task. ACT-R is situated at a level of aggregation considerably above basic brain processes but considerably below significant tasks like air-traffic control. The new version of the theory has been designed to be more relevant to tasks which require deploying significant bodies of knowledge under conditions of time pressure and high information-processing demand. This is because of the increased concern with the temporal structure of cognition and with the coordination of perception, cognition, and action.

Figure 1 displays the information flow in the ACT-R 4.0 architecture (Anderson & Lebiere, 1998). There are essentially three memories -- a **goal stack** that encodes the hierarchy of intentions guiding behavior, a **procedural memory** containing production rules, and a **declarative memory** containing chunks. These are all organized through the **current goal** that represents the focus of attention. The current goal can be temporarily suspended when a new goal is **push**ed on the stack. The current goal can be **pop**ped in which case the next goal will be retrieved from the stack. Productions are selected to fire through a **conflict resolution** process that chooses one production from among the productions that match the current goal. The selected production can cause **action**s to be taken in the outside world, can **transform** the current **goal** (possibly resulting in pushes and pops to the stack), and can make **retrieval request**s of declarative memory (such as what is the sum of 3 and 4?). The **retrieval result** (such as 7) can be returned to the goal. The arrows in Figure 1.1 also describe how new declarative chunks and productions are acquired. Chunks can be added to declarative memory either as **popped goal**s reflecting the solutions to past problems or as **perception**s from the environment. Productions are

created from declarative chunks called dependencies through a process called **production compilation** which takes an encoding of an execution trace resulting from multiple production firings and produces a new production that implements a generalization of that transformation in a single production cycle.

<Insert Figure 1 here>

Subsymbolic Level

ACT-R can be described as a purely symbolic system in which discrete chunks and productions interact in discrete cycles. However, ACT-R also has a subsymbolic level in which continuously varying quantities are processed, often in parallel, to produce much of the qualitative structure of human cognition. These subsymbolic quantities participate in neural-like activation processes that determine the speed and success of access to chunks in declarative memory as well as the conflict resolution among production rules. ACT-R also has a set of learning processes that can modify these subsymbolic quantities. Formally, activation reflects the log posterior odds that a chunk is relevant in a particular situation. The activation A_i of a chunk i is computed as the sum of its *base-level activation* B_i plus its *context activation*:

$$A_i = B_i + \sum_j W_j S_{ji} \qquad \textbf{Activation Equation}$$

In determining the context activation, W_j designates the attentional weight given the focus element *j*. An element *j* is in the focus, or in context, if it is part of the current goal chunk (i.e. the value of one of the goal chunk's slots). S_{ji} stands for the strength of association from element *j* to a chunk *i*. ACT-R assumes that there is a limited capacity of source activation and that each goal element emits an equal amount of activation. Source activation capacity is typically assumed to be *1*, i.e. if there are *n* source elements in the current focus each receives a source activation of *1/n*. The *associative strength* S_{ji} between an activation source *j* and a chunk *i* is a measure of how often *i* was needed (i.e. retrieved in a production) when chunk *j* was in the context. Associative strengths provide an estimate of the log likelihood ratio measure of how much the presence of a cue *j* in a goal slot increases the probability that a particular chunk *i* is needed for retrieval to instantiate a production. The base level activation of a chunk is learned by an architectural mechanism to reflect the past history of use of a chunk *i*:

$$B_i = \ln \sum_{j=1}^{n} t_j^{-d} \approx \ln \frac{nL^{-d}}{1-d} \qquad \textbf{Base-Level Learning Equation}$$

In the above formula t_j stands for the time elapsed since the *jth* reference to chunk *i* while *d* is the memory decay rate and *L* denotes the life time of a chunk (i.e. the time since its creation). As Anderson and Schooler (1991) have shown, this equation produces the Power Law of Forgetting (Rubin & Wenzel, 1996) as well as the Power Law of Learning (Newell & Rosenbloom, 1981). When retrieving a chunk to instantiate a production, ACT-R selects the chunk with the highest activation A_i. However, some stochasticity is

introduced in the system by adding gaussian noise of mean 0 and standard deviation σ to the activation A_i of each chunk. In order to be retrieved, the activation of a chunk needs to reach a fixed retrieval threshold τ that limits the accessibility of declarative elements. If the gaussian noise is approximated with a sigmoid distribution, the probability P of chunk i to be retrieved by a production is:

$$P = \frac{1}{1 + e^{-\frac{A_i - \tau}{s}}}$$ **Retrieval Probability Equation**

where $s = \sqrt{3}\sigma/\pi$. The activation of a chunk i is directly related to the latency of its retrieval by a production p. Formally, retrieval time T_{ip} is an exponentially decreasing function of the chunk's activation A_i:

$$T_{ip} = Fe^{-fA_i}$$ **Retrieval Time Equation**

where F is a time scaling factor. In addition to the latencies for chunk retrieval as given by the *Retrieval Time Equation*, the total time of selecting and applying a production is determined by executing the actions of a production's action part, whereby a value of 50 ms is typically assumed for elementary internal actions. External actions, such as pressing a key, usually have a longer latency determined by the ACT-R/PM perceptual-motor module (Byrne & Anderson 1998). In summary, subsymbolic activation processes in ACT-R make a chunk active to the degree that past experience and the present context (as given by the current goal) indicates that it is useful at this particular moment.

Just as subsymbolic activation processes control which chunk is retrieved from declarative memory, the process of selecting which production to fire at each cycle, known as conflict resolution, is also determined by subsymbolic quantities called utility that are associated with each production. The utility, or expected gain, E of a production is defined as:

$$E = P \bullet G - C$$ **Expected Gain Equation**

where G is the value of the goal to which the production applies, and P and C are estimates of the goal's probability of being successfully completed and the expected cost in time until that completion, respectively, after this production fires. Just as for retrieval, conflict resolution is a stochastic process through the injection of noise in each production's utility, leading to a probability of selecting a production i given by:

$$p(i) = \frac{e^{\frac{E_i}{t}}}{\sum_j e^{\frac{Ej}{t}}}$$ **Conflict Resolution Equation**

where $t=\sqrt{6}\sigma/\pi$. Just as for the base-level activation, a production's probability of success and cost are learned to reflect the past history of use of that production,

specifically the past number of times that that production lead to success or failure of the goal to which it applied, and the subsequent cost that resulted, as specified by:

$$P = \frac{Successes}{Successes + Failures}$$ **Probability Learning Equation**

$$C = \frac{\sum Costs}{Successes + Failures}$$ **Cost Learning Equation**

Costs are defined in terms of the time to lead to a resolution of the current goal. Thus the more/less successful a production is in leading to a solution to the goal and the more/less efficient that solution is, the more/less likely that production is to be selected in the future.

Experiment I Model

Modeling Methodology

If it is to justify its structural costs, a cognitive architecture should facilitate the development of a model in several ways. It should limit the space of possible models to those that can be expressed concisely in its language and work well with its built-in mechanisms. It should provide for significant transfer from models of similar tasks, either directly in the form of code or more generally in the form of design patterns and techniques. Finally, it should provide learning mechanisms that allow the modeler to only specify in the model the structure of the task and let the architecture learn the details of the task in the same way that human cognition constantly adapts to the structure of its environment. These architectural advantages not only reduce the amount of knowledge engineering required and the number of trial-and-error development cycles, providing significant savings in time and labor, but also improve the predictiveness of the final model. If the "natural" model derived a priori from the structure of the task, the constraints of the architecture and the guidelines from previous models of related tasks provide a good fit of the empirical data, one can be more confident that it will generalize to unforeseen scenarios and circumstances than if it is the result of post hoc knowledge engineering and data analysis. That is the approach that we have adopted in developing a model of this task, and indeed more generally in our use of the ACT-R architecture.

When faced with developing a model of this task, we did not try to reverse-engineer from their data and protocols which techniques and strategies subjects used when confronted with the task, but instead we asked ourselves which ACT-R model would best solve the task given the architectural constraints. An additional emphasis in developing the model was on simplicity, both because of the time constraints provided by the fly-off and because since the subjects had only had a limited amount of practice with the task it was fairly unlikely that they had developed highly elaborate strategies. Generally, for each phase, the total development time, including the time-consuming process of finding the best way to interface with the simulation, was less than 6 weeks, and the time to develop the model itself was less than a week. A more time-consuming part of the process is the repeated tweaking of the model (both in terms of real-valued parameters as well as symbolic knowledge structures) to attempt to improve the fit to the data. This practice, however widespread, can take arbitrarily large amounts of time and often results in very little meaningful improvements to the model. Our experience here confirmed that it would best be left to a minimum if tolerated at all. Indeed, from our perspective this project illustrated quite nicely the dual advantage of cognitive architectures. Because they provide considerable constraints upon the mechanisms and parameters to be used for building human performance models, they limit the degrees of freedom where other, non-first-principled methods have to resort to parameter-fitting and further validation. Moreover, because of those constraints and the leverage of built-in mechanisms, the development of the model is much more efficient, making human performance models more affordable for their many potential applications.

One common design pattern in ACT-R models of similar tasks (e.g. Lee & Anderson 2001) is the concept of unit task (Card, Moran & Newell, 1983). Unit tasks correspond to subtasks of more complex tasks that are associated with a specific goal in a given context. That decomposition has been shown to have significant psychological validity in the prediction of subject performance (Corbett, Anderson & O'Brien, 1995). Unit tasks further the goal of simplicity because they provide a way to decompose a model of a complex task into independent sets of productions applying in specific situations. Moreover, unit tasks correspond directly to the concept of goal type in ACT-R, with each goal of that type corresponding to a specific instance of that unit task and productions that match that goal type corresponding to the knowledge required to solve that unit task. The decomposition of ACT-R models is similar to the software engineering concept of object-oriented programming, with classes corresponding to goal types, instances of those classes corresponding to chunks (goals) of that type, and methods applying to objects of that class corresponding to productions that apply to goals of that type.

Of course unit task decomposition is not merely a software engineering principle for developing cognitive models, but rather it corresponds to an underlying psychological reality as well. The unit tasks for this simulation are fairly clearly identified. In both the aided and un-aided conditions, processing an aircraft that requires action by the central controller is a clearly defined unit task. In the color (aided) condition, scanning the radar screen for an aircraft that turned color, identifying its need for action, is another unit task. Similarly, in the text (un-aided) condition, the subtasks of scanning a single text window or radar screen area constitute unit tasks as well. Finally, in the text condition, selecting

the next part of the screen to scan is the top-level unit task. Those five unit tasks define the structure of the ACT-R model and the procedural knowledge required to solve them will be described in detail in the following subsections.

Another design pattern that appears in countless ACT-R models (e.g. cognitive arithmetic, alphabet arithmetic, instance-based problem solving, etc) deals with the trade-off between trying to retrieve an answer from memory, which tends to be fastest but most error-prone, and attempting to re-derive it using backup methods such as computation or perceptual scanning. In this simulation, this problem appears in many instances, such as identifying the position of an aircraft from its identifier when scanning a text window, or deciding whether an aircraft has been processed when scanning it on the radar screen. Both of these questions could be answered[1] either by attempting to retrieve a related memory (respectively of scanning or processing that aircraft) or by searching the proper screen area (respectively the radar screen or a text window) for the information. While ACT-R provides the capacity for the model to decide between each course of action based on their expected cost (in terms of time to perform the action) and probability of success (in providing the needed information), this requires learning from experience with the system which wasn't the focus of the phase I modeling effort (but was highlighted in the phase II model as will be seen in a later section). Instead, as is often

[1] There might be cases when the information is not present on the screen, such as when a text message pertaining to an aircraft has scrolled off the top of the window or when an aircraft mentioned in a message has exited the radar screen, but the display changes slowly enough that those cases are relatively rare.

the case, retrieval from memory is preferred over explicit scanning because of its relatively low cost. Only if that retrieval fails, either because the chunk encoding the information wasn't present in memory or because its activation had decayed below the retrieval threshold, will the strategy of explicit scanning be selected. This pattern of attempting to retrieve information from memory and only when it fails is a pervasive one in ACT-R models, and one that is transparently supported by the architecture. As the information is re-created from the environment or explicit computation, the activation of the chunk encoding it will gradually rise with practice until it can retrieved directly. This process of transition from explicit methods to a reliance on memory is a pervasive aspect of human cognition that ACT-R can account for in a direct, straightforward manner through its activation calculus.

Finally, a key aspect of our methodology that is also pervasive in ACT-R modeling (Anderson & Lebiere, 1998) is the use of Monte Carlo simulations to reproduce not only the aggregate subject data such as the mean performance or response time but also the variation that is a fundamental part of human cognition. In that view, the model doesn't represent an ideal or even average subject but instead each model run is meant to be equivalent to a subject run, in all its variability and unpredictiveness. For that to happen, it is essential that the model not be merely a deterministic symbolic system but be able to exhibit meaningful non-determinism. To that end, randomness is incorporated in every part of ACT-R's subsymbolic level, including chunk activations which control their probability and latency of retrieval, production utilities which control their probability of selections, and production efforts which control the time that they spent executing.

Moreover, as has been found in other ACT-R models (e.g. Lebiere & West, 1999; Gonzalez, Lebiere & Lerch, 2003), that randomness is amplified in the interaction of the model with a dynamic environment: even small differences in the timing of execution might mean missing a critical deadline, which results in an airplane going on hold (with the resulting 50-point penalty), which requires immediate attention, which might cause another missed deadline and so on. The magnitude of the sensitivity to random fluctuations was brought to our attention when an early, noise-free version of the model was run in real-time against the simulation. Even though both the model and the simulation were deterministic and the only source of randomness was small sub-second variations in synchronization between the two systems, performance varied by as much as 100 points in the same condition.

To model the variations as well as the mean of subject performance, the model was always run as many times as there were subject runs. For that to be a practical strategy of model development, it is essential that the model run very fast, ideally significantly faster than real-time. Our model ran up to 5 times real-time, with the speed limitation being due entirely to the communication bottleneck between model and simulation rather than the computational requirements of ACT-R, which can run at several hundred times real-time. This speed made it possible to run a full batch of 48 scenarios in about an hour and a half, enabling a relatively quick cycle of model development. One source of variation in subject performance that we could not exploit is individual differences. ACT-R has been able to provide a fine-grained account of individual differences in working memory performance through continuous variations in the value of the architectural parameter *W*

controlling spreading activation (Lovett, Reder & Lebiere, 1997). An obvious way to account for individual differences in this task would be to include variations of the effort production parameter controlling the speed of execution of the model, in a manner consistent with the slow man-fast man distinction of Card, Moran and Newell (1983). However, for the sake of simplicity and the avoidance of arbitrary degrees of freedom, we left the parameters unchanged. Generally, all parameters controlling the model were left at their default values, established either by the architecture or by existing models. The rate of base-level decay *d* was 0.5 and the level of activation noise was 0.25, both of which have been consistently used in many ACT-R models. The retrieval threshold was 0.0 and the latency factor was 1.0, both values being architectural defaults. The effort for productions that do not involve any perceptual or motor actions was left at the architectural default of 50msec. Two parameters were roughly estimated: the effort for perceptual productions was set at 500 msec and the effort for productions involving actions (typically move the mouse to a target and click) was set at 1sec.

Model

Six chunk types are defined using the **chunk-type** command. They consist of the name of the chunk type and the associated slots. One chunk type, **rule**, holds the basic content of the instructions to define each category of event by relating a specific action (e.g. contact) to the color used for that event in the aided condition (e.g. yellow) and the amount of penalty points associated with failing to act on it in a timely fashion (e.g. 50). Five chunks of that type are defined using the **add-dm** command to encode that

information for all five event types. The initial base-level activation of those chunks is set by the command **sdp** to reflect the level of practice at that point in the simulation, i.e. 120 references over the last hour (3600 seconds), to reflect the instruction study phase as well as the first block of practice. These history parameters will determine the activation of those chunks according to the base-level learning equation, which in turn will determine how fast and how reliably they can be retrieved. These chunks will only be used in the aided (color-coded) condition to map color of aircraft to required action. All other chunks defined by the **add-dm** command are simply symbols used in other chunks (which the system would define by default) and the initial goals for the color and text condition. The model tests the type of scenario obtained from the simulation to decide which of these two chunks to set as the initial goal. The other five chunk types that are defined correspond to the goals used for the five unit tasks that compose this task. Those goal types are **color-goal**, **text-goal**, **scan-text**, **scan-screen** and **process**. They and their associated procedural knowledge will be described in detail in the rest of this section. The productions that apply to each goal type will be listed in a table using an informal English description that is meant to capture their function without obscure syntactic details. Production names are in bold while words in italics correspond to production variables and words in bold within the production text correspond to specific chunks (constants). The order in which the productions are listed correspond to their order of priority in the conflict resolution process, with the earlier productions being favored and the later productions only allowed to fire if the preceding ones cannot match.

Table 2.1 presents the productions for the top-level unit task **color-goal**. The production **color-target-detection** detects a colored aircraft on the radar screen and notes the aircraft identity in the goal. The production **color-target-acquisition** notes the aircraft color in the goal and the production **color-action** retrieves from memory the chunk linking that color to the required action and pushes a subgoal to process that action on the aircraft. If none of these productions can apply, the production **wait** fires, essentially filling the 50msec of this production cycle before the next cycle of detection can take place. The productions above would provide a perfectly functional treatment of the **color-goal** unit task, but there is one additional production called **subgoal-next** which has to do with onset detection. If an aircraft turns color while a subgoal to process another aircraft is the current goal, the model will detect that aircraft while it processes the subgoal and create a prospective subgoal to process that new aircraft. That subgoal will be returned to the parent **color-goal** when the process subgoal is completed and the production **subgoal-next** will immediately push it without having to fire the productions to detect the aircraft, map its color to the action and create a new subgoal. This treatment is consistent with subject awareness of event onset and predicts the right slope for the time to handle an aircraft as a function of intervening events.

Subgoal-next

IF the *goal* is of type color-goal

and a *subgoal* to process a colored aircraft was formulated previously

THEN push that *subgoal*

Color-target-detection

IF the *goal* is of type color-goal
and a colored *aircraft* is present on the screen
THEN note that *aircraft*
Color-target-acquisition
IF the *goal* is of type color-goal and a colored *aircraft* has been detected
THEN note its *color*
Color-action
IF the *goal* is of type color-goal and an *aircraft* and its *color* have been identified
and a *rule* chunk can be retrieved linking *color* to *action*
THEN push *subgoal* to process *action* on *aircraft* at current *position*
Wait
IF the goal is of type color-goal
THEN do nothing

Table 2.1: Productions applicable to the unit task **color-goal**

Table 2.2 presents the productions for the top-level unit task **text-goal**. The text condition is more complex than the color condition because relevant events are much harder to detect and that is reflected in its unit task structure. Unlike **color-goal**, the **text-goal** unit task does not directly detect aircraft that require action and subgoal any process goal but instead directs attention to specific areas of the screen in which to perform that detection. There are four screen areas to be scanned, the three text message windows, **left** for incoming aircraft, **right** for exiting aircraft and **low** for speed changes, which are scanned by the unit task **scan-text**, and the radar screen area **between** the green and

yellow lines for exiting aircraft, which are scanned by the unit task **scan-screen**. The latter is necessary because the central controller has to initiate the transfer of exiting aircraft to other controllers, whereas all other actions are taken in response to a text message. There are four productions that implement a sequential scan of the four areas by pushing a subgoal to scan each area given the previous one. This solution was chosen for its simplicity and systematicity[2], but other are possible such as a random scan or a scan based on the probabilities of finding a new event in each of the four areas, a strategy that might be optimal and for which ACT-R's utility learning mechanism would be well suited. However, the data available was inconclusive on that aspect of subject behavior and finer-grained data such as eye movements would be needed to precisely determine subjects' strategies in that regard. Note that this systematic scan only happens when no event onset was detected in another window when scanning the present window.

Between-left

IF the *goal* is of type text-goal and the last area scanned was **between**

THEN push a subgoal to scan the text area **left** starting at the **bottom**

Left-right

IF the *goal* is of type text-goal and the last area scanned was **left**

THEN push a subgoal to scan the text area **right** starting at the **bottom**

Right-low

IF the *goal* is of type text-goal and the last area scanned was **right**

[2] Indeed, the author significantly improved his personal performance by adopting that method.

THEN push a subgoal to scan the text area **low** starting at the **bottom**
Low-between
IF the *goal* is of type text-goal and the last area scanned was **low**
THEN push a subgoal to scan the screen area **between**

Table 2.2: Productions applicable to the unit task **text-goal**

Table 2.3 presents the productions for the unit task **scan-text** responsible for scanning a text window. As initialized by the **text-goal** productions described previously, **scan-text** goals start scanning at the bottom of the screen. This is contrary to the usual top-down scanning pattern, but new messages appear at the bottom of the screen and it is therefore the best place to look for them. Subjects probably took some time to learn this scanning pattern but they are expected by that time in the simulation to have adopted the more efficient strategy. Again, more detailed data such as eye movements, and data from earlier trials would be needed to conclusively answer the questions regarding the subjects scanning strategies. The production **find-flush-message** scans upward from the current position (initially **bottom**) to find the next message that is flush against the left side of the window, indicating a message from an aircraft or another controller requesting action. If no such message can be found, the production **no-flush-message** pops the goal, which returns control to the **text-goal** unit-task. If a message is found requesting action, the model then tries to determine whether that action has already been completed. The production **memory-for-message** searches declarative memory for a chunk recording the completion of a process goal for the task and aircraft indicated by the message. Recall

that when completed goals are popped they become permanent declarative memory chunks that can be retrieved later. However, retrieval of a chunk is subject to its activation reaching threshold, and failure to retrieving a trace of past execution is no guarantee that it didn't happen. Therefore if memory retrieval fails the production **message-reply** will scan down the text window from the current message for an indented message containing the acknowledgment message that would have resulted from taking that action. If either a memory or a message indicating completion of the action is found, the goal is popped, because under the bottom up scanning strategy finding a message that had been attended to suggests that no unattended message older than the current message will be found. As we will discuss shortly, this is not an ironclad guarantee and it may be a natural source of skipped messages that result in violations. If no indication that the action requested by the message has taken place, the production **subgoal-message-task** pushes a subgoal to perform that action and clears the goal to allow further scanning to take place when that unit task is completed. Note that the strategy of first trying to retrieve a piece of information from memory and then resorting to an explicit strategy to reconstruct that information if the retrieval fails is a very general design pattern in ACT-R (e.g. Lebiere, 1998) that is very naturally supported by the architecture's conflict resolution mechanism. Since memory retrieval usually takes much less effort than implementing a complicated strategy, the utility learning mechanism will tend to assign a higher priority to the retrieval strategy which will then be attempted first. Again, since learning was not the focus of this model and training data was not available, this learning mechanism was not activated and instead the production ordering was relied upon to indicate priority.

Again, this production set would provide a perfectly adequate implementation of the **scan-text** unit task. But it would result in a very systematic pattern of execution by exhaustively scanning a text window to find and process all unattended events then move on to the next area and so on. While it might result in the right aggregate performance (and indeed did in the first version of the model, as will be elaborated in the discussion section), its deliberate character would prevent it to display the subjects ability to promptly respond to a new event as indicated by the sharply decreasing curve of number and average time of responses as a function of intervening events. The model needs to be able to focus on newly occurring events. The production **detect-onset-text** provides that capacity by detecting the onset of a new message in other text windows and record in the current goal to focus attention to that window as soon as the current message has been processed. The production **focus-onset-text** accomplishes that by focusing on a new goal to scan the text window in which the new message has appeared. This onset detection mechanism has a number of interesting attributes. First of all, the ability to detect the onset of a new event is very time-limited (fixed at 1 second in our model). While the onset detection productions have the highest priority in their unit task, if the model is otherwise busy during that limited time window (such as by an event-processing subgoal), it might miss the event onset and fail to record it. Second, only one event onset can be stored in the current goal and subsequent ones will not be recorded. An alternative would be to have the most recent onset overwrite the older ones, but again finer-grained data would be needed to shed light on that question. Third, the new goal to scan the text window in which the event onset appeared replaces the current goal rather

than being a subgoal. This is consistent with viewing the goal stack as a limited memory and not relying on it to provide a perfect memory of past situations. However, it also means that messages further up in the current window might not be processed because of the distraction of shifting to a new text window, constituting a very natural source of errors. Indeed, it suggests a rational analysis (Anderson, 1990) of onset detections: while they provide the ability to opportunistically respond to newly occurring events and emergency situations, they distract from the task at hand and might be detrimental to its performance. An onset detection mechanism along the lines of the one described here was subsequently added to ACT-R/PM (Byrne & Anderson, 2001), but much remains to be done to determine the proper treatment of onset detection in an integrated architecture such as ACT-R.

Detect-onset-text

IF the *goal* is of type scan-text and the area scanned is *window*

and a message onset is detected in area *next* which is not *window*

THEN make a note to scan text area *next*

Focus-onset-text

IF the *goal* is of type scan-text, no aircraft is selected and onset was detected in area *next*

THEN focus on a *subgoal* to scan text area *next* starting at **bottom**

Find-flush-message

IF the *goal* is of type scan-text of area *window* and no aircraft is currently selected

and *message* is the next flush message in *window* going up from current position

THEN note the *task*, *aircraft* and *controller* in *message*

No-flush-message

IF the *goal* is of type scan-text and no aircraft is currently selected

THEN pop the current goal

Memory-for-message

IF the *goal* is of type scan-text with current task *task* and aircraft *aircraft*

and there is a *chunk* for processing task *task* on aircraft *aircraft*

THEN pop *goal*

Message-reply

IF the *goal* is of type scan-text of area *window* with current aircraft *aircraft*

and *message* is the next indented message in *window* containg *aircraft* going down

from current position

THEN pop *goal*

Subgoal-message-task

IF the *goal* is of type scan-text with task *task*, aircraft *aircraft* and controller *controller*

THEN clear *goal* and

push *subgoal* to process task *task* on aircraft *aircraft* with controller *controller*

Table 2.3: Productions applicable to the unit task **scan-text**

Table 2.4 presents the productions for the unit task **scan-screen** responsible for scanning the radar screen, more specifically the area between the green and yellow lines in which exiting aircraft that need to be transferred can be detected. Because of the similarity between the two unit tasks, both of which consists in scanning a screen area to detect events that require actions, the set of productions for the unit task **scan-screen** is quite

similar to those for the unit task **scan-text**. **Scan-for-transfer** scans the radar area for exiting aircraft, **memory-for-transfer** and **trace-of-transfer** search in declarative memory and the top **right** text window respectively if the aircraft has already been transferred. If not, **subgoal-transfer** pushes a subgoal to transfer the aircraft. If no more exiting aircraft can be detected, **scan-done** pops the goal. The message onset detection productions **detect-onset-screen** and **focus-onset-screen** are similar to their counterpart in unit task **scan-text**.

Detect-onset-screen

IF the *goal* is of type scan-screen and no onset has been detected

and a message onset is detected in area *next*

THEN make a note to scan text area *next*

Focus-onset-screen

IF the *goal* is of type scan-screen, no aircraft selected and onset was detected in area *next*

THEN focus on a *subgoal* to scan text area *next* starting at **bottom**

Detect-red

IF the *goal* is of type scan-screen and no aircraft is selected

and a colored *aircraft* is present

THEN note *position* and *controller* associated with *aircraft*

and push subgoals to process both tasks for *aircraft* in *position* with *controller*

Scan-for-transfer

IF the *goal* is of type scan-screen and no aircraft is currently selected

and *aircraft* is outgoing in the **between** area

THEN note *aircraft* with its *position* and associated *controller*

Scan-done

IF the *goal* is of type scan-screen and no aircraft is currently selected

THEN pop *goal*

Memory-for-transfer

IF the *goal* is of type scan-screen with current *aircraft* and *controller*

and there is a *chunk* for processing *aircraft* with *controller*

THEN clear the *goal*

Trace-of-transfer

IF the *goal* is of type scan-screen with current *aircraft* and *controller*

and there is an indented message for *aircraft* in text area **right**

THEN clear the *goal*

Subgoal-transfer

IF the *goal* is of type scan-screen with current *aircraft* in *position* with *controller*

THEN clear *goal*

and push *subgoal* to process **transfer** on *aircraft* in *position* with *controller*

Table 2.4: Productions applicable to the unit task **scan-screen**

There are noteworthy differences as well. First, the model does not assume any specific scan ordering but instead detects exiting aircraft in an arbitrary order. Unlike the text window scanning for which a natural ordering existed, any order in scanning the **between** radar screen area (e.g. clockwise, counterclockwise, starting at any corner, etc) seems equally valid and indeed subject protocols indicating widely different strategies. Again, more precise data such as eye movements would be needed to determine the answer.

Second, the model avoided focusing on the same aircraft twice in the same unit task by using an *attended* feature similar to the one existing in ACT-R/PM. This application actually suggests a possible answer to a longstanding question in ACT-R/PM regarding the duration of the attended feature tag: unit tasks provide natural boundaries to reset attended tags. Third, there is an additional detection production **detect-red** that detects a red aircraft indicating holding violation in a manner similar to the color detection productions in the unit-task **color-goal**. When a holding aircraft is detected, **detect-red** pushes two subgoals for that aircraft corresponding to the two actions that can be performed depending on the direction of the aircraft: accept and welcome for incoming aircraft and transfer and contact for outgoing aircraft. That pipelining of actions certainly leads to more duplicated or incorrect messages than necessary, but a more precise treatment would require a complex reasoning process that would best be implemented as a separate unit task. Subjects caught in a holding violation might not want to spend the time on such a process and might resort to this shotgun approach, and this would have significantly increased the model complexity as well. Nonetheless, further analysis of subjects' strategy choice in the case of holding infractions would be needed to sort out the matter, and too little data was available because of the relative rarity of that condition.

Table 2.5 presents the productions for the unit task **process** responsible for actually processing an action request through a sequence of button clicks and mouse selections. This unit task is common to the color and text condition because although the information available will vary between conditions the basic logic of the unit task remains unchanged. The first action to perform is to click the button on the right side of the screen

corresponding to the requested action. The production **answer-speed-request** determines if the aircraft is blocked and pushes the corresponding button. The production **answer-other-requests** pushes the corresponding button for all other actions because no action-specific decision is necessary. The next action is to select the aircraft. However, in some conditions (e.g. responding to a text message) the location of the aircraft is not yet known and the aircraft will have to be located first. The production **memory-for-position** attempts to extract the aircraft position from an existing process chunk. If that fails, the productions **find-position-inner**, **find-position-between** and **find-position-outer** scan the radar screen area corresponding to the action requested (e.g. the outer area for accepting incoming aircraft) to find the aircraft position. This is another instance of the retrieve-vs-compute design pattern encountered in the two previous unit tasks. Once its position is determined the target can then be selecting by production **click-target**. The production **click-controller** then selects the external controller associated to the aircraft, unless preempted by productions **skip-speed-change-controller** and **skip-welcome-controller** that explicitly skip that step for the speed change and welcome actions respectively. The **click-send** production then clicks the send button and pops the goal, which becomes a memory chunk encoding the processing of this task. As in previous unit tasks, there is an additional production to detect the onset of an event, in this case the appearance of a colored aircraft on the radar screen, and creates a subgoal to process that aircraft when the current goal is completed. That subgoal is then returned to the parent goal and pushed by the production **subgoal-next** in the **color-goal** unit task.

Next-target

IF the *goal* is of type process and the display condition is **color**

and a *aircraft* of *color* is detected

and there is a *rule* associating *color* with *action*

THEN note *position* of *aircraft* and

create a *subgoal* to process *action* on *aircraft* in *position*

Answer-speed-request

IF the *goal* is of type process with action **speed-change** for *aircraft* in *position*

and step **select**

THEN determine if *aircraft* is blocked

and push button corresponding to accept-reject decision

and note that the step is now **target**

Answer-other-requests

IF the *goal* is of type process with *action* and step **select**

THEN push button corresponding to *action* and note that the step is now **target**

Memory-for-position

IF the *goal* is of type process with *aircraft* and no known position

and there is a *chunk* for processing *aircraft* in *position*

THEN note *position*

Find-position-inner

IF the *goal* is of type process with action **speed-change** for *aircraft* and no position

and the location of *aircraft* in screen area **inner** is found to be *position*

THEN note *position*

Find-position-between

IF the *goal* is of type process with action **contact** for *aircraft* with *controller*

and no known position

and the location of *aircraft* in screen area **between** on *controller* side is *position*

THEN note *position*

Find-position-outer

IF the *goal* is of type process with *aircraft* and no known position

and the location of *aircraft* in screen area **outer** is found to be *position*

THEN note *position*

Click-target

IF the *goal* is of type process with *aircraft* in *position* and step **target**

THEN select *aircraft* in *position* and update step to **controller**

Skip-speed-change-controller

IF the *goal* is of type process with action **speed-change** and step **controller**

THEN update step to **send**

Skip-welcome-controller

IF the *goal* is of type process with action **welcome** and step **controller**

THEN update step to **send**

Click-controller

IF the *goal* is of type process with *aircraft* step **controller**

THEN select *controller* associated with *aircraft* and update step to **send**

Click-send

IF the *goal* is of type process with step **send**

THEN push button **send** and pop *goal*

Table 2.5: Productions applicable to the unit task **process**

The final part of the model concerns the code at the top of the model that is used to compute the workload estimates. While ACT-R has traditionally shied away from such meta-awareness measures and concentrated on matching directly measurable data such as external actions, response times and eye movements, it is by no means incapable of doing so. For the purpose of this model, we proposed a measure of cognitive workload in ACT-R grounded in the central concept of unit task. Workload is defined as the ratio of time spent in critical unit tasks to the total time spent on task. Critical unit tasks are defined as tasks that involve actions, such as the process goal that involves handling an event with 3 or 4 mouse clicks, or tasks that involve some type of pressure, such as the scanning goal described above that results from an onset detection i.e. carries an expectation of a new event that needs to be handled promptly. The ratio is scaled to fit the particular measurement scale used in the self-assessment report.

Finally, two specific considerations need to be discussed. First is the decision not to use ACT-R/PM. That decision was primarily driven by practical considerations, including the tight development schedule for phase I and the fact that ACT-R/PM at the time only ran on the Macintosh while the D-OMAR simulation only ran on Windows. While the model is at a slightly higher degree of abstraction than ACT-R/PM (for example, it performs a search of a list of messages in a single production), it operates in substantially similar ways and an ACT-R/PM version could be developed fairly straightforwardly by expanding those specific productions that currently call the interface code directly. This would allow us to replace the only two parameters that we estimated, the average perception and action times, with more accurate ACT-R/PM predictions. However, it is

an open question whether a higher degree of fidelity at the perceptual and motor level would necessarily lead to a better model of the relatively higher-level data (e.g. total penalty points) presented here. But that question of the right level of analysis is a fundamental one that an ACT-R/PM version of this model would allow us to pursue.

The second consideration is the inclusion on the web site and CD-ROM of the complete text of the model. The first thing to point out is that the entire code of the model of a relatively complex task can indeed be included in a dozen fairly sparse pages. This is a reflection of the architecture's ability to generate complex behavior from a comparatively simple model. More fundamentally, providing the running code of our models has been an increasingly important practice in the ACT-R community. For example, the code from all the models described in our book (Anderson & Lebiere, 1998) is available on our web site (http://act.psy.cmu.edu) and can even be run directly on the web without having to download and install ACT-R. A point-and-click web interface enables visitors to easily change the model parameters and re-run the model to determine if its predictions are overly sensitive to the values of the parameters. Moreover, modelers are encouraged to adopt, if not pieces of models directly (which has been done, e.g. Byrne & Anderson, 2001), certainly the design patterns used in other models, as we have attempted to do in this case. The goal of this openness is both to facilitate model development and to increase the constraints on the resulting models in order to increase their predictiveness and generality.

Experiment I Results

Because the variability in performance between runs, even of the same subject, is a fundamental characteristic of this task, we ran as many model runs as there were subject runs. Figure 2 compares the mean performance in terms of penalty points for subjects and model for color (left three bars) and text (right three bars) condition by increasing workload level. The model matches the data quite well, including the strong effects of color-vs-text condition and of workload for the unaided (text) condition.

<Insert Figure 2 here>

Because ACT-R includes stochasticity in chunk retrieval, production selection and perceptual/motor actions, and because that stochasticity is amplified by the interaction with a highly dynamic simulation, it can reproduce a large part of the variability in human performance, as indicated by Figure 3 which plots the individual subject and model runs for the two conditions that generated a significant percentage of errors (text condition in medium and high workload). The range of performance in the medium workload condition is almost perfectly reproduced other than for two outliers and a significant portion of the range in the high condition is also reproduced, albeit shifted slightly too upward. It should be noted that each model run is the result of an identical model that only differs from another in its runtime stochasticity. The model neither learns from trial to trial nor is modified to take into account individual differences.

<Insert Figure 3 here>

The model reproduces not only the subject performance in terms of total penalty points, but also matches well to the detailed subject profile in terms of penalties accumulated under eight different error categories, as plotted in Figure 4.

<Insert Figure 4 here>

The model also fits the mean response times (RT) for each condition, as reported in Chapter 8. The differences in RT between conditions are primarily a function of the time taken by the perceptual processes of scanning radar screen and text windows. A more detailed analysis is presented in Figure 5, which plots the detailed pattern of latencies to perform a required action for each condition and number of intervening events (i.e. number of planes requiring action between the time of a given plane requiring action and the time the action is actually performed). The model predicts very accurately the degradation of RT as more events compete for attention, including the somewhat counterintuitive exponential (note that RT is plotted on a log scale) increase in RT as a function of number of events rather than a more straightforwardly linear increase.

<Insert Figure 5 here>

In a crucial test of the model's multi-tasking abilities, it also closely reproduces the probability of response to a required action in terms of number of intervening events

(plotted in Figure 6) before the action can be performed, a very sensitive measure of the ability to detect and process events immediately after they occur.

<Insert Figure 6 here>

That multi-tasking capacity results from the model's ability to detect event onsets and set the next goal to process those events. Thus, despite ACT-R's strong goal-directed behavior, as indicated by its structure pictured in Figure 1, it can exhibit the proper level of multi-tasking abilities without requiring any alteration to its basic control structure. Interestingly, a version of the model that ignores event onsets and stays with a very systematic scanning strategy actually performs quite well but provides a very different multi-tasking profile.

Finally, the model reproduces the subjects' answers to the self-reporting workload test administered after each trial. Since ACT-R doesn't have any built-in concept of workload, we simply defined the workload of an ACT-R model as the scaled ratio between the time spent in critical unit tasks to the total time on task. The critical unit tasks in which the model feels "pressured" or "busy" are defined as the **Process** goals, in which the model is busy performing a stream of actions, and the **Scan-Text** goals that are the result of an onset detection, in which the model feels "pressured" to find and process a new event requiring action. As shown in Figure 7, that simple definition captures the main workload effects, specifically effects of display condition and of schedule speed. The latter effect results from reducing the total time to execute the task (i.e. the

denominator) while keeping the total number of events (roughly corresponding to the numerator) constant, thereby increasing the ratio. The former effect results from adding to the process tasks the message scanning tasks resulting from onset detection in the text condition, thus increasing the numerator while keeping the denominator constant thereby increasing the ratio as well. Another quantitative effect that is reproduced is the higher rate of impact of schedule speed in the text condition (and the related fact that workload in the slowest text condition is higher than workload in the fastest color condition). This is primarily a result of task embedding, i.e. the fact that a process task can be (and often is) a subgoal of another critical unit task (scanning a message window following the detection of an onset in that window), thus making the time spent in the inner critical task count twice.

<Insert Figure 7 here>

In summary, the advantages of this model are that it is relatively simple, required almost no parameter tuning or knowledge engineering, provides a close fit to both the mean and variance of a wide range of subject performance measures as well as workload estimates, and suggests a straightforward account of multi-tasking behavior within the existing constraints of the ACT-R architecture.

Experiment II Model

Initial model

The methodology adopted in creating the experiment I model was to not try to reverse-engineer subjects' procedures through a cognitive task analysis or similar methods but instead to simply develop a model that was simple and arose naturally from the architecture. At the basic cognitive level, it meant relying on architectural mechanisms like chunk creation to seamlessly accomplish functions like episodic memory for past actions and aircraft positions. At the higher, structural level, it meant leveraging the close relation between the architectural concept of goal and the HCI concept of unit task to structure the model around a modular set of goals and the knowledge needed to solve them. We will follow this methodology again in the development of the experiment II model.

At the structural level, this new model involved the removal of three unit tasks from the original model and the addition of one. The unit tasks removed were related to the text condition, which was not used in this model. They were the high-level unit task that handles the allocation of attention to various screen areas, and the specialized unit tasks to scan text windows and the radar screen. Because of unit task modularity, they didn't need to be removed and could simply have been ignored, never being called upon, but we took them out for reasons of simplicity. The two remaining unit tasks are the high-level unit task for scanning the radar screen and identifying color-coded aircraft and the low-

level unit task responsible for producing the sequence of actions needed to process an aircraft. The new unit task being added is inserted between the two. It involves a decision goal that is called by the high-level goal when a magenta aircraft is identified as requesting an altitude change. This goal involves deciding which action needs to be performed on the aircraft, then calls the process goal to perform it. This new unit task involves eight new productions that can apply to goals of the decision type, one of which being the crux of the decision engine while the others handle relatively straightforward stimuli input and feedback processing. The new model has a total of 19 production rules distributed over the three goal types of color, decision and process.

At the cognitive level, categorization is handled by relying on basic architectural mechanisms. While the concept of categorization evokes the idea of production rules, the basic mechanism on which the initial model relies is memory. Before rules can be formulated, the knowledge must reside in the system on which to base those rules. Thus this model will rely on the same basic mechanism as the experiment I model, that is ACT-R's automatic creation of memory chunks encoding past goals, in this case goals of the new decision type. When a decision is made and the feedback processed, the decision goal is popped and becomes a long-term memory chunk. Future decisions can then be made from retrieving past decision chunks.

This model can be characterized as an instance-based model (e.g. Logan, 1988). Those models are characterized by an initial reliance on a general-purpose strategy (e.g. relying on external aids, performing a computation procedure, or, as in this case, simply

guessing). As that strategy is exercised, knowledge from past decision-making instances builds into long-term memory and can gradually be used as the basis for making decisions. This gradual switch from general procedures to specific expertise is a hallmark of human cognition. In ACT-R, that approach has been applied with great success to a broad array of domains including control problems, i.e. the Sugar Factory (Lebiere, Wallach & Taatgen, 1998; Wallach & Lebiere, 2002) and the Transportation Task (Wallach & Lebiere, 2002), game playing, i.e. Paper Rock Scissors (Lebiere & West, 1999; West & Lebiere, 2001), Backgammon (Sanner, Anderson, Lebiere & Lovett, 2000) and 2x2 Games (Bracht, Lebiere & Wallach, 1998; Lebiere, Wallach & West, 2000) and decision making, i.e. real-time dynamic decision making (Lerch, Gonzalez & Lebiere, 1999; Gonzalez, Lerch & Lebiere, 2003) and multi-person decision-making tasks (Lebiere & Shang, 2002). One argument often raised about the general instance-based approach is that it has so many degrees of freedom in representation and parameters that it can be applied to produce anything. While such objections are often disingenuous (and are often leveled at the practice of cognitive modeling in general, e.g. see (Roberts and Pashler, 2001)), the ACT-R models listed above model a significant number of tasks over a broad range of domains while adopting consistent representations and parameter values. The ability to apply the same mechanisms across a wide range of tasks illustrates the major integrative advantage of cognitive architectures.

We will now examine the model in detail. As previously mentioned, the production rules in the two remaining unit tasks from the experiment I model are essentially unchanged. Exceptions involve a single production in each task that interacts with the new decision

goal. In the top-level color goal, a new production **color-magenta-action** detects the magenta color associated with a plane requesting an altitude change, then pushes a goal to make a decision on whether to accept or reject the request (instead of directly processing the plane). In the low-level process goal, a new production **answer-altitude-requests** detects that the request is for an altitude change and presses the button (accept or reject altitude change) corresponding to the decision. Since this is the only request for which the button to select is not uniquely determined by the request but is instead a function of a decision made, a different production is thus required.

All other productions apply to the decision goal. The order in which the productions are listed represent their utility ranking, and thus usually the order in which they fire to solve a given goal. The first three productions, **target-fuel**, **target-turbulence** and **target-size**, encode the characteristics of the aircraft, i.e. its fuel, turbulence and size respectively, by moving attention to the various pieces of information near the aircraft. The production **remember-decision** is the key production for this goal because it is primarily responsible for the decision-making. It attempts to make a decision by retrieving a past decision for an aircraft sharing the characteristics of the current one. If it is successful in retrieving such a chunk, it simply makes the decision that was correct for that chunk. If no chunk can be retrieved, however, then a backup production called **guess-decision** will make a decision by simply guessing randomly. Once a decision has been made, the production **subgoal-process** pushes a subgoal to process the aircraft with that decision. After the process subgoal has been completed, the decision goal is resumed. The production **wait-for-feedback** will wait for the feedback to appear. Once a feedback is available,

indicating either a correct or incorrect decision, the production feedback can fire. If the feedback indicates an incorrect decision, the decision is changed to the correct one. In either case, the goal is then popped, creating a declarative memory chunk (or reinforcing an identical one) holding the correct decision for an aircraft with these characteristics. That chunk can then potentially be retrieved as a basis for future decisions.

Color-Magenta-Action (**color** unit task)

IF the *goal* is to detect a color aircraft at *position* and its *color* is **magenta**

THEN push a *goal* to make a decision for *aircraft* at *position*

Target-fuel/turbulence/size (3 separate productions)

IF the *goal* is to make a decision for *aircraft* and no fuel/turbulence/size is known

THEN encode the *fuel/turbulence/size* of *aircraft* in the *goal*

Remember-decision

IF the *goal* is to make a decision for *aircraft* of *fuel*, *turbulence* and *size*

AND there is a memory of a decision for an aircraft of *fuel*, *turbulence* and *size*

THEN select decision

Guess-decision

IF the *goal* is to make a decision for *aircraft* of *fuel*, *turbulence* and *size*

THEN randomly decide between **accept-altitude** and **reject-altitude**

Subgoal-process

IF the *goal* is to make a decision for *aircraft* at *position*

THEN push the *goal* to process decision for *aircraft* at *position*

Feedback

IF the *goal* is to make a decision and *feedback* is available

THEN update decision according to *feedback* and pop the *goal*

Wait-for-feedback

IF the *goal* is to make a decision and a *decision* has been made

THEN wait for feedback

Answer-altitude-requests (**process** unit task)

IF the *goal* is to process an altitude-request *action* and the *step* is **select**

THEN push the *button* corresponding to the *action* and change the *step* to **target**

Table 4.1: Production Rules for Decision Goal and Related Goals

The effort parameters for these productions were set in accordance with the parameters for productions in the experiment I model and with similar parameters in other ACT-R models. By default, all productions took 50 msec to fire. The three encoding productions (**target-fuel**, **target-turbulence** and **target-size**) were assigned a latency of 200 msec. Because those items are in direct proximity to the aircraft and in predictable locations, that is directly compatible with the 185 msec estimate for small shifts of attention, such as when scanning menu items using the perceptual/motor layers (Byrne & Anderson, 1998). The **feedback** production latency was set to 500 msec, in accordance with the color-detection productions in the color goal since both represent the detection of an unscheduled event such as the change of color of an aircraft or the appearance of the feedback icon. The **wait-for-feedback** production latency was set to 1 second, as for the

wait production in the experiment I model, representing the coarseness of the general alertness loop. The **answer-altitude-requests** production latency was also set to 1 second, as for all other action productions, representing the average action time factoring for an averaging of Fitt's law mouse movements, action preparation and clicking movement. As in the experiment I model, the latency times were not fixed but instead varied according to a uniform distribution of +/- 25% around the mean. In summary, those parameters were not estimated to fit the data but instead generalized directly from the experiment I model and other architectural guidelines.

The critical step in the decision goal is the attempt to retrieve a past decision to provide the basis for the current one. That step is described schematically in Figure 8. On top is the current goal, with each square representing one slot of the goal. After the first three encoding productions have fired, the goal contains the actual size, fuel and turbulence of the current aircraft, with no decision currently made.

<Insert Figure 8 here>

At the bottom is one of possibly many decision chunks in declarative memory. Note incidentally that those chunks have the same structure as the current decision goal: since past goals become chunks when they are popped, the correspondence between structures is logical and allows for a direct correspondence in matching. One could request that the chunk retrieved from memory match exactly the characteristics of the current aircraft in the goal. This would correspond to the exact (symbolic) match process in ACT-R.

However, this would be undesirable for a number of reasons. First, at the start the knowledge base is still very sparse and activations are weak: requiring the retrieval of an exact match would severely limit the probability of successful retrievals and reduce the decision to random guessing. Second, retrieving items that do not perfectly match allow for the model to generalize to new instances that have never been seen before, an essential characteristic in the real world where characteristics are not binary and the same situation is never seen exactly again. Finally, it makes the process more robust by preventing a single specific instance for exerting excessive influence (e.g. if it happens to be wrong) by letting all neighboring instances participate in the retrieval process rather than limit it to the one that happens to match exactly. This process of generalizing to similar stimuli directly produces the patterns observed for central vs. peripheral stimuli.

In the training phase, only eight decision chunks will be created in memory, because that is the number of unique stimuli. For each new round of stimuli, the goal being popped is identical to an existing chunk in memory and is thus merged with it, resulting in a strengthening of the existing chunk through the base-level learning equation. As displayed in the table above, the activation of the chunk is determined by its base-level activation B_i, with noise of amplitude s added. Over time, the base-level activation of decision chunks will increase, making it increasingly likely that their activation will be higher than the retrieval threshold τ and retrieval will be successful. If the memory chunk doesn't match the retrieval pattern perfectly, its match score will equal its activation decreased by the similarity between desired retrieval pattern and actual chunk value, scaled by the mismatch penalty MP. This partial matching process will apply for

all slots specified in the retrieval pattern, e.g. in the case illustrated above the similarity between large and small, Sim_{sl}, and between turbulence level 1 and 3, Sim_{13}, both apply additively to the match score. Partial matches are less likely to be the most active and to be retrieved, but if the initial activation was high enough to overcome the mismatches and/or the activation of the perfectly matching chunk was sufficiently low, they have a chance to win the activation race and be the retrieved chunk.

Just as for productions, parameters involved for declarative memory were set using constraints from the architecture and other models. The latency factor *F* scaling retrieval latency was left at the architectural default of 1.0. The decay rate *d* in the base-level learning equation was also left at its architectural default of 0.5 used in almost all ACT-R models. The mismatch penalty *MP* scaling the similarity decrements in the partial matching equation was left at its default value of 1.5. The activation noise *s* controlling the stochasticity of memory retrieval was left at its default value of 0.25 used in many ACT-R models. The only architectural parameter that doesn't have a consensus default value is the retrieval threshold τ, which was coarsely estimated at –1.0. As Anderson et al (1998) have observed, the value of the retrieval threshold seems to vary with the average activation level and cannot seem to be fixed at this time. However, the value used here is well within the range of values for that parameter used in other models. As for chunk-specific parameters, the prior values for the activations of the color-action mapping chunks were left at their values set in the experiment I model. The only additional parameters to be specified were the similarities between the quantities used in the stimuli, i.e. the fuel, size and turbulence. Similarities between quantities are typically

set according to regular scales, usually linear or exponential scales (e.g. Lebiere, 1998; Wallach & Lebiere, 2003). In the initial model, we set the similarities to decrease linearly as a function of distance on each scale, reaching minimal values for the extreme items of the scale.

Further Refinements

Based on the results of the first model (see next section), we implemented three changes and a significant addition to the model.

The first change was primarily in reaction to the fact that, while the response time for the secondary task (transferring planes) was about right (as was to be expected since that task and that part of the model were essentially unchanged since the first experiment), the response time for the primary task (authorizing altitude changes) was significantly too high. We reasoned that a possible reason was that while the altitude change task was clearly presented as the primary task, we did not provide a priority ordering between the various tasks. We made that choice partly for consistency with the experiment I model and partly for simplicity, but it was clear that subjects gave higher priority to the primary task. Therefore, we modified the production ordering to give priority to the magenta aircraft over others when multiple planes request action at the same time. As expected, the response time for the primary task decreased significantly (by about 1 second), bringing it significantly closer to the subject data.

The second change concerned the similarities between stimuli components. One consequence of the linear similarities is that extrapolated stimuli had the same error rate as their trained neighbors because the translation in stimuli values simply added a constant value to the mismatch penalty for all training chunks, leaving the probability of retrieving them unchanged. While linear similarities are often used for their simplicity, exponentially decreasing similarities have also been used and correspond more closely to human similarity metrics on domains like numbers (e.g. Whalen, 1997). Therefore, we changed the similarity scale between stimuli components to decrease exponentially with distance. That distribution has one parameter, which is the rate of the exponential decrease. It was fixed to leave the similarities between training stimuli unchanged, therefore affecting only the similarities to extrapolated stimuli. The result of a switch to an exponential similarity function is to decrease the similarity between close stimuli and increase the similarity between distant stimuli. This leads to an increase in probability of extrapolated error, because distant instances, which are not likely to generalize well, are now more likely to be retrieved and generate the incorrect response.

The third change concerned the workload definition. While the workload formula based solely on time on task captured the main effects, it did so so weakly that the match to the data is quite poor. There is just not enough difference in time spent in critical unit tasks between the various conditions and blocks to reproduce the size of the effects in the data. However, one measure of performance is strongly correlated with the observed changes in workload: the percentage of errors in altitude change decisions. Therefore, we added the time-based and success-based (in terms of number of errors) measures of effort, still

divided by total time on task, with the same multiplicative factor as in experiment I. One basic question was how to combine effort and success given that they involved two separate scales. To bridge the gap, we assigned to the goal of making an altitude decision the value G from the production utility function, which is its intended semantic in term of time worth devoting to the task. Thus, we multiplied the number of errors by the value of G, which was set to 15 seconds, added it to the time spent on critical unit tasks (in this case, the decision and process goals) and divided by the total time on task. The result is a computational workload measure that closely captures the human data.

The main addition originated from the recognition that while the instance-based model did an excellent job at capturing human performance for problem type 6, it just could not learn fast enough to capture the very steep learning curve for problem type 1. Therefore, while memory is still the primary foundation for categorization as is confirmed by the problem type 6 data, an additional mechanism, rule learning, must be introduced to account for the problem type 1 data. While category rule learning can certainly be thought of as a conscious process where explicit rules can be formulated, represented as chunks in declarative memory, then iteratively tested, modified and rejected or accepted, that process is fraught with degrees of freedom. In effect, a great number of different algorithms can be implemented (Anderson & Betz, 2001), individual differences are paramount, and the architecture provides very little constraint on the process. Therefore, we tried a different approach to provide for the learning of general rules while preserving strong architectural constraints.

To accomplish those ends, we represented categorization rules as production rules. Specifically, we created one production rule for each possible single-dimensional categorization rule, for a total of 6 production rules. Those productions could have been created through the process of production compilation (Taatgen & Anderson, 2002), but we wanted to avoid the complexity of the underlying process of explicitly formulating those rules. Those six production rules now compete with the Remember-decision and guess-decision rules. The basis of the competition is the subsymbolic utility learning mechanism, which tracks the effectiveness of those rules at producing the correct answer and successfully solving the decision goal. For problem type 6, the single-dimensional production rules do no better than the random rule and worse than the remember rule, and are therefore weeded out. For problem type 3, no single-dimensional rule can provide perfect categorization but some can do significantly better than the random rule, and even the remember rule until enough instances have been learned. In that case, the rule first predominates until it is replaced by the retrieve production. For problem type 1, one of the six rules can provide perfect performance and its utility will quickly become dominant, leading to the nearly uniform use of that rule. The only parameters of the utility learning process are the value of the goal, G, which has previously been fixed at 15, and the value of the utility noise parameter, which is left at the default value of 1.0. A process of categorization rule learning has been added while preserving strong architectural constraints and avoiding arbitrary degrees of freedom.

Experiment II Results

Original Results

The most important quantitative results are the percentages of error committed in the primary category task, presented in Figure 9. As for other following data figures, the left plot is for problem type 1, the central plot is for problem type 3 and the right plot is for problem type 6. The fit to problem type 6 is excellent. This is consistent with the fact that no useful (linear) rule exists for problem type 6 and that an instance-based strategy like the one used in the model is likely to be the most effective for that problem type. For problem type 3, the model captures the shape of the curve but is consistently slower than human subjects at learning the category by an approximately constant factor. The fit to problem type 1 is the worst, with the model only starting to significantly learn the category in block 4 while humans have already significantly mastered it by block 2. While instance-based learning is more efficient on problem type 1 than 6 because a neighboring instance retrieved through partial matching is more likely to be of the right category, it is not nearly enough to match the human subjects. This suggests that a more efficient strategy exists for learning problem type 1 (and probably problem type 3).

<Insert Figure 9 here>

Figure 10 presents in the same format the penalty points for the secondary task, processing the aircraft moving between controller airspaces. While errors on the

secondary task are too few to generate significant numbers of penalty points, the model generally produces similar levels and patterns. The main sources of errors in the secondary task are the lack of time to accomplish the task in a timely manner and commission errors when retrieving color-action mapping chunks. Those two sources of errors are fundamentally the same as for the primary task. Therefore the two error measures are not independent but instead constrain each other through the same architectural mechanisms. They cannot be adjusted independently but instead provide converging evidence on model performance.

<Insert Figure 10 here>

Figure 11 presents the response time data for the secondary task. Because the response time to the secondary time is primarily determined by the latency of the processing steps and those parameters were left unchanged from the experiment I model, this is a direct prediction of the original model. No significant speedup with practice or any significant effect of primary task category is predicted, in line with the human data.

<Insert Figure 11 here>

Figure 12 presents the response time data for the primary task. The model consistently overestimates the amount of time required by the primary time. In particular, response time for the primary task is larger than for the secondary task because the primary task involves an additional decision step that requires significant time. However, this doesn't

take into account the fact that the primary task, as indicated by its name, carries a higher priority than the secondary task and, when primary and secondary tasks conflict, the former is likely to take precedence. In the initial model, we did not implement any specific precedence scheme, which might have led to this overestimate of primary task response time.

<Insert Figure 12 here>

A speedup with practice of about 1 second is predicted in all conditions, consistent with the data. This results from the increasing success and speed of retrievals. Initially, retrieval of previous instances is more likely to fail, which takes longer than successful retrievals. Moreover, over time the activation of chunks representing previous instances increases with rehearsal, which according to the retrieval latency equation decreases the retrieval time. Both factors contribute to the speedup. However, the speed up seems to take place somewhat later than for the subjects. Also, the model doesn't predict the shorter response time for problem type 1 observed for the subjects. This confirms the conclusion reached from the error rate data that rule learning might be involved for problem type 1, which would also decrease the response time in addition to increasing accuracy.

Figure 13 presents the workload ratings for the various conditions. No change was made to the definition of workload used for experiment I, which was a scaled ratio of time spent in critical goals to total time on task. The critical goals are the process goals, as in

experiment I, and the new decision goals. Because no change was made to the definition or the parameters, this is a direct prediction from the experiment I model. While it does a pretty good job at predicting base workload, such as in block 4 and 8 of problem type 1 and block 8 of categories 3 and 6, it fails to reproduce the full range of the problem type and practice effects observed in the rest of the human data. The model in fact exhibits very slight effects of problem type and practice, but because they only reflect the response time decrease observed for the primary task (specifically the decision goal), they are insufficient in capturing the significant effects in the human data. Since the human data does not indicate a sizable difference in response time but significant effects of problem type and practice on response accuracy that mirror the effects observed in the workload data, it seems reasonable that the subjects workload self-assessments reflect not only considerations of time but success as well.

<Insert Figure 13 here>

Figure 14 presents error percentage data for the primary task in the transfer condition. The data presented represents the percentage of errors in the primary task for the last block (8) of the training phase, the instances of the transfer phase that were seen in the training phase ("Trained") and the instances of the transfer phase that were seen in the training phase ("Extrapolated"). We will focus on the data points for the transfer phase. The most important thing about the transfer phase is that it is handled exactly the same as the training phase, i.e. every stimulus is answered by attempting to retrieve a similar instance from declarative memory. No new procedure, with the attending degrees of

freedom that it would introduce, is used for the transfer phase. The match to the trained examples is excellent.

<Insert Figure 14 here>

For the extrapolated examples, the model predicts a similar error percentage to the trained examples, with the minor variations in the results due to the stochastic nature of the model runs. This results because of the form of the partial matching equation used. An extrapolated stimulus will have an additional activation penalty subtracted from its match score compared to the neighboring trained stimulus, but since the similarity function used in the original model is linear, the same penalty will apply to all chunks and the probability of retrieving any given chunk will be unchanged (other than for their probability of reaching the retrieval threshold, but the chunks are active enough that this is not a factor). This is a direct consequence of using a linear similarity metric. Other forms of similarity functions (e.g. ratio or exponential, as have been used in other ACT-R models) have decreasing penalties with distance and would show the proper increase in error for extrapolated instances. The aggregation over broad categories of stimuli, such as trained, extrapolated and equidistant, might obscure more specific results of the model. Figure 15 presents a comparison of human data and model results on the training phase for all individual stimuli:

<Insert Figure 15 here>

Each individual point in the graph corresponds to a single stimulus (modulo category-preserving transformations), plotted by problem type. The X axis is the decision probability for the stimulus (accept, but it could equally well be decline) in human data, and the Y axis is the same for model results. Thus, a perfect fit would have all data points on the x=y diagonal. The more points deviate from that line, the poorer the fit. Quantitative fits by categories are given at the top of the figure. Again, an equation of y=0+1x with R^2=1.0 would indicate a perfect fit. The linear regression curves actually displayed are not quite that perfect, but all have a small intercept (absolute value of 0.05 or lower) and a slope roughly between 0.8 and 1.0. The underestimate of the slope for problem type 3 and especially problem type 1 is consistent with the larger consistency values for the model in the previous figure, especially for problem type 1 where the model is slower at learning the correct categorization values and therefore produces more extreme values. The R^2 correlations are generally high, indicating a good fit, though interestingly and somewhat surprisingly R^2 is best for problem type 1 (0.890) and worse for problem type 6 (0.485), which is the opposite of the results for the aggregate error percentages presented previously where the best fits were for problem type 6 and the worse for problem type 1! This primarily results from the characteristics of the categories. Problem type 1 is easier to classify and thus produces more extreme probability values, which makes larger correlation values more likely. Conversely, problem type 6 is harder to classify, with lots of mixed probabilities toward 0.5, which reduces possible correlations. Thus, R^2 correlations is actually a misleading indicator of model fit, in this case primarily reflecting characteristics of the task. A better measure of fit is Root Mean Square Error (RMSE), which measures the deviation between data and

predictions (Schunn & Wallach, 2003). RMSE is 14.1% for Problem type 1, 13.4% for Problem type 3 and 12.5% for Problem type 6, which correctly indicates a better fit for Problem type 6 and a worse fit for Problem type 1. Similar results can be plotted for transfer stimuli only, with similar fits and actually a slightly smaller RMSE for Problem type 6.

Final Results

As described in the modeling section, the main change between original and final model is the introduction of 6 production rules representing all possible single-dimensional categorization rules to compete with the retrieval and random strategies on the basis of learned production utility. The principal goal was to allow faster learning of problem type 1. Figure 16 presents the learning curves of error percentages on the primary task for the three categories for the original and revised model. One can see that the final results are significantly improved over the original ones. For problem type 6, no significant change occurs and the excellent fit to human data of the original model is preserved. Since no single-dimensional rule can do better than 50% correct, i.e. chance, they are initially indistinguishable of the random production and then are quickly discarded in favor of the retrieval production rule. For problem type 3, the best a single-dimensional rule can do is to be successful 75% of the time, which is initially significantly better than the random and retrieval strategies and will boost performance to the subject level. Most significantly, for problem type 1 a perfect single-dimensional rule exists and will be quickly identified. Because of randomness in the utility computations,

other rules, especially the retrieval rule, still occasionally fire depending on their utility level, generating less-than-perfect model-performance (about 10% errors) similar to humans.

<Insert Figure 16 here>

Figure 17 presents the response time for the primary task in the final model. Prioritizing the primary task over the secondary task has led to a decrease in the primary task response time, much closer to subjects RT for categories 3 and 6, but still about 1 second too high for problem type 1.

<Insert Figure 17 here>

Interestingly, the response time for the secondary task has not significantly increased because a better prioritization resulted in better performance overall as confirmed by Figure 18, which presents the penalty points for the secondary task. A better prioritization scheme for the primary task has not only lowered response times but also reduced error rates for the secondary task on a par with human level. As we have seen many times, components and parameters of the model have an influence on multiple data measures and cannot be optimized separately.

<Insert Figure 18 here>

Figure 19 presents the workload ratings for the final model. By adding a success-based component to the workload formula, the model can now capture practice and problem type effects in the workload measure. Even though workload levels seem a bit too high by about a constant factor, both the size of reduction with practice and the increase with problem type difficulty are about the right size, which is notable since the size of the success factor in the workload equation was not a free parameter but was instead determined by the same G factor as weighing cost and success in the utility equation.

<Insert Figure 19 here>

Figure 20 presents the performance in the transfer task. Changing the similarity function to exponential similarities that exhibit sharper initial differences and then gradually flattening similarities similar to those obtained in human rating studies (e.g. Whalen, 1996) increases errors for extrapolated items because it reduces the relative probability of retrieving neighboring items. Since the change in similarity functions preserved the similarities between trained items, it didn't change performance in the training task, and also fixed the single parameter in determining the exponential function. Therefore, the size of the increase in errors for extrapolated items was not optimized but instead a direct prediction of the shift to an exponential similarity function.

<Insert Figure 20 here>

Discussion and conclusion

Parameterization

Roberts and Pashler (2000) suggested that the behavior of cognitive models should be studied over their entire range of possible parameters to determine not only what data models can account for but also what data they cannot account for. It is of course an open question what the model parameters are. Real-valued architectural and knowledge parameters seem to qualify, but they do not really constitute degrees of freedom if they are treated as constants set by the architecture or by other models. On the other hand, Baker and Koedinger (2003) have suggested that every knowledge structure itself, such as each chunk and production rule, should be counted as a free parameter. Our view is that while as long as the knowledge structures are specified by modelers there will be a possibility of exploiting degrees of freedom in model specification, which need not be the case. Our methodology in developing our model has been to aim for the simplest, most natural way to solve the problem in the ACT-R architecture, and explicitly mention when we revised that model and why. Moreover, Anderson, Bothell, Douglas & Haimson (2003) and Taatgen (2003) have used the production compilation mechanism to automatically encode instructions whose interpretation would then be compiled into the production rules executed by the model. Because task-specific declarative knowledge is the result of a direct encoding of instructions given to subjects and task-specific production rules are the product of an architectural compilation mechanism (and a general-purpose interpretation mechanism), one can argue that no degrees of freedom

exist in the creation of their model. While we did not follow that methodology here, we tried to avoid endowing the model with any expert knowledge that would clearly go beyond the instructions received.

Nonetheless, examining the influence of real-valued parameters on the model results is a valid and often worthwhile exercise in which we have engaged regularly (e.g. Lebiere, 1998; Lebiere & Wallach, 2001). In this section, we will describe the impact of variations of three architectural parameters directly involved in the declarative memory retrieval process central to the instance-based categorization strategy. Those parameters are the retrieval threshold *RT*, which determines when a chunk is active enough to be retrieved, the activation noise *S* which controls the stochasticity of the chunk activations and therefore of the retrieval process, and the mismatch penalty *MP*, which scales the activation penalty for mismatches and thus controls the degree of retrieval generalization. The key measure of performance as a function of parameter variation is the probability of categorization errors for the primary task for all training blocks (a block here corresponds to a single presentation of all instances, i.e. half a block as described previously). Figure 21 presents the probability of categorization errors as a function of the retrieval threshold:

<Insert Figure 21 here>

As expected, performance is worse for relatively high retrieval thresholds (0.0 and –0.5), which delay retrieval from memory longer. But one would assume that the lower the retrieval threshold, the easier the access to memory and therefore the better the

performance. But that is counting without the possibility of errors of commission in memory retrieval, i.e. the possibility of retrieving an incorrect instance chunk because it is very active and can overcome mismatch penalties. Thus an overly low retrieval threshold leads to a process where a few chunks are retrieved very quickly, build up more strength through rehearsal, and intrude upon other retrievals, leading to a permanently high number of errors. That is the pattern displayed for retrieval threshold values of –1.5 and lower. One is better off delaying retrieval until all instances have had some time to establish their activation and will not be so easily invaded by over-active neighbors. Somewhat surprisingly (and satisfyingly), the retrieval threshold value of –1.0 that was chosen to correspond to the human learning curve, especially for problem type 6, also turns out to be optimal in terms of providing the best long-term performance, i.e. lowest number of errors. This echoes the conclusion reached in (Lebiere, 1998) regarding the influence of various parameters on the learning of arithmetic facts through years of studying and experience. This suggests that perhaps the human cognitive architecture is even more flexible that previously thought in adapting its mechanisms to provide optimum long-term performance.

Figure 22 presents the variations in performance as a function of the activation noise *S*:

<Insert Figure 22 here>

Different noise values seem to provide best performance at different stages of training. For instance, a high noise value (e.g. 0.5) is best in the first handful of blocks because it

increases the probability of retrieving anything rather than deciding randomly, while a very low noise value (e.g. 0.1) is best after a lot of training, i.e. dozen blocks, because it reduces the probability that stochastic activation variations will lead to an error of commission. Intermediate values, such as the default value of 0.25, provide best performance for intermediate amounts of training. This suggests that a truly optimal architecture would start with a high noise associated to new knowledge structures that would gradually decrease with practice. Lebiere (1998) suggested that it would produce a power law of practice for the reduction of commission errors. It is also similar to the technique of simulated annealing used in connectionist algorithms such as the Boltzmann Machine (Ackley, Hinton & Sejnoski, 1985). Finally, Figure 23 presents the probability of errors as a function of the mismatch penalty *MP*:

<Insert Figure 23 here>

A similar pattern to the previous two figures emerges. Overly lax mismatch penalties (e.g. 0.5) lead to a permanently high percentage of errors. However, different values provide best performance for different amounts of training. The default value of 1.5 provides the fastest initial learning among *MP* values that trend toward perfect performance, thereby striking the best balance between the need for initial generalization and later precision in memory retrieval.

Implications for ACT-R modeling

As mentioned previously, it has generally been the long-term approach of the ACT-R modeling community to view the various models developed in the architecture as compatible pieces of human knowledge and skills that could ultimately be integrated back into a whole individual. This presents constraints and opportunities that provide strong guidance to the enterprise of developing models within the framework of a unified theory of cognition. One opportunity is the potential ability of reusing previous models and therefore be able to build increasingly complex models out of model libraries, in a manner similar to software engineering practices. One constraint is the need to be compatible, in both parameters and knowledge representation, with previously developed models. As previously discussed, we leveraged this methodology in developing this model, and its parameters and representations do reflect the consensus of the ACT-R community. In turn, this model suggests some new guidelines, practices and extensions for future models.

One such guideline is the adoption of exponentially decreasing similarity metrics for continuous quantities. Past models have not been strongly sensitive to the specific shape of the similarity function as long as it remained monotonically decreasing with distance, but the generalization test of Experiment 2 provided a strong constraint in that regard, that seems retrospectively quite natural. Exponentially decreasing similarities will generally result in the accuracy of partial matching to decrease with the distance from known instances, a result that intuitively seems to hold in general fashion.

Another implication lies in the use of unit tasks and their implications. The concept of unit task is crucial to the functional organization of our model, but it is also relevant in other dimensions. One suggestion is that the attended tags associated with perceptual scanning should expire at the end of the associated unit task, which would provide a more natural limit on the growth of those tags than artificial upper bounds. Another implication of unit tasks is on their use in determining cognitive workload, a methodology that could be applied to any other model and provide a connection with a large Human Factors literature in which that concept plays a fundamental role.

A final general recommendation would concern the cognitive modeling enterprise in general. While quantitatively fitting model to data is a central tenet of the field, there is such a thing as too much of a good thing. The dangers of overfitting model to data are well-known to machine learning practitioners, and most of them might be applicable to model development. Given the pervasive variability of human behavior and the need for the efficient, affordable development of cognitive models, it might well be worth adopting the 80/20 rule as a guiding principle of cognitive modeling.

Summary of Questions and Answers

This section presents the ACT-R architecture of cognition, the methodology used in developing models and its account of individual differences, cognitive workload, multi-tasking and categorization.

How is cognition represented in your system?

Cognition is represented in terms of a computational architecture that implements a unified theory of cognition (Newell, 1990). While unified, the architecture is highly modular and includes separate modules for procedural skill, long-term declarative memory, the current context (a.k.a. goal), and perceptual/motor systems including visual, manual, auditory and speech (Anderson et al, submitted).[3] The latter modules communicate through limited buffers with the central production system. All modules operate in parallel but are internally serial, as is their communication through the buffer system. The central production system represents procedural skill in the form of production rules. Knowledge in declarative memory (as well as the other modules and buffers) is represented in the form of structured chunks. Rules and chunks, as well as the operations of the other modules, are strongly limited in their complexity, i.e. the "Atomic Components of Thought" (Anderson & Lebiere, 1998). While rules and chunks are

[3] As described previously, for practical reasons we used a previous version of the architecture without perceptual/motor modules. Instead we estimated compatible latency parameters for the production rules corresponding to perceptual/motor actions.

represented symbolically to capture the sequential, structured nature of cognition, their characteristics are determined by associated subsymbolic quantities that endow them with "soft" qualities such as adaptivity, similarity-based generalization and stochasticity. Production rules are selected according to their utility and chunks are retrieved from memory according to their activation, both of which reflect the history of those structures. All components of the model, including rules and chunks and their subsymbolic parameters, are learnable by the architecture.

What is your modeling methodology?

Our modeling methodology is based on emphasizing the power and constraints of the ACT-R architecture. The basic methodology is to create the most natural and effective model of the task given the architecture, i.e. a model that respects rather than fights the constraints of the architecture and naturally leverages its mechanisms. The model relies naturally on fundamental features of the architecture, such as memory, for a broad range of purposes from incidental learning to concept formation. The central organizing construct to guide structured cognition is the concept of goal. Goals correspond well to the concept of unit task in human-computer interaction (Card, Moran and Newell, 1983). Complex models are organized around a set of goal types, each with the skills needed to solve them in the form of production rules. Being able to add and remove goal types modularly provides both a tractable way to author complex models, as well as a theory of skill compositionality.

What role does parameter tuning play?

Parameter tuning plays a limited role in ACT-R model development. Some degree of parameter flexibility is required of any cognitive model because of the variety of ways that cognition can be applied to solving a task, and of the differences between individuals. However, parameter tuning must be limited and principled to address concerns that models can account for anything (Roberts and Pashler, 2001) and to provide actual predictiveness. Architectural parameters should be fixed across models (modulo individual differences) because they represent a cognitive constant. Parameters associated with knowledge structures should be learned or set according to reasonable principles (again allowing for individual differences). Knowledge structures constituting the model, i.e. chunks and productions, can be viewed as parameters themselves (Baker, Corbett & Koedinger, 2003). Therefore, as described previously, they should also be learned or set to reflect the natural way for the architecture to solve the problem rather than specially engineered to fit the data.

What is your account of individual differences?

The ACT-R architecture provides a number of accounts of individual differences. The first source of variation in individual differences is simply noise, especially when interacting with a complex dynamic environment, as demonstrated in our results for experiment 1. Stochasticity is a component of every subsymbolic mechanism, including activation, utility and latency computations, which in turn determine every cognitive step including production rule firing and memory retrieval. The second source of individual

differences are changes in architectural parameters that account for variations in fundamental abilities such as working memory (Lovett, Reder & Lebiere, 1999), psychomotor speed and emotions (Ritter et al, 2003). Once estimated, an individual's parameters can be applied to a model of any task to obtain predictions of that individual performance on that particular task. The final source of individual differences is variations in knowledge structures (chunks and production rules) and their associated parameters. Because those variations can be extremely complex and task-specific, it is the hardest source of individual differences for which to derive a consistent account.

What is your account of cognitive workload?

Cognitive workload is defined as a function of the operations of the architecture. Certain goals (unit tasks) involving external manipulations and interruptions are defined as critical. The measure of cognitive workload is the ratio of time spent solving those goals to the total time on task. A similar but finer-grained measure of workload focused on atomic cognitive, perceptual and motor actions might be defined in a manner similar to the BOLD response in fMRI experiments but the data in this task did not address this level of detail. A single measure of workload is provided, but it could easily be defined in a modality-specific way by directly exploiting the modular nature of the architecture, basically defining a workload dimension per module in line with workload theories such as multiple resource theory (Wickens, 1992). Mechanisms by which subjects estimate workload are not specified, but could originate in a mechanism for aggregate retrievals of past goals called blending (Lebiere, 1999). In this model, our assumption was that

performance determined workload, but the model could be augmented to allow workload to determine strategy, and thus performance.

What is your account of multi-tasking?

Structured cognition in ACT-R is organized around the concept of goal. However, production rules can match inputs from any number of modules, especially perceptual buffers, to provide reactive as well as goal-driven behavior. Detection of a perceptual event can lead to a cognitively controlled goal switching. After the external event has been handled, the goal can be switched back to the original one, or cognition can continue on another path. Switching back and forth between goals can be accomplished simply by retrieving previous goals from memory. Or multiple tasks can be accomplished concurrently by combining their goal representations through extensive training (e.g. Byrne & Anderson, 1998). However, the architecture imposes constraints on multi-tasking: the former solution requires lengthy and uncertain goal retrievals while the latter will lead to a slowdown in cognitive operations because of a diffusion in spreading activation.

What is your account of categorization?

Categorization is not a primitive function of the architecture but rather depends upon more basic mechanisms. The initial basis of categorization is memory, specifically the identification of a stimulus by the retrieval of a similar instance from declarative

memory. Similarity-based partial matching provides generalization and the gradual emergence of soft categories. Explicit categorization rules can also be formulated, which are in turn compiled into production rules.[4] Production utility learning can then be used to select between competing categorization rules and instances. Therefore, while categorization is not an ACT-R primitive, architectural constraints provide limits on categorization performance through underlying mechanisms like memory decay and stochasticity.

Acknowledgments

The author would like to thank John R. Anderson for many helpful suggestions during the course of the project, Dan Bothell for supporting the integration for Experiment 1 and Eric Biefeld for supporting the integration for Experiment 2. This project was supported by grants from the Office of Naval Research.

[4] For the sake of simplicity and efficiency, we did not use in this model the production compilation mechanism but instead encoded directly the kind of production rules that would be created.

References

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985) A learning algorithm for Boltzmann machines. Cognitive Science, 9, 147--169.

Anderson, J. R. (1990). The Adaptive Character of Thought. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Bothell, D. J., Douglass, S. A. & Haimson, C.. (2003) Learning a complex dynamic skill. In Proceedings of the 2003 ACT-R Workshop. Pittsburgh, PA.

Anderson, J. R., & Betz, J. (2001). A hybrid model of categorization. Psychonomic Bulletin and Review, 8, 629-647.

Anderson, J. R., Bothell, D., Lebiere, C. & Matessa, M. (1998). An integrated theory of list memory. Journal of Memory and Language, 38, 341-380.

Anderson, J. R. & Lebiere, C. (1998). The atomic components of thought. Mahwah, NJ: Erlbaum.

Anderson, J. R., Matessa, M., & Lebiere, C. (1997). ACT-R: A theory of higher level cognition and its relation to visual attention. Human Computer Interaction, 12(4), 439-462.

Anderson, J. R. & Schooler, L. J. (1991). Reflections of the environment in memory. Psychological Science, 2, 396-408.

Baker, R. S., Corbett, A. T., & Koedinger, K. R. (2003) Statistical techniques for comparing ACT-R models of cognitive performance. In Proceedings of the 2003 ACT-R Workshop. Pittsburgh, PA.

Bracht, J., Lebiere, C., & Wallach, D. (1998). On the need of cognitive game theory: ACT-R in experimental games with unique mixed strategy equilibria. Paper presented at the Joint Meetings of the Public Choice Society and the Economic Science Association, New Orleans, LA.

Byrne, M.D., & Anderson, J.R. (1998). Perception and Action. In J. R. Anderson & C. Lebiere (Eds.) The Atomic Components of Thought. Mahwah, NJ: Erlbaum.

Byrne, M. D., & Anderson, J. R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing. Psychological Review, 108, 847-869.

Card, S. K., Moran, T. P., & Newell, A. (1983). The Psychology of Human Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Corbett, A. T, Anderson, J. R., & O'Brien, A. T. (1995). Student modeling in the ACT Programming Tutor. In P. Nichols, S. Chipman & B. Brennan (Eds.), Cognitively diagnostic assessment (pp. 19-41). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gonzalez, C., Lerch, F. J., & Lebiere, C. (2003). Instance-based learning in real-time dynamic decision making. Cognitive Science.

Lebiere, C. (1998). The dynamics of cognition: An ACT-R model of cognitive arithmetic. Ph.D. Dissertation. CMU Computer Science Dept Technical Report CMU-CS-98-186.

Pittsburgh,PA. Available at http://reports-archive.adm.cs.cmu.edu/.

Lebiere, C. (1999). The dynamics of cognitive arithmetic. Kognitionswissenschaft [Journal of the German Cognitive Science Society] Special issue on cognitive

modelling and cognitive architectures, D. Wallach & H. A. Simon (eds.)., 8 (1), 5-19.

Lebiere, C., & Shang, J. (2002). Modeling group decision making in the ACT-R cognitive architecture. In Proceedings of the 2002 Computational Social and Organizational Science (CASOS). June 21-23, Pittsburgh, PA.

Lebiere, C., Wallach, D. & Taatgen N. (1998). Implicit and explicit learning in Act-R. In F. E. Ritter & R. Young (Eds.). Proceedings of the 2nd European conference on cognitive modeling, pp. 183-189, Nottingham: Nottingham University Press.

Lebiere, C., & West, R. L. (1999). A dynamic ACT-R model of simple games. In Proceedings of the Twenty-first Conference of the Cognitive Science Society, pp. 296-301. Mahwah, NJ: Erlbaum.

Lebiere, C., & Wallach, D. (2001). Sequence learning in the ACT-R cognitive architecture: Empirical analysis of a hybrid model. In Sun, R. & Giles, L. (Eds.) Sequence Learning: Paradigms, Algorithms, and Applications. Springer LNCS/LNAI, Germany.

Lebiere, C., Wallach, D., & West, R. L. (2000). A memory-based account of the prisoner's dilemma and other 2x2 games. In Proceedings of International Conference on Cognitive Modeling 2000, pp. 185-193. NL: Universal Press.

Lee, F. J. & Anderson, J. R. (2001). Does learning of a complex task have to be complex? A study in learning decomposition. Cognitive Psychology, 42(3), 267-316.

Lerch, F. J., Gonzalez, C., & Lebiere, C. (1999). Learning under high cognitive workload. In Proceedings of the Twenty-first Conference of the Cognitive Science Society, pp. 302-307. Mahwah, NJ: Erlbaum.

Logan, G. D. (1988). Toward an instance theory of automatization. Psychological Review, 95, 492-527.

Lovett, M. C., Reder, L. M., & Lebiere, C. (1997). Modeling individual differences in a digit working memory task. In Proceedings of the Nineteenth Conference of the Cognitive Science Society, pp. 460-465. Mahwah, NJ: Erlbaum.

Lovett, M. C., Reder, L. M., & Lebiere, C. (1999). Modeling working memory in a unified architecture: An ACT-R perspective. In Miyake, A. & Shah, P. (Eds.) Models of Working Memory: Mechanisms of Active Maintenance and Executive Control. New York: Cambridge University Press.

Newell, A. (1990) Unified Theories of Cognition. Cambridge, MA: Cambridge University Press.

Newell, A. & Rosenbloom, P.S. (1981). Mechanisms of skill acquisition and the power law of practice. In J.R. Anderson (Ed.).Cognitive skills and their acquisition (pp. 1-56). Hillsdale, LEA.

Ritter, F. E., et al (2003). Using cognitive modeling to study behavior moderators: pre-task appraisal and anxiety. In Proceedings of the 2003 ACT-R Workshop. Pittsburgh, PA.

Roberts, S., & Pashler, H. (2000). How persuasive is a good fit? A comment on theory testing. Psychological Review, 107, 358-367.

Rubin, D.C. & Wenzel, A.E. (1990). One hundred years of forgetting: A quantitative description of retention. Psychological Review, 103, 734-760.

Sanner, S., Anderson, J. R., Lebiere, C., & Lovett, M. C. (2000). Achieving efficient and cognitively plausible learning in Backgammon. Proceedings of The Seventeenth

International Conference on Machine Learning. San Francisco: Morgan Kaufmann.

Taatgen, N. A. (2003) Variability of behavior in complex skill acquisition. In Proceedings of the 2003 ACT-R Workshop. Pittsburgh, PA.

Taatgen, N.A. & Anderson, J.R. (2002). Why do children learn to say "broke"? A model of learning the past tense without feedback. Cognition, 86(2), 123-155.

Wallach, D., & Lebiere, C. (2002). On the role of instances in complex skill acquisition. In Proceedings of the 43rd Conference of the German Psychological Association.

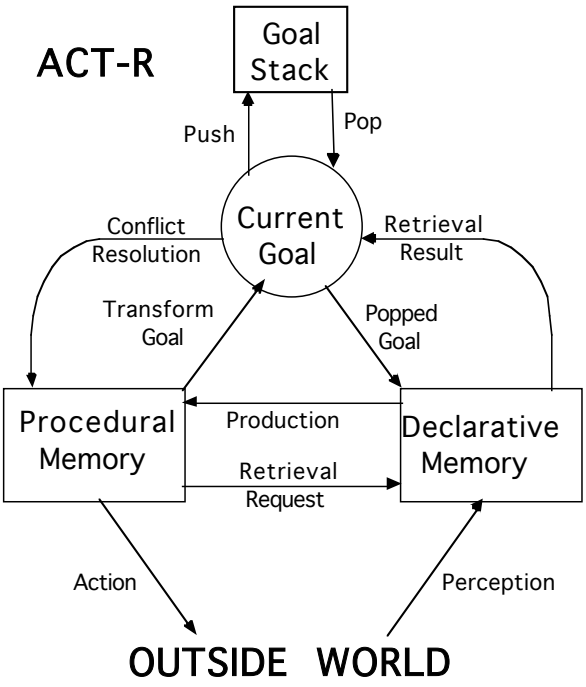
Wallach, D. & Lebiere, C. (2003). Conscious and unconscious knowledge: Mapping to the symbolic and subsymbolic levels of a hybrid architecture. In Jimenez, L. (Ed.) Attention and Implicit Learning. Amsterdam, Netherlands: John Benjamins Publishing Company.

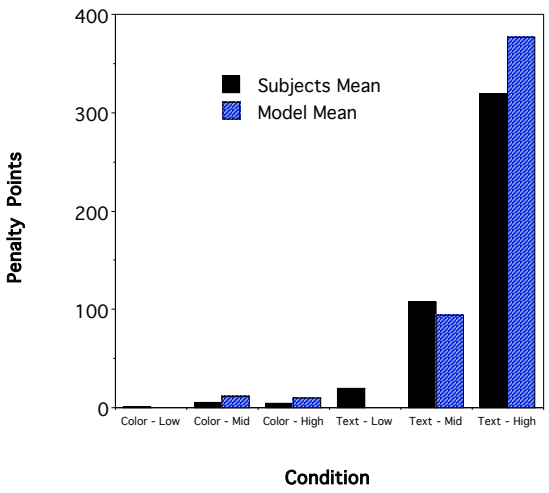
West, R. L., & Lebiere, C. (2001). Simple games as dynamic, coupled systems: Randomness and other emergent properties. Journal of Cognitive Systems Research, 1(4), 221-239.

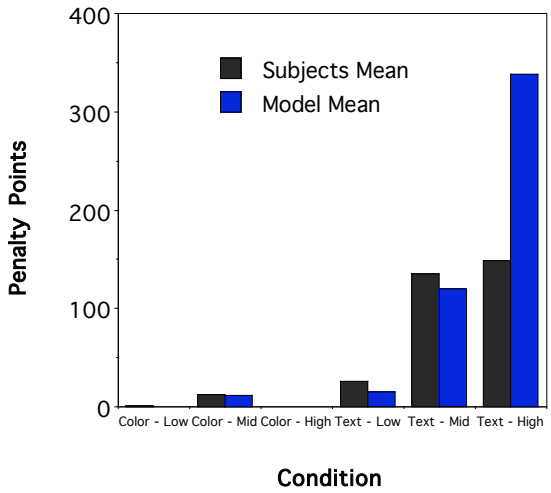
Whalen, J. (1996). The influence of the semantic representations of numerals on arithmetic fact retrieval. Unpublished dissertation.

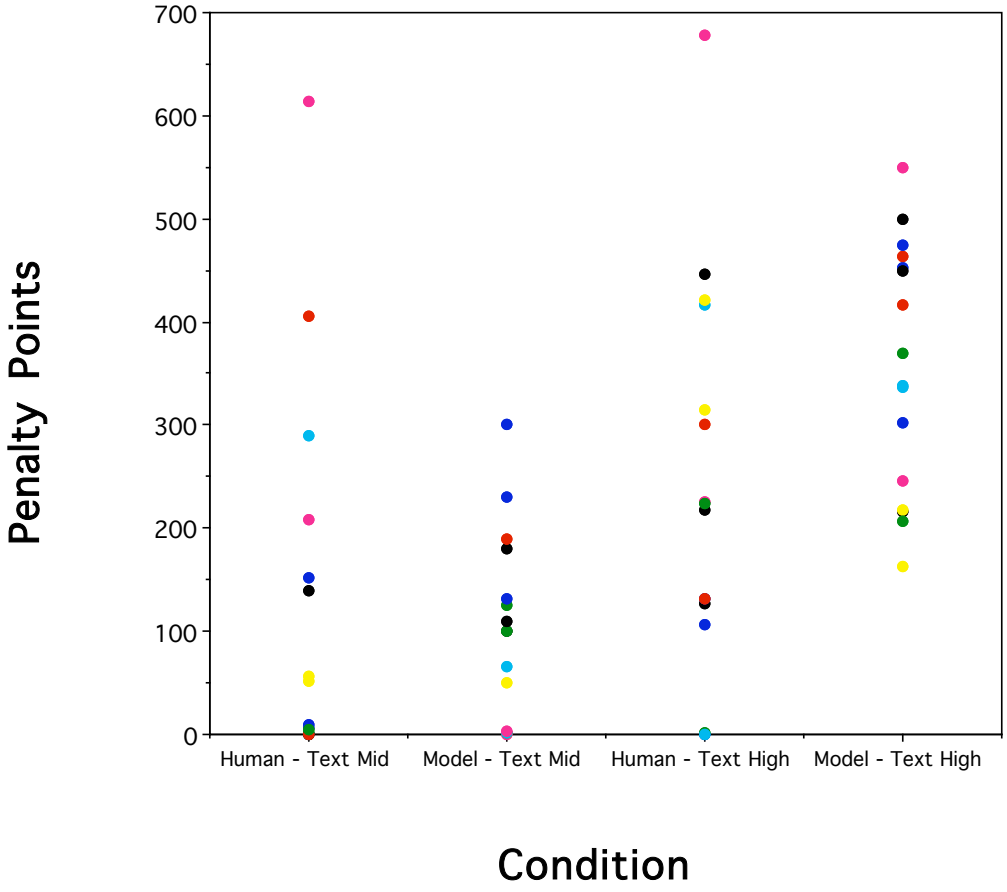
Wickens, C. D. (1992). Engineering Psychology and Human Performance. New York, New York: Harper Collins.

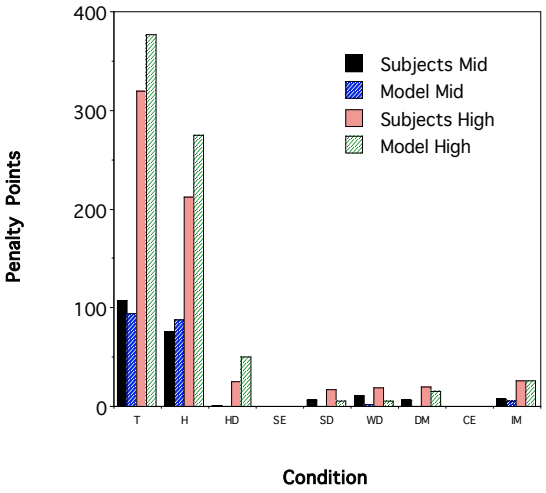
Figures

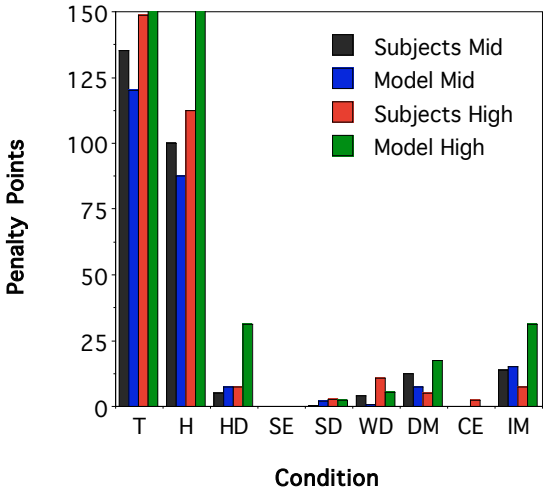

ACT-R
Goal Stack
Push
Pop
Current Goal
Conflict Resolution
Retrieval Result
Transform Goal
Popped Goal
Procedural Memory
Production
Declarative Memory
Retrieval Request
Action
Perception
OUTSIDE WORLD

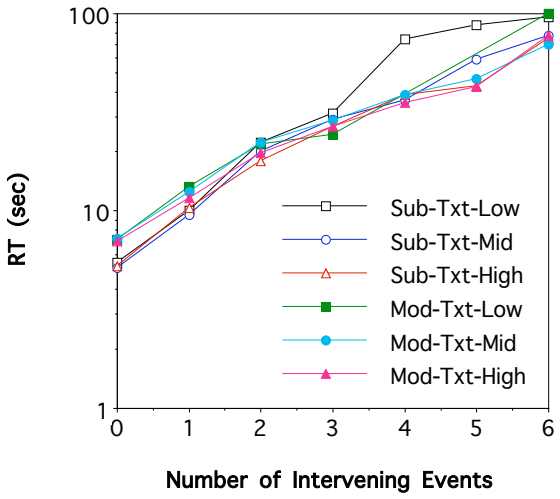

400
300
200
100
0
Penalty Points
Subjects Mean
Model Mean
Color - Low
Color - Mid
Color - High
Text - Low
Text - Mid
Text - High
Condition

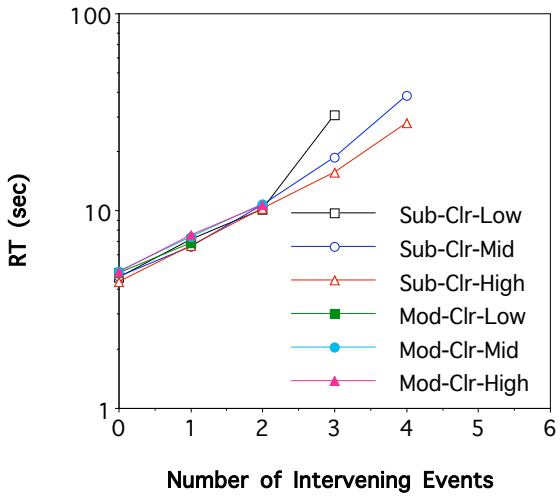

400
300
200
100
0
Penalty Points
Subjects Mean
Model Mean
Color - Low
Color - Mid
Color - High
Text - Low
Text - Mid
Text - High
Condition

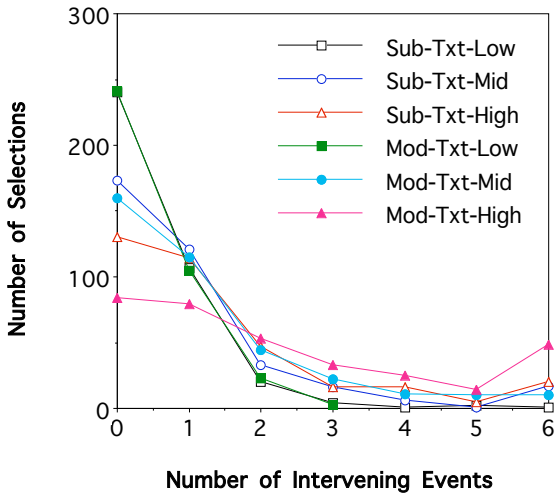

Penalty Points
700
600
500
400
300
200
100
0
Human - Text Mid
Model - Text Mid
Human - Text High
Model - Text High
Condition

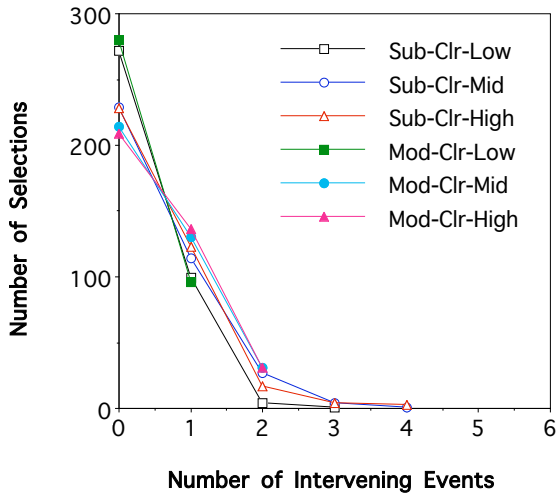

Penalty Points
400
300
200
100
0
Subjects Mid
Model Mid
Subjects High
Model High
T
H
HD
SE
SD
WD
DM
CE
IM
Condition

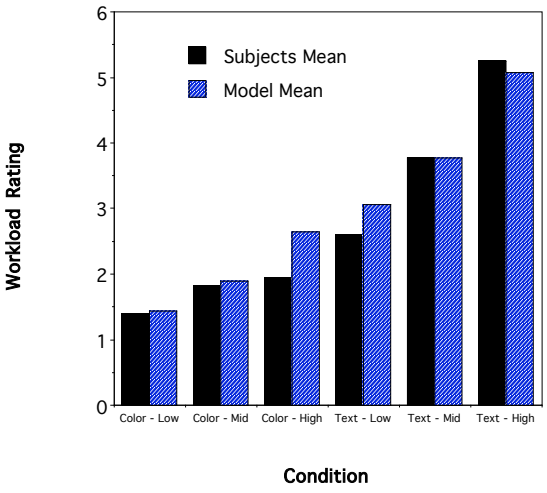

Penalty Points
150
125
100
75
50
25
0
Subjects Mid
Model Mid
Subjects High
Model High
T
H
HD
SE
SD
WD
DM
CE
IM
Condition

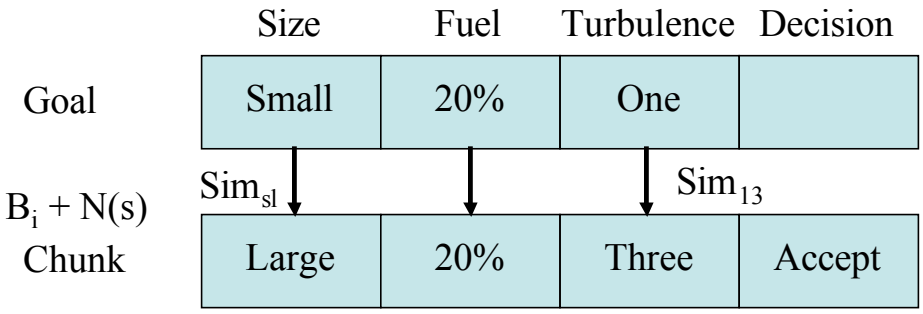

100
10
1
RT (sec)
0
1
2
3
4
5
6
Number of Intervening Events
Sub-Txt-Low
Sub-Txt-Mid
Sub-Txt-High
Mod-Txt-Low
Mod-Txt-Mid
Mod-Txt-High

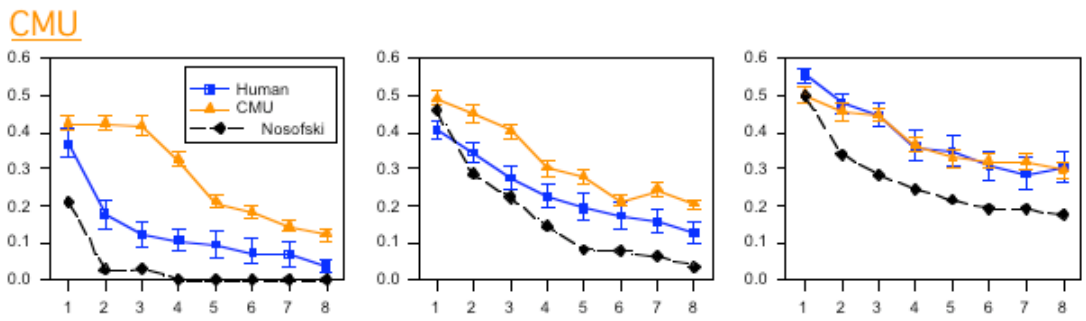

100
10
1
RT (sec)
0
1
2
3
4
5
6
Number of Intervening Events
Sub-Clr-Low
Sub-Clr-Mid
Sub-Clr-High
Mod-Clr-Low
Mod-Clr-Mid
Mod-Clr-High

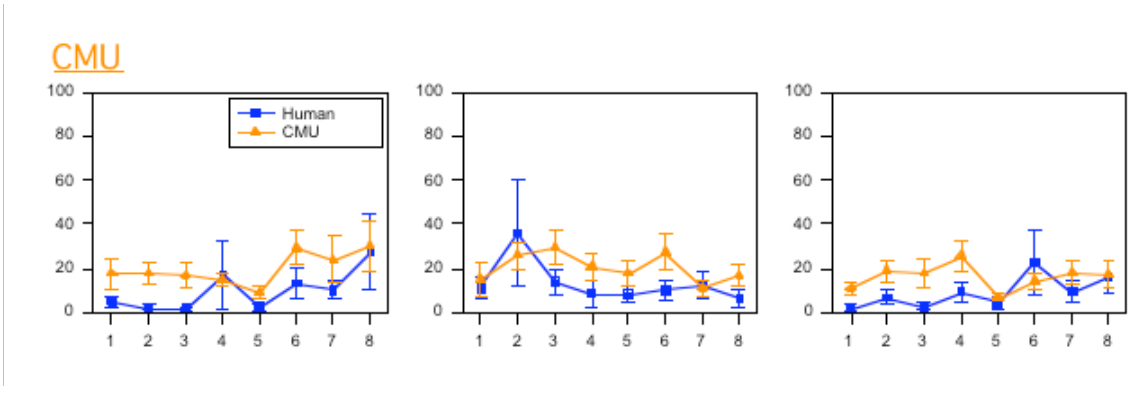

300
200
100
0
0
1
2
3
4
5
6
Number of Selections
Number of Intervening Events
Sub-Txt-Low
Sub-Txt-Mid
Sub-Txt-High
Mod-Txt-Low
Mod-Txt-Mid
Mod-Txt-High

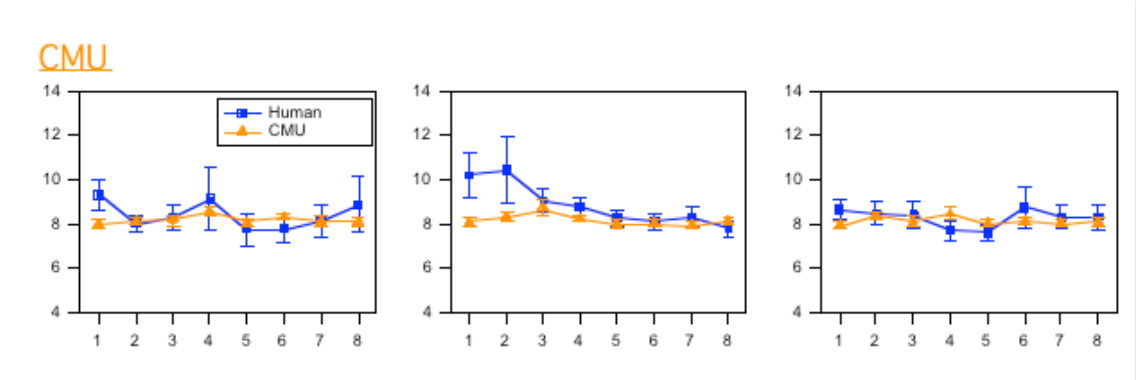

300
200
100
0
0
1
2
3
4
5
6
Number of Selections
Number of Intervening Events
Sub-Clr-Low
Sub-Clr-Mid
Sub-Clr-High
Mod-Clr-Low
Mod-Clr-Mid
Mod-Clr-High

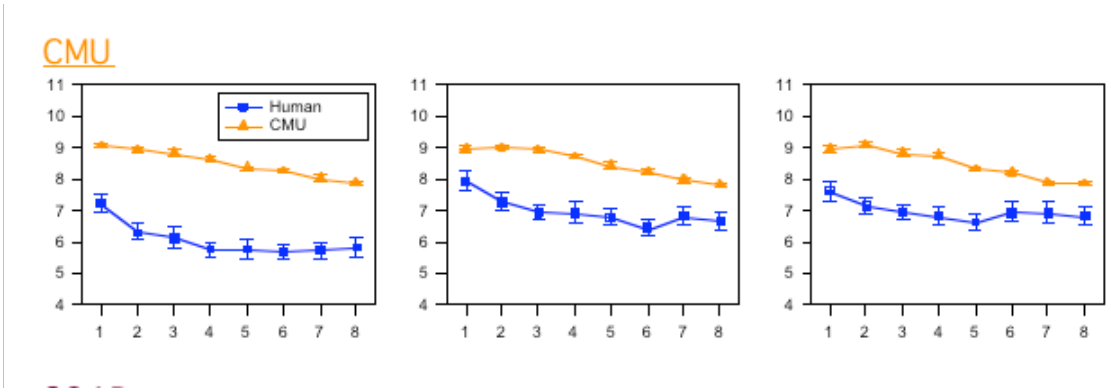

Subjects Mean
Model Mean
Workload Rating
0
1
2
3
4
5
6
Color - Low
Color - Mid
Color - High
Text - Low
Text - Mid
Text - High
Condition

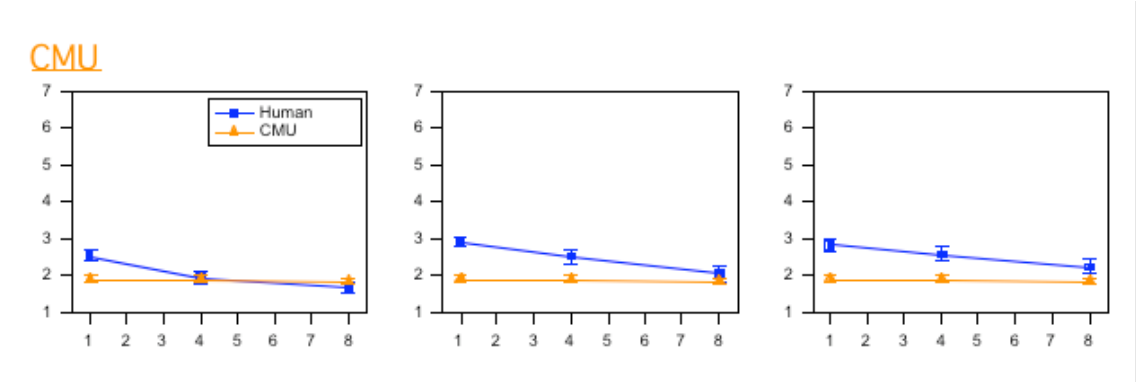

Size
Fuel
Turbulence
Decision
Goal
Small
20%
One
Sim_sl
Sim_13
B_i + N(s)
Chunk
Large
20%
Three
Accept

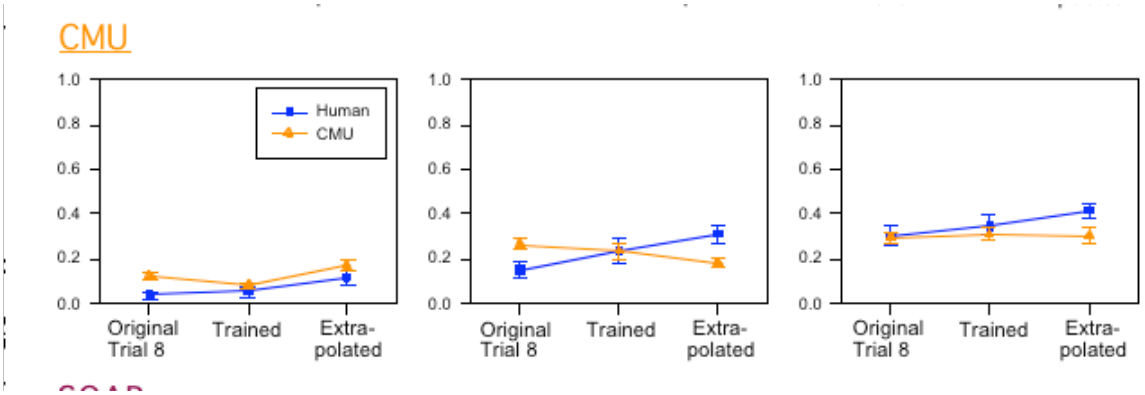

CMU
Human
CMU
Nosofski
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
4
5
6
7
8

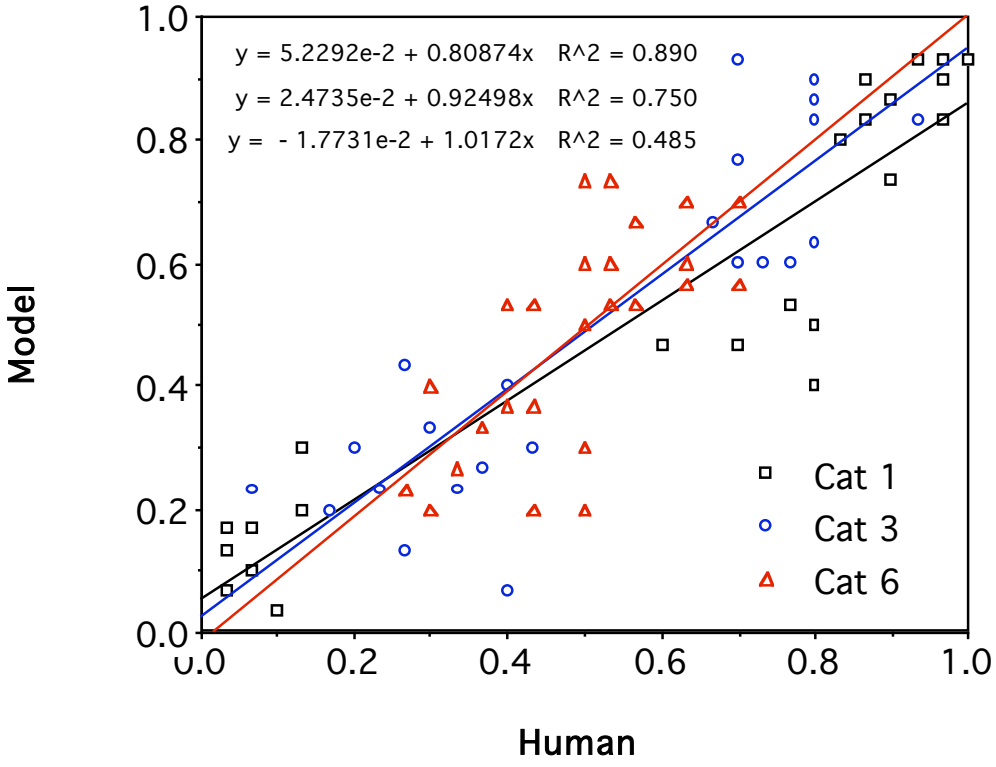

CMU
Human
CMU
100
80
60
40
20
0
1
2
3
4
5
6
7
8


CMU
Human
CMU
14
12
10
8
6
4
1
2
3
4
5
6
7
8

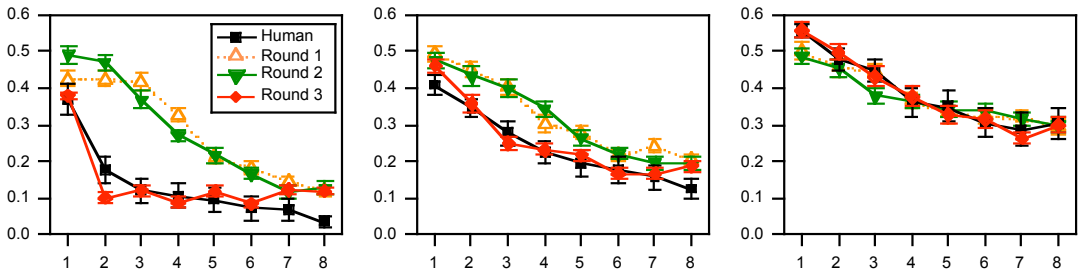

CMU
Human
CMU
11
10
9
8
7
6
5
4
1
2
3
4
5
6
7
8


CMU
Human
CMU
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8

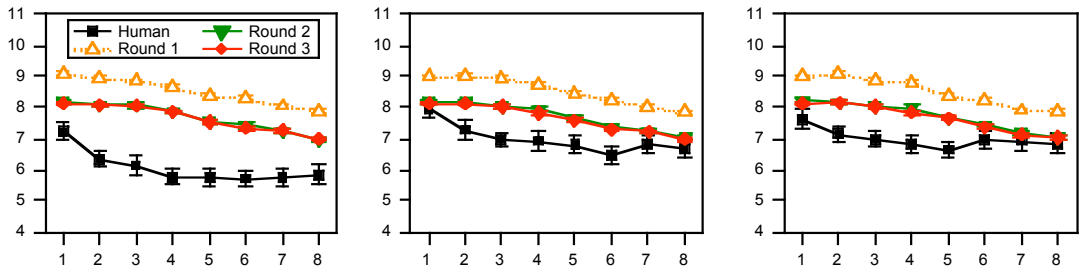

CMU
1.0
0.8
0.6
0.4
0.2
0.0
Human
CMU
Original Trial 8
Trained
Extra-polated
Original Trial 8
Trained
Extra-polated
Original Trial 8
Trained
Extra-polated

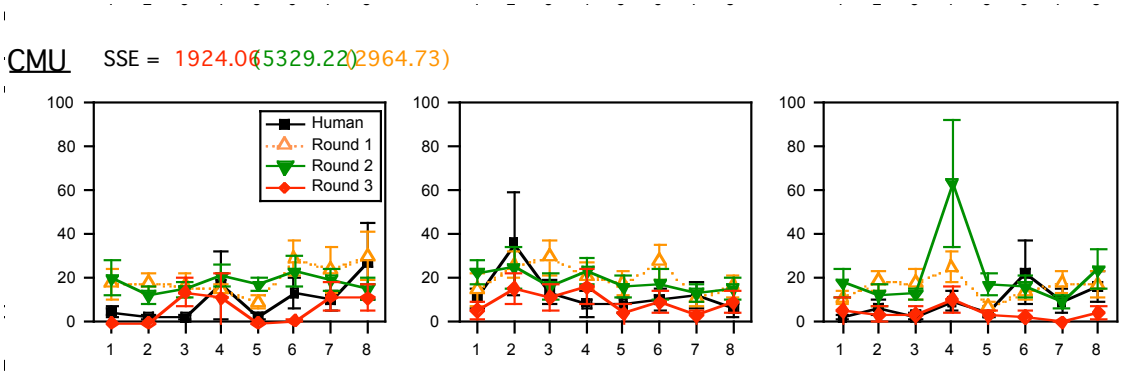

y = 5.2292e-2 + 0.80874x R^2 = 0.890
y = 2.4735e-2 + 0.92498x R^2 = 0.750
y = - 1.7731e-2 + 1.0172x R^2 = 0.485
Cat 1
Cat 3
Cat 6
Model
Human
0.0
0.2
0.4
0.6
0.8
1.0

CMU $G^2 = 7.23(46.61)(49.69)$

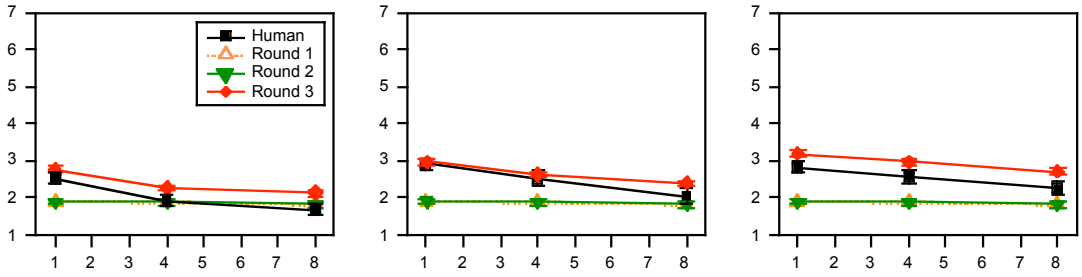


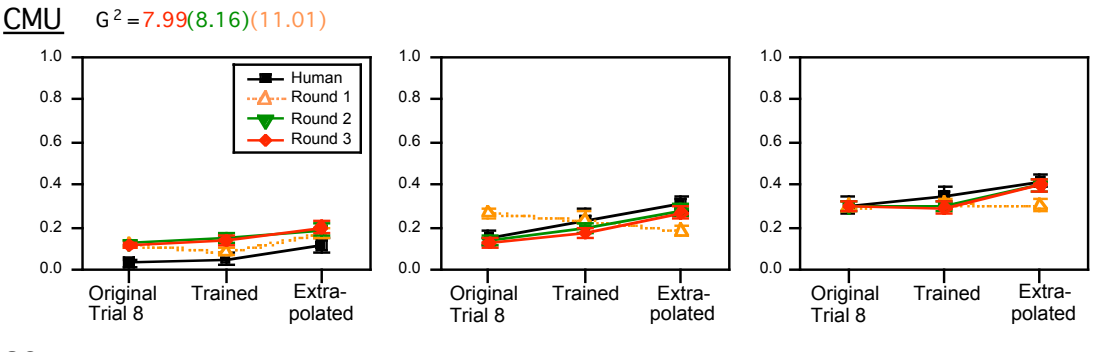
CMU SSE = 30.72 (32.36) (88.67)

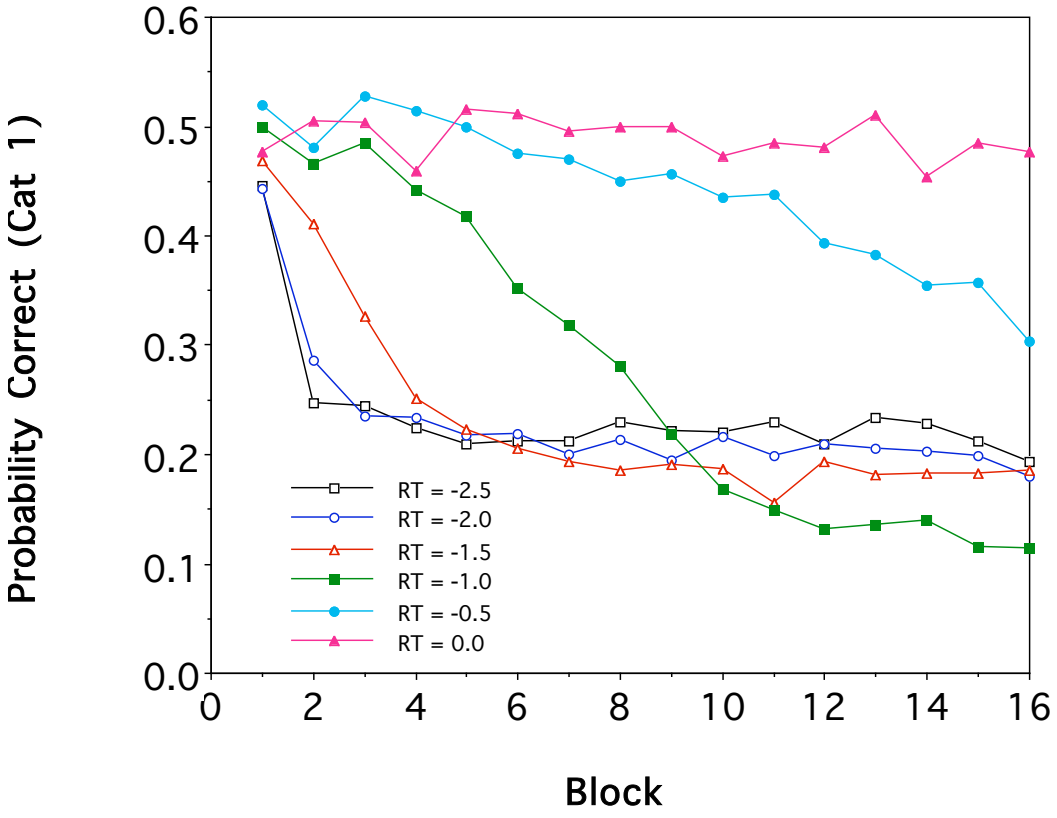


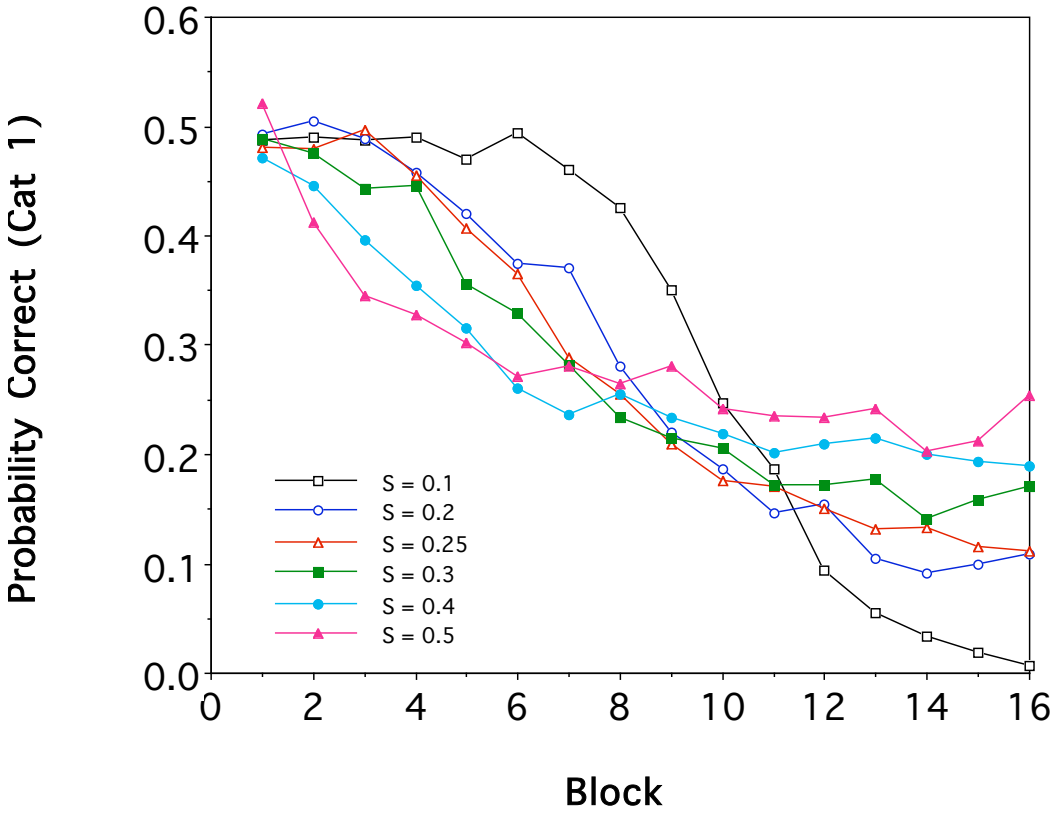

CMU
SSE = 1924.06 (5329.22) (2964.73)
Human
Round 1
Round 2
Round 3
100
80
60
40
20
0
1 2 3 4 5 6 7 8

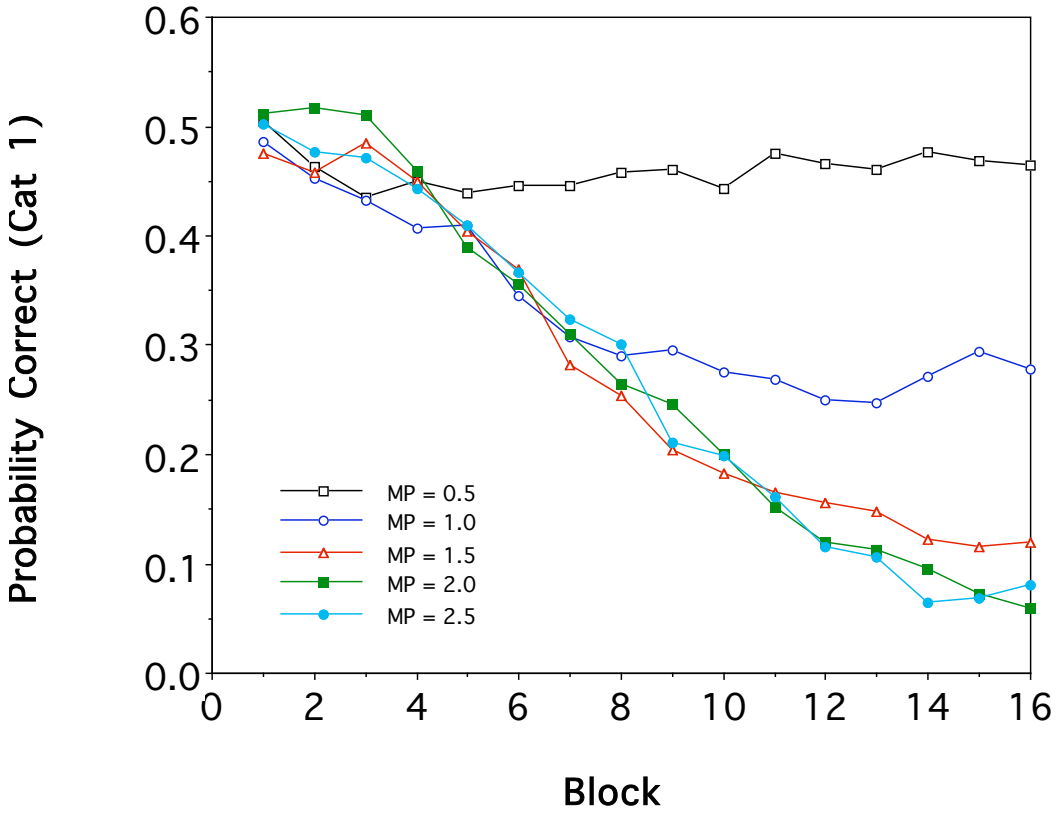
CMU SSE = 1.05(3.34)(3.37)




CMU
G 2 = 7.99(8.16)(11.01)
1.0
0.8
0.6
0.4
0.2
0.0
Human
Round 1
Round 2
Round 3
Original Trial 8
Trained
Extra-polated


Probability Correct (Cat 1)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
12
14
16
Block
RT = -2.5
RT = -2.0
RT = -1.5
RT = -1.0
RT = -0.5
RT = 0.0


Probability Correct (Cat 1)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
12
14
16
Block
S = 0.1
S = 0.2
S = 0.25
S = 0.3
S = 0.4
S = 0.5


Probability Correct (Cat 1)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
12
14
16
Block
MP = 0.5
MP = 1.0
MP = 1.5
MP = 2.0
MP = 2.5