Running Head: CONVERGENCE AND CONSTRAINTS

Convergence and Constraints Revealed in a Qualitative Model Comparison

Christian Lebiere[1], Cleotilde Gonzalez[2]

[1]Psychology Department

[2]Dynamic Decision Making Laboratory

Social and Decision Sciences Department

Carnegie Mellon University

Walter Warwick

MA&D Operation

Alion Science and Technology

Please address all correspondence to:
Christian Lebiere
Psychology Department - Baker Hall 345A
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
Email: cl@cmu.edu
Phone: 412-268-6028

Abstract

We contrasted and compared independently developed computational models of human performance in a common dynamic decision-making task. The task, called Dynamic Stocks and Flows, is simple and tractable enough for laboratory experiments yet exhibits many characteristics of macrocognition. A macrocognitive model was developed using a computational instantiation of Recognition-Primed Decision-Making. A microcognitive model was developed using the ACT-R cognitive architecture. Both models followed an instance-based learning paradigm and displayed striking similarities, including their constraints, limitations, and the key breakthrough that enabled satisfactory (though still short of human-like) performance, suggesting the emergence of a general design pattern. On the basis of this comparison we argue that while some substantive differences remain, microcognitive and macrocognitive approaches provide complementary rather than contradictory accounts of human behavior.

Keywords: Macrocognition, Dynamic Decision Making, ACT-R, RPD, Model comparison

Introduction

As Hoffman and McNeese (this issue) point out, research in macrocognition is often presented as a reaction to so-called microcognitive research and its focus on understanding the "mechanisms of intelligence," theory building, and well-controlled experimentation. On its face, this orientation would seem to put macrocognitive research at odds with the long tradition of computational cognitive modeling that seeks to provide detailed accounts of the invariant mechanisms that capture the general properties of human cognition (Newell, 1990). Furthermore, the fact that such models produce quantitative predictions over a range of measures including, for example, response time, errors, eye movements, and even fMRI activation patterns (Anderson, 2007), often at very fine-grained time scales, might seem to make them even further removed from the macrocognitive emphasis on research on complex tasks over long time intervals and in naturalistic settings.

On deeper inspection, however, this apparent incompatibility turns out to be merely superficial. Although much of the history of computational cognitive modeling has a microcognitive flavor, in more recent years there has been a concern that cognitive models generalize beyond the laboratory to complex, real-world tasks. Similarly, a concern for quantitative predictions does not preclude the explicit representation of macrocognitive functions and processes, as the other manuscripts in this special issue demonstrate. A macrocognitive orientation does, however, put a burden on the computational modeler to explain how those models relate to the macrocognitive functions and processes they purport to represent; where the microcognitive modeler can point directly to the computational implementation as a theory of cognition (i.e., as a "mechanism of intelligence"), the macrocognitive modeler has no such

luxury. In fact, accounts of macrocognition are often deliberately underdetermined with respect to mechanism, focusing instead on descriptive accounts of the cognition "in the wild."

But underdetermination need not be taken as evidence for incompatibility. Quite to the contrary, we see a complementary relationship between macrocognitive theories and computational implementations. Macrocognitive theories need candidate mechanisms to be computationally instantiated in order to generate quantitative, testable predictions (a sine qua non of scientific validity). Relying on microcognitive architectures to provide such mechanisms, validated against controlled experiments, is a more promising approach than blinding searching through a very extensive space of possible computational mechanisms. Conversely, microcognitive architectures strongly constrain but do not solely determine performance in complex tasks: they need to be augmented with models that specify how the architectural mechanisms are applied to the task. While those models can have many origins, including empirical sources such as cognitive task analysis (e.g., Chipman, Shraagen, & Shalin, 2000), the limits of knowledge engineering methods are well known (e.g., Studer, Benjamins, & Fensel, 1998) and more systematic approaches are needed. While modeling paradigms can originate from the architecture itself (e.g., Taatgen, Lebiere, & Anderson, 2006), macrocognitive theories constitute another potential source, promising both systematicity and empirical grounding. In this paper we attempt to demonstrate both directions of the complementarity between macrocognitive theory and computational implementation, revealing the constraints, commonalities, and differences that result from the two perspectives.

To do this, we have taken a comparative approach, explaining how two different computational approaches were used to model a task that embodies learning and decision-making in a complex dynamic environment. Although our approach is clearly related to other

model comparison efforts (e.g., Erev et al., submitted; Gluck & Pew, 2005; Lebiere & Bothell, 2004; Lebiere, Archer, Best, & Schunk, 2008) we emphasize qualitative understanding of model development and performance over quantitative measures of goodness-of-fit. The arguments over using "good fits" as evidence for "good theories" have been a matter of recent controversy (e.g., Roberts and Pashler, 2000; Cassimatis, Bello, & Langley, 2008). Furthermore, we emphasize understanding of what *did not* work in the model development process. Making sense of what a model or architecture cannot do or what needs to be added to a model to make it work is valuable not just because it decreases the expectedness of valid prediction, but because it reveals the work that goes into developing an adequate representation and helps us understand the relative contribution of modeler, architecture and parameterization in the generation of those predictions.

We describe below the process by which we went about developing two computational models. The models were developed completely independently by separate modelers with access to the same task information using computational architectures rooted in very different traditions. One model was developed using a "naturalistic" extension of the Micro Saint Sharp task network simulation environment (Warwick, McIlwaine, Hutton, & McDermott, 2001; Warwick & Hutchins, 2004). The other model was developed using an instance-based model of decision-making implemented within the ACT-R cognitive architecture (Anderson & Lebiere, 1998; Gonzalez, Lerch, & Lebiere, 2003; Gonzalez & Lebiere, 2005).

A qualitative comparison such as the one developed here has several advantages. First, by focusing attention on the model development, rather than on just the numerical results of a fitting process, we are reminded that models do far more theoretical work than simply generating predictions. Second, by applying different modeling approaches to a common problem, we

provide a concrete context in which to understand both the computational mechanisms and the effort entailed in developing a model of macrocognitive processes. Even working within a given architecture, a modeler will have significant freedom in representing the knowledge available, the procedural strategies applied to the task and the parameterization of the model components or the architectural mechanisms. Understanding how the modeler proceeds in developing a model goes a long way toward understanding whether the resulting model corresponds in any interesting way to macrocognition. Finally, and most importantly, in this model comparison we will argue that a fair amount of convergence has occurred from two otherwise disparate modeling approaches. Given the fundamental differences between theories of macrocognition and microcognitive architectures, the degree of overlap in the approaches is remarkable. We see the convergence as evidence for a sort of "design pattern" that might prove especially fruitful as researchers continue to explore computational models of macrocognitve processes.

We begin by describing the general method we used for our model comparison. Next, we describe the Dynamic Stocks and Flows (DSF) task and the associated experimental protocol and resulting human performance data. We argue that the task should be germane to those interested in understanding macrocognition. We then describe the two different computational models of human performance in the DSF task, each of which implements an instance-based model of decision making, but comes from widely different theoretical perspectives. For each model, we will provide a detailed description of the underlying architectures and their mechanisms along with a discussion of what went into the process developing the DSF models themselves. Finally, we will conclude with a discussion of the convergence between the two approaches and what this might tell us about macrocognition.

Methodology for Model Development and Comparison

Our model comparison effort was guided by three methodological requirements. First, the modeling effort should focus on human performance rather than the simulation of the task environment or the complexities associated with software integration. Past experience has taught how easily model comparison efforts can be derailed by requirements for modeling or integrating with high-fidelity simulations of the task environment. Second, the comparison should engage the core mechanisms of the respective architectures as directly as possible, with a minimum degree of engineering or data fitting. Finally, the comparison should ultimately reveal deep similarities and differences in the respective approaches and the extent to which they correspond to the theories they purport to implement.

We believe the task environment we describe below satisfies the first and second requirement; it is sufficiently simple and lightweight to be modeled easily but at the same time it is open-ended, to discourage over-parameterization and over-engineering of the model and to test its generalization over a broad range of situations, and it is dynamic, to explore emergent behavior that is not predictable from the task specification. To further discourage over-engineering of the models, we firewalled the development teams from each other and we withheld the human performance data until each team was satisfied with its model performance based solely on a detailed description of the task and one experimental condition. In this way we hoped to highlight the "best modeling practices" supported by the respective architectures without inadvertently introducing a common modeling approach or prejudices about the kinds of behavior the models should produce. Only then did we examine qualitative fits to the data under original experimental condition and, later, another transfer condition. Finally, to ensure that the

comparison was hitting on deep similarities and differences, we made the description of the development processes the focus of the comparison rather than measures of goodness-of-fit. Though these descriptions might strike some as evidence of "flailing," modeling human performance is more than just finding the right kind of representation of the task, as if such representations existed independently waiting only for direct expression in any architecture. Rather, it is finding the right representation *that can be supported by the given architecture*. And it is in this light where the real similarities and differences between architectures are revealed and the correspondence between macrocognitive theory and computational model is most clearly assessed.

The DSF Task

The stock management problem (balancing accumulations through the management of inflows and outflows) is a common process in every-day life that arises at every temporal, spatial, and organizational scale (Cronin, Gonzalez, & Sterman, 2009). At the firm level, for example, the capabilities and competitive advantages arise from the accumulation of resources and knowledge (Dierickx & Cool, 1989; Sterman, 1989). Managers must control their cash flows to maintain adequate stocks of working capital, and production must be adjusted as sales vary to sustain sufficient inventory. A simple stock management problem is the one reduced to its most essential elements: one *stock* (a resource that accumulates or depletes over time) and *flows* that alter the stock (inflows that increases the stock and outflows that decreases the stock).

In the past, researchers investigated the perceptions of simple dynamic systems (Booth Sweeney & Sterman, 2000; Cronin & Gonzalez, 2007; Cronin et al., 2009; Sterman & Booth Sweeney, 2002;). A conclusion from these investigations is that even simple stock problems are unintuitive and difficult, even with a minimal number of variables, and even for highly educated

people with strong technical backgrounds. Understanding and controlling such systems in the real world, where the systems themselves are far more complex, the information is far less certain, and the consequences of action are harder to trace, is more difficult and demands significant skill. Managing stocks is a dynamically complex problem, that requires detecting problems, adapting to changing conditions, and deciding between many alternatives courses of action, thus, reflecting processes directly relevant to macrocognition.

The Dynamic Stocks and Flows (DSF) is a simulation-based tool for studying learning and decision making in the context of simple stock management problems (Dutt & Gonzalez, 2007; Gonzalez & Dutt, 2007). Although the simulated stock problems are simple in the traditional sense (i.e., they have few elements to manage), the DSF is still dynamically complex (Sterman, 2000). The complexity arises from the interaction between the decisions made and the structure of the environment over time.

The DSF represents the essential elements of every dynamic system: a single *stock*, which represents an accumulation of discrete (i.e., units in inventory) or continuous (i.e., water) units, *inflows*, which increase the level of the stock, and *outflows*, which decrease the level of the stock. The goal of this task is to maintain the stock at a particular level or at least within an acceptable range. External inflow and outflow increase or decrease the level of stock, both of which are outside the control of decision makers. Stock levels are also influenced by the user's decisions of inflow and outflow, which increase or decrease the level of the stock and are under the control of the user. Further, the level of the stock at time *t* depends upon the state of the system at the previous time *t-1*, a characteristic of dynamic systems called *interdependency* (Edwards, 1962). Also inherent in dynamic systems are feedback loops, where a variable can affect itself and other variables.

Figure 1 represents the graphical user interface of DSF. The stock is represented graphically as a tank. In this version, the stock represents continuous units of the stock as water in a tank. The markings on the left side of the tank represent the water level in the tank at any instance of time. There are 4 pipes connecting the tank, as shown in Figure 1. Two pipes labeled *User Inflow* and *Environment Inflow* are located on the input side and increase the level of stock in the tank; two pipes labeled *User Outflow* and *Environment Outflow* are located on the output side and decrease the level of stock in the tank.

INSERT FIGURE 1 HERE

The user must set the inflow or outflow rates (*user inflow* or *user outflow*) at each instance of time, typically to compensate for the environmental inflow and outflow that may push the stock away from its desired level. The environmental inflow and outflow are external functions that can be set up by the experimenter. The target level of stock is shown with a red horizontal line with *Goal* mentioned on the right side and also in the *Goal* information box, as shown in the information section next to the *Amount in Tank* box.

The user enters the number of units for inflow and outflow in the decision boxes at the bottom of the screen and hits *Submit*. Then, the Environment Inflow and Outflow makes an effect on the stock. The user receives "feedback" about the number of units of Environment Inflow and Outflow; these appear as numbers in the black boxes during each time period, as shown in Figure 1. Just after the Environment Inflows and Outflows take place, the system causes the user specified inflows and outflows to take place. The number of units of user inflow pops up in a red box next to the User Inflow pipe, while the number of units of user outflow pops

up in a green box next to the User Outflow pipe. The system also provides feedback by presenting all values of the flows that occurred in that time period in the Information section shown in Figure 1.

The user can then begin the next time period (which is shown in a time display box at the top of the tank) by submitting new inflow and outflow values in the decision boxes and pressing *Submit*. Users must do their best to control the water level and maintain it at the targeted value, given the variable nature of the Environment Inflows and Outflows.

Human performance in DSF

Behavioral data collection using DSF has demonstrated that controlling even this simple dynamic system can be challenging for humans. Gonzalez and Dutt have collected a variety of human performance conditions using DSF (Dutt & Gonzalez, 2007; Gonzalez & Dutt, 2007). For example, Dutt and Gonzalez (2007) presented data from an experiment in which participants were asked to maintain the level of water in the tank to 4 gallons or within +/- 0.1 gallons from the goal during all the 100 time periods. The human data came from two conditions defined by the Environment Inflow function: one condition followed an *increasing* linear function and the other condition followed a *decreasing* linear function over trials. The environment outflow was constant and set at 0 Gallons/Time Period. Hence, Environment net flow was equal to Environment Inflow. The initial water level in the tank was fixed in both conditions at 2 gallons. In the increasing condition the Environment Inflow function increased over the course of 100 time periods from 2 to 10 Gallons/Time Period according to the function: 0.08 * (*TimePeriod*) + 2. The decreasing condition was exactly opposite, decreasing from 10 to 2 in decrements of 0.08. In both function there was an equal amount of water flowing into the tank over the course of 100 time periods (which was 604 gallons). The human performance results for the increasing and

decreasing functions in DSF are presented in Figure 2. These were reported and are discussed in Dutt and Gonzalez (2007).

INSERT FIGURE 2 HERE

In the model comparison exercises presented in this paper, we first implemented the linear increasing function and then used the same model to predict the results of the linear decreasing function in Dutt and Gonzalez (2007).

RPD Computational Model of Behavior in the DSF Task

Inspired by Klein's model of the recognition-primed decision-making (Klein 1989, 1993, 1998), we have extended the available "decision types" in the Micro Saint Sharp task network modeling environment to include an "RPD" decision type.[1] The RPD decision type is intended to support the representation of an experience-driven decision-making process where courses of action are the emergent "by-products" of recognition rather than the result of deliberative analysis or the application of rule-based knowledge. In particular, the RPD decision type implements computational analogues for three prominent features of Klein's theoretical model. First, just as the recognition-primed decision model emphasizes the importance of experience over the application of fixed, normative strategies, the RPD decision type depends on the accumulation of experience to shape decision-making performance during a simulation. Second, just as recognition-primed decision-making was presented as an alternative to analogical and case-based models of decision-making[2], recognition in the RPD decision type draws on the entirety of experience rather than focusing on any single past episode. Third, just as the course of action is an immediate "by-product" of situation assessment in Klein's model[3], the RPD decision

type is, at root, a mechanism for learning associations between situations, courses of action, and outcomes such that, with enough experience, the model simply reacts to each new situation with whatever course of action it has come to associate with that situation without engaging in any deliberative, optimizing or rule-based reasoning.

Computationally, our approach extends Hintzman's *multiple-trace memory model* (1984, 1986a, 1986b). The basic idea is to represent a decision maker's long-term memory as a set of episodes, each of which represents the situation that prompted a decision (encoded as a cue vector), the course of action taken (from a fixed set of discrete alternatives), and an outcome measure of that action (either successful or not). Recognition occurs when a new situation (i.e., cue vector) is presented. A "similarity value" is computed between the new situation and the corresponding portion of *each* of the remembered episodes. This value is used to determine the proportional contribution that each and every remembered episode makes to a composite recollection of courses of action taken in the past and their outcomes. The result is a distribution of recognition strengths across the available course of action given the new situation. At this point, the model depends on a fixed selection heuristic (e.g., choosing either the course of action with the greatest recognition strength or performing a weighted random draw across all recognized courses of action weighted with respect to recognition strength), and the selected course of action for that situation is implemented, evaluated, and stored as a new episode in long-term memory for use in the next decision.

While we defer the technical description of the computational mechanisms to Appendix A, there are several features of the approach worth noting here. Decision-making episodes are encoded using bit-strings and similarity values are calculated by taking a dot product between a cue vector representing the current situation and that portion of each remembered bit string that

encodes the cue vectors of past situations. In order to ensure a uniform structure for these cue vectors, the cues that prompt recognition are re-represented internally using discrete enumerations of the ranges of values they can assume. This approach to representing cues and calculating similarity values implicitly supports fuzzy matching between "adjacent" cue values (the closer the value, the more similar the match), but it also has a more serious implication that determining similarity is a purely syntactic process within the RPD decision type; no matter what kinds of cues the internal enumerations encode, similarity is calculated bit-by-bit without any consideration for what those bits represent.

The computational mechanisms that implement the similarity-based recall, the recognition of a crisp course of action from the distribution of recognition strengths, and the accumulation of experience are invariant features of the RPD decision type, but the content of the modeled decision is not. That is, any decision modeled using the RPD decision type in Micro Saint Sharp will use these same mechanisms, but the cues, courses of action, and outcome evaluations of those actions must be defined by the modeler. In most cases, the "structure" of the decision simply falls out from the task. For example, in modeling binary choice in a categorization task, the cues are simply the dimensions along which the stimuli are given, courses of action are just the available categories, and outcomes are determined by whether the membership decision matches the actual membership of the stimulus. The structure of the dynamic stocks and flows task did not lend itself to such an immediate representation of cues, courses of action, or outcomes. Moreover, given our overriding interest in model comparison, we chose to insulate our development efforts as much as possible in the hopes that independent development would highlight differences in the modeling approaches. Toward this end, we not only worked separately of each other, but we also chose not to consult verbal protocols from the

experiment to eliminate the possibility that we would be inadvertently biased toward a common representation of the task. Instead we relied on our subjective intuitions and computational experimentation to guide model development. In the case of the RPD decision type, this led to an iterative development cycle with each stage in the cycle comprising a choice of cues to model, a scheme for transforming those cues into internal representations, and a specification of the discrete courses of action the model would choose among and how those choices would be reinforced.

First, taking what seemed to be the most obvious approach, we modeled the decision as a choice among fixed adjustments to the stock (e.g., decrease-five-gallons, increase-two-gallons, etc.) prompted by two cues, the current stock and the current exogenous inflow. The continuous values of both the current stock and inflow were mapped to enumerations; so, for example, if the current stock was between 2.5 and 5.0 gallons, the RPD cue would assume the value of "about right." Initially, each cue was mapped to a five-valued enumeration. Decisions were positively reinforced whenever they resulted in the current stock that was within a predefined range of the target stock.

Although straightforward and intuitive, this initial representation of decision-making within the DSF task was a failure. More specifically, in order to control the stock, the model had to associate input patterns—i.e., values of the current stock and inflow—with the appropriate course of actions—i.e., reductions in stock to offset the inflows given the current stock; but the model was never able to learn those associations and stock would increase monotonically throughout the trial.

INSERT FIGURE 3 HERE

Initially, we thought that the failure might be rooted in the lack of fine-grained control the model had over the stock. We speculated that, limited to only coarse corrections, the model might be missing the target stock so often as to prevent it from receiving enough reinforcement to learn the appropriate associations between input patterns and outflow actions. Because the model was never forming stable associations between situations and actions it would, by default, end up guessing and failing most of the time. So, on the basis of this speculation, we increased the number of courses of action available to the mode (thereby making each correction more precise). Similarly, we also increased the granularity of the cues (to make them more diagnostic; that is, we mapped real-valued inputs to a larger number of enumerated values). And we even eliminated some courses of action from consideration that we knew would never be appropriate under the particular instance of the DSF task we were modeling. The hope was that having fewer "incorrect" options to consider, the model would perform better.

While these changes prevented the monotonic increase, it was clear that the model was still far from controlling the stock. We then began to reconsider our initial speculation; reversing our thinking completely, we wondered whether the episode structure was too complex to learn in a short span of 100 trials. So we simplified the input by reducing the granularity of the cue enumerations, and even eliminating one cue altogether with the hope that this could make the associations between input and action easier to learn. But this did not work either. Finally, we experimented with different reinforcement strategies and some more general parameter settings

that controlled learning rates within the model. Again, the model was never able to demonstrate any degree of control over the stock.

Our only real breakthrough came when we changed our basic representational strategy. In particular, rather than represent inputs and courses of action in absolute terms of specific inflow amounts, stocks, and outflow adjustments, we defined cues in the relative terms of the degree of difference from desired stock and courses of action as proportional adjustments to the current stock (using the previous inflow as the basis of the proportional adjustment and including a "match" course of action, where the proportion equals 100% of previous inflow). Similarly, decisions were reinforced whenever the total stock was moved closer to the target stock (rather than evaluating the outcome in terms of whether the target stock was within a specified window of the target stock). With these changes, the model was finally able to exhibit some degree of control over the stock. Still, there were quantitative differences in the model performance when compared to the human performance. For example, humans were able to learn to control the stock much more quickly and precisely than the model; but overall, the model exhibited plausibly human-like performance.

INSERT FIGURE 4 HERE

What's more interesting is to reflect on the steps it took to get to this level of performance on this particular task and how our approach in general, having been inspired by a study of macrocognition, compares to the more "microcognitive" approach to instance-based decision making taken in ACT-R. In hindsight, representing the DSF task in the relative terms might seem like it should have been the obvious choice from the beginning. Indeed, from the

perspective of control theory, recognizing differences from target states and reinforcing actions that return the system to that state are old hat. But coming at the DSF task from with a model of recognition-primed decision-making in hand, such representations were far from obvious. We had become accustomed to thinking of decision problems in terms of mapping inputs to outputs, finding the appropriate association between situations and suitable courses of action. It wasn't until very late in this effort that we realized that the DSF would not easily fit that mold and that, given the constantly changing nature of the environment in the DSF, there would *never* be a fixed mapping between situations and actions.[4] Any association between the current stock and the decision to reduce stock by such-and-such a fixed amount (or proportion) would eventually become obsolete as the rate of inflow changed; that is, any association learned between a situation and a course of action would eventually need to be unlearned and another, newer association learned anew as the adjustment required in absolute terms would change over time. Moreover, it is worth noting that the architecture of the RPD decision type forces us to model control in essentially categorical terms. Once the modeler enumerates the cues and courses of action, model performance depends on learning the purely syntactic associations between patterns of input cues and enumerated courses of action. Even though the cues and courses of action in this case represent quantities, the internal representations are non-numeric in the sense that there are no internal mechanisms to support mathematical or other similarly rich manipulations of them; the model cannot average past inflows, compute differences, or estimate the impact of particular adjustments.

In this light, the fact that a recognitional model of decision-making can exhibit some degree of human-like control over the DSF task might seem remarkable. Then again, we might also take this as evidence that the model is doing something like the actual human subjects, who

also have a difficult time with the task. The question now is whether the performance of the model is just an artifact of a well-chosen problem representation or something deeper. This question came into sharp relief when we ran the very same model we had developed for the increasing inflow condition of the DSF task under a decreasing condition. Given that nothing about the inflow condition had been directly represented in the RPD decision, we expected *a-priori* that the model performance would be qualitatively similar, under both conditions. That is not what we found (see Figure 4). While the RPD model again exhibits qualitative similarity to the human performance data, including the gross overshooting of the target outset early in the trials, overall it seems to do a much better job controlling the stock in the decreasing condition. In particular, it lacks the periods of un-learning and relearning that are evident in the increasing condition. Given that the internal representations are identical, it would seem that only the nature of the task environment could account for the difference, namely, that in the decreasing inflow conditions, new inflows are continually perturbing the stock less and less and thus whatever association was successful before is likely to be successful later. Again, it is not clear whether credit is due to a useful representation of the problem, or a happy accident of the task environment. To address that question we now turn to an ACT-R model of the DSF task. As we will discuss below, the ACT-R model is able to take advantage of a much more expressive representation of the problem to improve the quality of its control but it shares many of the same, more general architectural features of an instance-based, recognitional model of decision-making.

ACT-R Computational Model of Behavior in the DSF Task

ACT-R's primary architectural commitments are two-fold. At the organizational level (see Figure 5), the architecture is composed of a set of modules, including perceptual (visual),

motor (manual), and declarative memory modules (as well as self-standing goal and imaginal (problem) buffers), coordinated by the procedural (production rules) module through limited-capacity buffers. Each processing step within a module is massively parallel (e.g., all production rules in the procedural module are matched at once, as are chunks of information in the declarative module) while communication between modules is serial and asynchronous (e.g., only one request for information retrieval can be sent to declarative memory at a time, and a single chunk will be returned through the retrieval buffer whenever the retrieval is completed). Activity in the modules has been correlated with fMRI BOLD response in specific brain regions, bringing to bear neuroscience constraints on the architectural organization (Anderson, 2007). While we will not discuss the neuroscience underpinnings of cognitive performance any further, it does illustrate how cognitive architectures can integrate constraints from other levels of description, from neural models to macrocognitive models (Jilk, Lebiere, O'Reilly, & Anderson, 2008). The second level of architectural commitments concerns the representations and processes taking place in each module, and in particular the declarative memory module and the procedural module. ACT-R's approach is a hybrid combination of a simple, constrained symbolic representation (chunks in declarative memory, production rules in the procedural module) together with subsymbolic selection mechanisms that adapt to the statistical structure of the task and its environment. The former underlies our ability to perform almost any task and quickly learn novel combinations of knowledge while the latter captures the soft, adaptive nature of human performance including both its abilities (e.g., generalization) and its limitations (e.g., forgetting). A tight integration of those two very different types of abilities is necessary to account for the full range of human cognition (Anderson & Lebiere, 2003).

INSERT FIGURE 5 HERE

A number of modeling paradigms have emerged to apply the ACT-R architecture to various classes of problems (e.g., Taatgen, Lebiere, & Anderson, 2006). In our model, we adopted the instance-based learning approach that we previously applied to a broad range of decision-making tasks (e.g., Gonzalez et al., 2003; Gonzalez & Lebiere, 2005; Wallach & Lebiere, 2003). Modeling a broad range of tasks using not only the same architectural mechanisms but also the same techniques and parameters is a key unifying attribute of cognitive architectures. It also imposes stronger constraints on the model's predictiveness than a model developed and parameterized tabula rasa to fit the task and data. The logic of the model is straightforward: extract a small number of decision attributes from the full problem representation, set up a goal to make a decision based on those attributes, and attempt to retrieve and generalize a previous decision made in a similar situation. To bootstrap the system, we used a first-order heuristic rule that attempted to resolve the discrepancy between the current and desired water levels. This heuristic is both simple and optimal in the absence of knowledge regarding the future external inputs and outputs, as is the case for a subject starting the experiment. To avoid introducing an additional parameter arbitrating between the rule-based and instance-based decision strategies, we decided to initialize the system with a few instances extracted from that rule (e.g., if the stock is 5 gallons less than the target level, set the inflow level to 5) rather than represent the rule itself, thus preserving the purely instance-based nature of the model.

Given this basic task strategy, the structure and mechanism of the cognitive architecture put very strong constraints upon the resulting performance. The production rules to implement the instance-based strategy are few and straightforward (basically just a couple of rules to set up the decision goal, request the instance-based retrieval, and perform the action), so the key determinant of performance is the retrieval of past instances from declarative memory. Other than the initial chunks extracted from the heuristic rule to bootstrap the process, each memory instance chunk is created automatically from each accomplished decision-making goal. The activation of each chunk, reflecting processes such as recency and decay, will determine its availability. In particular, more recent memory items will be more salient and thus more recent experience favored, a significant difference from the RPD model. However, from there the processes are substantially similar. Activation is modulated by the degree of match between the current situation and the memory chunk as reflected in the similarity of their values in a process called partial matching. The value(s) returned by the memory retrieval process do not correspond to those of a single memory chunk but instead reflect a blending process (Lebiere, 1999) that returns the best consensus value over all memory chunks, weighted by their probability of retrieval according to a softmax process. Details of this process and equations are included in Appendix B.

Given the basic instance-based strategy and those architectural constraints, the one remaining degree of freedom in defining the model is the choice of relevant attributes used in making and representing each decision. That is not a surprise since the relevant attributes are not known to subjects either, and different subjects (or the same subject at different times) might well pay attention to different attributes. As for the RPD model, we were surprised at how rich the space of choices was compared to past experience with other similarly simple control

systems. As a reminder, the ACT-R model was developed by a different modeler, independently and in parallel to the RPD model, and the representation explorations for each model were not in any way informed by those of the other model. In our first-pass model, we specified these three components to the decision-making goal (and therefore to the instance chunks that those goals will become when completed): the discrepancy between current and target water levels, the action to be taken as the difference between user inflow and outflow, and the outcome of that action in terms of remaining discrepancy (i.e., deviation from target water level) after the action took place. All those quantities were readily available from the display and did not represent significant cognitive work to integrate in the current goal. The decision-making procedure was as follows: perform a memory retrieval using the discrepancy (attribute 1) as the cue determining the similarity of past instances to the current situation. The blending process alluded to above returns, given that cue, the consensus value of the other two attributes: the suggested action (attribute 2) and expected outcome (attribute 3). The model then corrects the suggested action by the expected outcome (i.e., remaining discrepancy) and puts it directly into effect (i.e., without any mediating process such as for the RPD model). If the resulting action is positive, the model specifies the user input level to that amount and leaves the user output level to 0, and vice versa if the action amount was negative. The action taken and its (almost) immediate outcome are recorded in the decision goal chunk together with the original cue (the original discrepancy), which is then cleared and enters long-term memory as a decision instance.

The performance of the first model is displayed in Figure 6a in terms of the current water level as a function of trial. The target water level is 4 and the average of 16 model runs (as many as the number of subjects in the experiment) of 100 trials is displayed. On the positive side, the model reproduces quite well the initial overshooting of the target level from the starting value of

2 to about 7, followed by a gradual decrease to the target level of 4 in about 10 trials. This initial overshooting results because the model's backup heuristic (as encoded in the initial instances) assumes no environmental inputs or outputs and instead simply attempts to bridge the gap between current and desired level. The environmental input starting at a value of 2 leads to the excess water level until the model gradually learns that its action leads to a positive outcome (i.e., excess water), which leads to a gradual correction as those instances get stronger and more numerous and come to dominate the blending process that produces recommended action and expected outcome. However, after that initial correction the water level starts to drift steadily upward to end up at about 8 units (i.e., 4 above the desired level) by the end of the 100 trials. This drift results because the model consistently underestimates the amount of environmental input. Since that amount keeps steadily increasing with each trial, the model's knowledge of the system residing in the individual control instances is bound to produce an estimate biased in the past, just like an estimate of the size of computer memory based on a sample of five-year-old computers is bound to underestimate its current value. This effect can be modulated (e.g., by increasing the rate of memory decay) but cannot be eliminated through parameter variations. It is a fundamental implication of the model representation.

INSERT FIGURE 6 HERE

The opposite phenomenon happens with decreasing linear environmental inflows (see Figure 6b): ACT-R brings the system down to the desired stock but then the water level keeps drifting down, constantly overestimating the expected environmental inflow. Note however the

correspondence with the human data in both the magnitude and duration of the initial overshoot, both about twice as large as for the increasing environmental inflow condition.

One way to learn to control a system that is constantly drifting is to represent knowledge of that system in relative rather than absolute terms. This insight is intuitively appealing, since many of our everyday experiences in controlling complex systems (e.g., driving a car) are expressed in relative terms as well (e.g., turn the steering wheel clockwise to move the car to the right). In this case, that means that while future levels of environmental inputs will constantly keep changing, one thing that is constant under the simple function of linearly increasing environmental inputs, hence presumably easily learnable, is its rate of increase. To be able to learn that information regarding the system dynamics, one needs to represent information about the system in relative rather than absolute terms. We tried a number of variations that led to similar outcomes. The version whose performance is displayed in Figure 7a still represents the current situation as the discrepancy between current and desired level, but represents the action in terms of **change** in user-controlled flow (rather than absolute difference in input and output) and represents the outcome in terms of **change** in water level (as opposed to absolute difference to target level). The decision procedure is otherwise unchanged. The model now has a more stable sense of the system dynamics and is able to keep the water level relatively steady around the desired level. Oscillations remain that reflect the stochasticity and dynamism of its knowledge base in declarative memory as well as the robustness of the control process: as the water level gradually drifts away from the desired value, the model slowly corrects it and brings it back in line. A similar pattern can be observed for the decreasing input condition in Figure 7b. However, the model is now in a sense too good because it does not display much if any of the initial overshooting, and then too briefly. It seems that the right model might be a combination of

an initial absolute representation combined with a gradual switch to a relative representation. One reason why the subjects would initially prefer an absolute representation is that it is cognitively easier as it is all directly available on the external screen. In contrast, to be able to maintain the relative representation in the second model requires additional cognitive work in terms of maintaining internally between trials the previous action and outcome in order to be able to estimate the difference. Thus it seems reasonable to speculate that subjects will not go to these lengths until they have a sense that they cannot control the system using an absolute representation and that they need to switch to a more demanding relative representation in order to be successful. Modeling that switch in representation is a future challenge of our modeling efforts. Achieving that type of effect is usually less difficult than avoiding the introduction of myriad free parameters in the metacognitive process controlling representation, where the modeler often gets out of the model exactly what has been put in. Thus the true challenge is to find a way to affect this change of representation in a way as controlled and parameter-free as the accumulation and application of knowledge.

INSERT FIGURE 7 HERE

Qualitative Model Comparisons and Conclusion

We presented two computational models, coming from distinct architectures and modeling approaches, to model human behavior in a common dynamic task, controlling the accumulation of a stock given an inflow and an outflow. The first computational model is an instantiation of macrocognitive processes, specifically recognition-primed decision-making,

while the other model is based on a traditional microcognitive framework, the ACT-R cognitive architecture.

One striking result of our model comparison is the degree to which those seemingly different modeling approaches have independently converged toward computational architectures with considerable similarities (see Table 1). Both approaches rely on a "flat" representation of individual traces as cues-action-outcome; that is, there is no internal complexity to the trace, no composite representations smuggled in as “individual” cues, no pointers to other data structures, and no implicit hierarchical relationship among the traces themselves. Both approaches relay on a similarity-based recall mechanism operating in parallel over the entire store of traces. And both approaches employ a "blending" process in which multiple traces contribute to the course of action. Although it might seem that this convergence might have been recognized a priori, without a common task to which the architectures can be applied it is far too easy to dismiss differences and overlook commonalities between modeling approaches. To paraphrase the cliché, modelers are often separated by a common language. In this case, the model comparison revealed that the independent implementation of the instance-based decision-making paradigm has overcome what might appear to be the more fundamental division between macrocognitive and microcognitive approaches.

Even more striking is that the resulting models themselves were so similar. Even though each approach implements a model of instance-based decision making, the similarity between the models was not preordained. Just as people can adopt widely different strategies to a task using the same basic cognitive mechanisms, so too can modelers develop highly distinct models, even using the same framework. For example, as we demonstrated above, the choice of relative versus absolute representation had a profound impact on model performance under both

approaches and it is a choice that would not seem to be constrained by the architectures. And yet both modeling teams followed the same development trajectory, starting with the simpler absolute representation, concluding that it could not generate the desired behavior, both in terms of functional control of the task and in terms of matching human behavior, and generalizing to the more complex but powerful relative representation. We offer this as anecdotal evidence against the common perception that a computational model reflects only the skill of the modeler. To those who might still view this convergence as a happy accident, we would argue that it is worth noting that an instance-based approach is likely not the first choice a modeler might make in representing the human performance in the DSF task. Indeed, the mechanisms that support similarity-based recall and action are relatively lightweight compared to those that might be used to sense, estimate, act, and adjust. And to the extent that thinking along these lines dictates a modeling solution, we would argue that the architecture constrains the modeler.

Despite all of these striking similarities, however, there were also real differences that emerged from our model comparison. For instance, there is a statistical learning mechanism in ACT-R that allows recency effects to impact performance; no such mechanism exists in the RPD decision type. There is also an important difference in the internal representations the architectures employ. The ACT-R model uses numerical representations for cues and courses of action that allow both inflow and adjustments to be given as continuous quantities. The RPD decision type represents cues and courses of action as discrete enumerations and thus depends on the modeler to "bin" values appropriately. Finally, ACT-R supports the implementation of any user-defined similarity function, whereas the RPD decision uses a fixed similarity function that depends on a specific cue-encoding scheme (see Appendix A below for details). Each of these differences carries significant theoretical weight.

One could argue that this model comparison has failed because it has not rendered a verdict as to which side of these differences is right and which is wrong, or because the models do not fit the data well enough. But that was never the point. Rather, our intent was to illuminate general features of computational approaches to macrocognition, rather than to validate a single theoretical viewpoint or achieve a perfect fit to a specific data set. In this respect, the comparison has been a success insofar as it provides two independent examples of instance-based approaches to decision making being used to simulate performance and thus suggest a candidate "design pattern" for computational approaches to macrocognition. The comparison also provides specific examples of computational mechanisms and brings a level of specificity to the question of what is at stake in accepting one or another putative representation of macrocognition. Finally, and perhaps most importantly, this comparison reminds us that computational cognitive modeling is a powerful tool for understanding human performance, and that differences between specific frameworks adopted to pursue that goal are often exaggerated.

Acknowledgements

This research was partially supported by the Army Research Laboratory (award DAAD19-01-2-0009). We would like to thank Varun Dutt for his work on the development and data collection with the DSF task.

References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York, NY: Oxford University Press.

Anderson, J. R., & Lebiere, C. L. (2003). The Newell test for a theory of cognition. *Behavioral & Brain Sciences, 26*, 587-637.

Booth Sweeney, L., & Sterman, J. D. (2000). Bathtub dynamics: Initial results of a systems thinking inventory. *System Dynamics Review, 16*(4), 249-286.

Cassimatis, N., Bello, P., & Langley, P. (2008). Ability, breadth and parsimony in computational models of higher-order cognition. *Cognitive Science, 32(8)* 1304-1322.

Chipman, S. F., Schraagen, J. M., & Shalin, V. (2000). *Cognitive Task Analysis.* Mahwah, NJ: Lawrence Earlbaum Associates.

Cronin, M., & Gonzalez, C. (2007). Understanding the building blocks of system dynamics. *System Dynamics Review, 23*(1), 1-17.

Cronin, M., Gonzalez, C., & Sterman, J. D. (2009). Why don't well-educated adults understand accumulation? A challenge to researchers, educators and citizens. *Organizational Behavior and Human Decision Processes, 108*, 116-130.

Dierickx, I., & Cool, K. (1989). Asset stock accumulation and sustainability of competitive advantage. *Management Science, 35*(12), 1504-1511.

Dutt, V., & Gonzalez, C. (2007). Slope of inflow impacts dynamic decision making. Paper presented at the *25th International Conference of the System Dynamics Society*.

Edwards, W. (1962). Dynamic decision theory and probabilistic information processing. *Human Factors, 4*, 59-73.

Erev, I., Ert, E., Roth, A. E., Haruvy, E., Herzog, S., Hau, R., Hertwig, R., Stewart, T., West, R., & Lebiere, C. (submitted). A choice prediction competition, for choices from experience and from description. *Journal of Behavioral Decision Making*.

Gluck, K., & Pew, R. (2005). *Modeling human behavior with integrated cognitive architectures*. Mahwah, NJ: Erlbaum.

Gonzalez, C., & Dutt, V. (2007). Learning to control a dynamic task: A system dynamics cognitive model of the slope effect. Paper presented at the *8th International Conference on Cognitive Modeling*, Ann Arbor, MI.

Gonzalez, C., & Lebiere, C. (2005). Instance-based cognitive models of decision making. In D. Zizzo & A. Courakis (Eds.), *Transfer of knowledge in economic decision making*. New York: Palgrave McMillan.

Gonzalez, C., Lerch, F. J., & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science 27*(4), 591-635.

Hintzman, D. L. (1984). MINERVA 2: A simulation model of human memory. *Behavior Research Methods, Instruments & Computers, 16*, 96-101.

Hintzman, D. L. (1986a). *Judgments of frequency and recognition memory in a multiple-trace memory model*. Eugene, OR: Institute of Cognitive and Decision Sciences.

Hintzman, D. L. (1986b). "Schema abstraction" in a multiple-trace memory model. *Psychological Review, 93*(4), 411-428.

Jilk, D. J., Lebiere, C., O'Reilly, R. C., & Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental & Theoretical Artificial Intelligence, 20*(3), 197-218.

Klein, G. (1989). Recognition-primed decisions. In W. B. Rouse (Ed.), *Advances in man-machine systems research* (pp. 47-92). Greenwich, CT: JAI Press.

Klein, G. (1993). *Naturalistic decision making: Implications for design*. Wright-Patterson Air Force Base, OH: Crew System Ergonomics Information Analysis Center.

Klein, G. (1998). *Sources of power: How people make decisions*. Cambridge, MA: The MIT Press.

Lebiere, C. (1999). Blending. In *Proceedings of the Sixth Annual ACT-R Workshop*. George-Mason University, Fairfax, Virginia.

Lebiere, C., Archer, R., Best, B., & Schunk, D. (2008). Modeling pilot performance with an integrated task network and cognitive architecture approach. In D. Foyle & B. Hooey (Eds.), *Human performance modeling in aviation*. Mahwah, NJ: Erlbaum.

Lebiere, C., & Bothell, D. (2004). Competitive modeling symposium: Pokerbot World Series. In *Proceedings of the 2004 International Conference on Cognitive Modeling*. Mahwah, NJ: Erlbaum.

Newell, A. (1990). Unified theories of cognition. Cambridge, MA: Harvard University Press.

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

Sterman, J. D. (1989). Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. *Management Science, 35(3),* 321-339.

Sterman, J. D. (2000). *Business dynamics: Systems thinking and modeling for a complex world*. Cambridge, MA: McGraw Hill.

Sterman, J. D., & Booth Sweeney, L. (2002). Cloudy skies: Assessing public understanding of global warming. *Systems Dynamics Review, 18*(2), 207-240.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data and Knowledge Engineering (DKE), 25(1-2),* 161-197.

Taatgen, N., Lebiere, C., & Anderson, J. R. (2006). Modeling paradigms in ACT-R. In R. Sun (Ed.), *Cognition and multi-agent interaction: From cognitive modeling to social simulation*. New York: Cambridge University Press.

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

Warwick, W., & Hutchins, S. (2004). Initial comparisons between a "naturalistic" model of decision making and human performance data. In *Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation*.

Warwick, W., McIlwaine, S., Hutton, R. J. B., & McDermott, P. (2001). Developing computational models of recognition-primed decision making. In *Proceedings of the 10th Conference on Computer Generated Forces*.

Appendix A: RPD Process Equations

Here we describe the encoding scheme for cue enumerations, the calculation of similarity values and the calculation of recognition strengths across courses of action.

First, suppose a cue is represented using *n* values. Internally, the first enumerated value would be encoded with n-1 "1"s, the last value with n "-1"s and the m^{th} of n values with n-m "-1"s and the remaining n-1 leading bits of "1"s.

Now, let *p* be a vector of cues that define a situation and let *t* be a *single* remembered decision making episode, where p_i and t_i are the i^{th} bits of the *probe* and episode respectively. Suppose the cue vector requires *k* bits to encode, then the similarity value, $s_{p,t}$, between *p* and *t* is given by:

$$s_{p,t} = \frac{1}{k}\sum_{i=0}^{k-1} p_i t_i$$

The resulting similarity value will be between –1 and 1. The closer the similarity value is to 1, the greater the similarity is between the current situation and episode; a similarity value of 1 indicates a perfect, cue-by-cue match between probe and trace. A similarity value of -1 indicates a perfect cue-by-cue mismatch. As we describe below, a similarity value of -1 would result in a row subtracting strength for recognition. To some, this might seem like an intuitively appropriate analogue for computing the impact of completely dissimilar situation on recognition. To us, however, the symmetry of the calculation was less intuitive. So, rather than have negative similarity values subtracting from recognition, we simply round those value to zero, where they make no contribution, positively or negatively, on recognition.

Courses of action are encoded using a different scheme in the bit-string. We use what is, essentially, a monadic notation for encoding courses of action. For example, the first of five enumerated course of action would be encoded as "00001," the third as "00100" etc. This notation is useful because distribution of recognition strengths is computed COA-by-COA where the similarity value of each episode is multiplied with the associated COA bit. More formally, let c_i be the recognition strength of the i^{th} course of action, then its value is given by:

$$c_i = \sum_{j=1}^{n} t_{ji} a_j r_j$$

where t_{ji} is the bit in the jth remembered episode corresponding to the ith course of action, a_j is the activation value for that episode and r_j is the success value of the situation-COA pair (the value of n will grow as the model accumulates experience).

The activation value a_j is just the similarity value of the jth episode raised to some integer power. By “accelerating” the similarity value in this way the contribution a given episode makes becomes a non-linear function of its similarity value and, thus, a high similarity will result in that episode making a disproportionately greater contribution of its associated course of action to the distribution of recognition strengths. In this way we can represent differences in the specificity of the recognition process.

The result, r_j, is a records the outcome of the associated course of action as either positive (encoded as "1") or negative (encoded as "-1"). In this way, successful outcomes increase recognition strength for the associated course of action, while unsuccessful outcomes decrease the recognition strength for the associated course of action.

Appendix B: ACT-R Process Equations

Access to a chunk is determined by its activation, which is a quantity defined by the following equation:

$$A_i = \sum_{j=1}^{n} t_j^{-d} - \sum_{cue} \left(1 - Sim\left(cue, value\right)\right)$$

The first term determines the base-level activation of a chunk as a function of past references and captures both the power law of learning and the power law of forgetting, while the second term reduces the activation of a chunk according to the degree to which it (mis)matches the required pattern, i.e. the current situation. Usually, the process results in the retrieval of the single chunk with the highest activation after noise is added, which makes it a stochastic process with probabilities of retrieval described by the Boltzmann (softmax) equation where t is a temperature parameter that is a function of the noise level:

$$P_i = e^{A_i/t} \Big/ \sum_j e^{A_j/t}$$

However, for domains in which a continuous estimate needs to be generated, the process is generalized to return a consensus value of the entire set of chunks according to the blending equation (Lebiere, 1999; Gonzalez et al., 2003):

$$V = \min \sum_i P_i \cdot \left(1 - Sim\left(V, V_i\right)\right)^2$$

This states that the value V retrieved is the value that minimizes the dissimilarity with actual values V_i in each chunk i, weighted by the probability Pi of retrieving that chunk as defined above. If the values V_i are numerical and the similarity function Sim(V, V_i) is linear, then the process is equivalent to a probability-weighted averaging. In general, this provides a similarity-based blending process similar to that observed in neural networks. The three

equations above, combined with the automatic learning of new instance chunks from previous problem solving episodes, determine directly the outcome of the instance-based process given the prior history. All architectural parameters such as rate of memory decay d or noise level t were left at the default value that we used in our prior instance-based models, and the similarity function Sim was likewise set at the usual linear scaling function traditionally used between quantitative values such as in this case levels of water.

Footnotes

[1] Micro Saint Sharp normally supports three different models of decision making: "tactical" decisions, in which the branching at the task network level is determined by user-defined Boolean conditions, probabilistic decisions, where the branching is determined by user defined probability distributions, and "multiple" decisions, which allow parallel execution of downstream tasks.

[2] Cf. Klein (1998).

[3] Klein (1998) actually describes three variants of the recognition-primed decision: one version is the simple match, discussed here while the two other versions describe the role of diagnosis and "story building" in decision making.

[4] Compare this to what goes on in any categorization task. Even when a decision space is complicated, typically the hyperplanes that partition the space are fixed.

Table 1

Comparison of components of ACT-R and RPD models

ACT-R Model	RPD Model
Cues-action-outcome chunk	Cues-action-outcome trace
Chunk activation: frequency/recency	Trace activation determined by similarity
Matching reflects cue similarities	Recognition reflects cue similarities using transformed representation
Blending retrieval reflects all chunks	Recall draws on every past trace
Continuous action with opt. correction	Discrete course of actions
Outcome recorded in decision chunk	Outcome with respect to “success criterion” recorded in trace

Figure Captions

Figure 1. The Dynamic Stocks and Flow (DSF) simulation. The center of the screen shows a water tank carrying 1 gallon of water. The markings on the left side of the tank represent the water level in the tank at any instance of time. The *Goal (Gallons)* at 4.00 gallons refers to the level to maintain in each time period of the simulation run. The *Amount in Tank (Gallons)* at 2.00 gallons refers to water level that the tank will have at the end of the current time period. A participant in this experiment enters the Inflow value in *Enter the number of Inflow (units/second)* and the outflow in *Enter the number of Outflow (units/second)* and press the *Submit* button.

Figure 2. Human data performing in (a) increasing and (b) decreasing linear Inflow function in DSF – Data originally reported by Gonzalez and Dutt (2007).

Figure 3. Model performance on the DSF task using absolute representation within the RPD decision type. Performance is given as an average of current stock per trial over 16 runs for each environmental inflow condition: (a) increasing and (b) decreasing. The target stock is 4 in each condition.

Figure 4. Model performance on the DSF task using relative representation within the RPD decision type. Performance is given as an average of current stock per trial over 16 runs for each environmental inflow condition: (a) increasing and (b) decreasing. The target stock is 4 in each condition.

Figure 5. ACT-R architectural diagram including main architectural modules and buffers, and communication processes.

Figure 6. Model performance of an average of 16 runs with absolute representation for (a) increasing and (b) decreasing environmental inflow. Performance is displayed as current stock level for each of 100 trials with desired stock level being 4.

Figure 7. Model performance of an average of 16 runs with relative representation for (a) increasing and (b) decreasing environmental inflow. Performance is displayed as current stock level for each of 100 trials with desired stock level being 4.