

Exploration for Understanding in Cognitive Modeling

Kevin A. Gluck

Clayton T. Stanley

*Air Force Research Laboratory
711 HPW/RHAC
2698 G Street, Bldg 190
WPAFB, OH 45433, USA*

KEVIN.GLUCK@US.AF.MIL

CLAYTON.STANLEY.1@US.AF.MIL

L. Richard Moore, Jr.

*Lockheed Martin at
Air Force Research Laboratory
6030 S. Kent St
Mesa, AZ 85212, USA*

LARRY.MOORE@MESA.AFMC.AF.MIL

David Reitter

*Department of Psychology
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213, USA*

REITTER@CMU.EDU

Marc Halbrügge

HALBRUEGGE@GMAIL.COM

Editor: Christian Lebiere, Cleotilde Gonzalez, and Walter Warwick

Abstract

The cognitive modeling and artificial general intelligence research communities may reap greater scientific return on research investments – may achieve an improved understanding of architectures and models – if there is more emphasis on systematic sensitivity and necessity analyses during model development, evaluation, and comparison. We demonstrate this methodological prescription with two of the models submitted for the Dynamic Stocks and Flows (DSF) Model Comparison Challenge, exploring the complex interactions among architectural mechanisms, knowledge-level strategy variants, and task conditions. To cope with the computational demands of these analyses we use a predictive analytics approach similar to regression trees, combined with parallelization on high performance computing clusters, to enable large scale, simultaneous search and exploration.

Keywords: model comparison, exploration, sensitivity, necessity, efficiency, predictive analytics, high performance computing

1. Introduction

Computational process modeling is necessary in the study of human cognition and other intelligent systems because the complex interactions among their adaptive, generative, and stochastic processes render verbal reasoning about the systems ineffective (McClelland, 2009). In the broad context of the cognitive science community's requirement for rigorous models and architectures, some have advocated the adoption of formal comparative methodologies. For instance, Kintsch, Healy, Hegarty, Pennington, and Salthouse (1999) encouraged, "direct experimental face-offs among models," noting that, "such comparisons would be very informative, and much more could be and should be done in this respect than has heretofore been attempted" (p. 436). Pew, Gluck, and Deutsch (2005) offered a similarly positive assessment, stating that model comparisons are, "a productive way to push the frontiers in terms of model architecture development as well as our understanding of stable, productive methodologies for moving from architecture to detailed, robust human performance models" (p. 405). Even more recently, Langley, Laird, and Rogers (2009) advised the research community to, "devote more serious attention to methods for the thoughtful evaluation of cognitive architectures" and prescribed, "systematic experiments that are designed to identify sources of power" (p. 156). They did not, however, offer specific guidance regarding effective experimental designs for pursuing this objective. What sort of systematic experiments with cognitive models and architectures might reveal their sources of power, their key explanatory mechanisms, the critical processes, representations, and assumptions that enable desired functionalities?

Estes (2002) had a suggestion about this. The suggestion is systematic experiments that allow investigation not only of the *sufficiency* of models and their mechanisms, but also the *necessity* of the mechanisms in those models. Estes' treatment of this topic was in the context of psychological models that predict human experimental data. He characterized the goal as the identification of *appropriate* models. Appropriate is defined as a model that "... is necessary and sufficient for prediction of the data" (p. 5). If a model's predictions are correct, one can conclude that the model is sufficient for predicting the data. One cannot conclude, however, that the assumptions of the model are necessary. Necessity can only be demonstrated by modifying the model's assumptions and showing that the modified version no longer predicts the data. If this can be shown, then the assumptions and processes built into the model are, in fact, necessary and sufficient for predicting the data, and therefore they comprise an appropriate model. Estes wrote:

"... an improved strategy is, at each step, to compare a reference model with an alternative version of the same model that differs from it with respect only to inclusion or exclusion of a single parameter or process. One gains information, not only about the sufficiency of the reference model for predicting the test data, but also about the contribution of the component whose exclusion leads to (relative) disconfirmation of the model." (p. 8)

These necessity analyses involve varying only one model component at a time, thereby revealing the extent to which that particular component is a source of power in the model.

Sensitivity analysis is a version of this model evaluation methodology. Instead of varying the presence of a model component or characteristic, a sensitivity analysis involves systematic changes to the value of a parameter, in order to evaluate the extent to which model outcomes are influenced or determined by that parameter. The greater the influence, the more powerful is the parameter in the model.

Sensitivity and necessity analyses are methods for systematic exploration of the performance profiles and fit surfaces of model variants. They are relatively efficient and easy to do with

mathematical models. Unfortunately, exploring the performance spaces of computational systems is neither efficient nor easy. It is resource intensive. Even with the dramatic gains in processor speed and multi-core computing that have been realized in the last 20 years, we do not have available on our desktops and laptops today the processing resources required to quickly complete large-scale explorations.

Beyond the challenge of acquiring access to larger scale processing resources, we also have the challenge of using whatever processing resources we have as efficiently as possible. Exploration at the intersections of architectural mechanisms and knowledge-level model variants is combinatorially explosive and will overwhelm even the largest processing systems if not handled intelligently. We need exploration methodologies and resources that help us explore not only more completely, in the interest of improving understanding, but also more efficiently, in the interest of maximizing the information value of whatever computing time we have available.

Our objective in this paper is to encourage both the cognitive modeling and artificial general intelligence research communities to consider the possibility that we will reap greater scientific return on our research investments – that we will achieve a broader and deeper understanding of our architectures and models – if we expand our emphasis in model development, evaluation, and comparison to include sensitivity and necessity analyses. We provide use cases for a convergence of methodologies that are atypical in computational cognitive process modeling, demonstrated in the context of the Dynamic Stocks and Flows (DSF) model comparison challenge. Our purpose is not to conduct and describe an extensive model comparison in its entirety. Instead, we are making a focused methodological point regarding the benefits of systematic model explorations, both in the context of rigorous comparisons and in the broader contexts of cognitive modeling and intelligent systems more generally. The next section describes the efficient exploration algorithms and large scale processing resources that provide the technological backbone for the subsequent analyses.

2. Efficient Exploration and Large Scale Processing

A common distributed computing scenario involves a parameter search space divided into an optimized grid. The nodes within the grid are then distributed amongst the nodes in a computational cluster and the model or simulation is executed in parallel to produce a response surface. This is an adequate description of our first foray into high performance computational resources (Gluck, Scheutz, Gunzelmann, Harris, & Kershner, 2007).

However, the demands of our noisier and more complex models necessitated increased optimization in our searching strategies. Optimized searching strategies are commonly used within workstations to solve many mathematical problems, and so are well understood. For example, there are many variations of gradient descent, where one follows the local gradient on the response surface to locate optima. Genetic algorithms are also increasingly popular, whereby proven attributes are combined with random mutations to home in on optimum parameters (Kase, Ritter, and Schoelles, 2007).

Although these techniques do generally enjoy some degree of success, our experiences exposed a shortcoming in that the qualitative nature of the remainder of the parameter space remained hidden. Unanswered questions remain about the nature of the solution: Is there a single optimum, or multiple? What shape does it take? What is the nature of the relationships among parameters? How influential are the parameters? How does optimal parameter performance compare with canonical or default parameter performance?

2.1 Predictive Analytics

In an effort to improve on our cognitive model exploration and optimization methodologies, we have turned to the field of predictive analytics. Predictive analytic methodologies are typically focused on post hoc analyses rather than parameter space search, but the ability to make mathematical predictions about a space is a necessary precursor in deciding where to explore, so their relationship is not quite as disparate as might initially appear. Remarkably, the idea of applying techniques from predictive analytics towards computational searching seems largely unexplored.

Adapting predictive analytic approaches to computational searching has several advantages. First, many methodologies are nonparametric—they make no assumptions about the data themselves. This is ideal for parameter spaces, where the shape and linearity of the response surface is driven by theory rather than preference. Second, predictive analytics make no specific requirements on where a space is sampled. Rather than being fully dependent upon the accurate calculation of specific grid points, approaches derived from predictive analytics can, and perhaps should, sample stochastically over a broad range of the space. Broader, thinner sampling also increases robustness in finding localized, high frequency phenomena.

Predictive analytics sports a number of approaches that might be adapted for computational searches and exploration (for a brief overview, see Friedman, 1991). Each approach presents a trade space between predictive power and computational efficiency, which we have yet to fully explore. For our initial implementation, however, we have chosen a technique similar to recursive partitioning regression, or regression trees (Alexander and Grimshaw, 1996). With this approach, the overall parameter space is recursively divided into an n-ary tree, with each of the leaves spanning a specific portion of the search space. Each leaf can then be regressed independently to provide a local gradient for searching and, when all the leaves are considered together, a qualitative feel for the full space. More details will be provided below, but let it suffice for the moment to say that the approach was chosen because it provides good predictive power with relatively small computational overhead.

2.2 Cell and Tornado

Our current implementation of regression trees adapted for searching consists of two software programs named Tornado and Cell. Tornado provides real time visualization and some basic analytical tools. Some of the illustrations in this paper came from the Tornado software.

Cell, on the other hand, executes the model or simulation as needed and performs the actual searching process. Both Cell and Tornado share much of the same code base, but they vary in their user interface and function. Typically, many Cell instances run in parallel on large-scale computational resources to tackle the parameter space while an instance of Tornado might occasionally check on progress or analyze the final results.

For the remainder of this discussion we will focus on Cell, as it provides the optimized search methodology of interest. A simple configuration file is specified when Cell starts, which describes the parameters and ranges in the search space, the measures of interest from the model, and optional target measures to search for. These targets define “areas of interest” that guide the search process. They are treated as implicit dependent measures whose values are the calculated RMSE between the model and the target. If no targets are explicitly specified, the algorithm will treat areas with higher unresolved residual variation as “areas of interest.”

Cell’s behavior is guided by two governing principles:

- 1) Skew the sampling distribution towards areas of interest.

2) Divide the space into hypercubes that maintain a consistent density.

In other words, relevant areas of space will be sampled with greater density, and areas of greater density will be more finely subdivided.

Without any structural information about the space other than the geometry, Cell begins by creating a uniform random distribution of points to sample across the full space. The model is then executed accordingly to retrieve the measures of interest at each requested point, and calculate the deviation from any specified targets. This continues until enough samples are returned to produce a reasonably good regression prediction (Knofczinsky and Mundfrom, 2008) in all the child hypercubes that would result after a potential split.

Once a critical mass of samples have accumulated, a split occurs, and the parent is divided into $2^{\text{dimension}}$ child hypercubes based around a point in the center of the parent. From a computational perspective, this forms the basis of an n-ary tree. Eventually, if there is sufficient parameter sampling within the child hypercubes, they too will split into children, with each spanning an even smaller fraction of the overall space.

After the first hypercube is split, however, the sampling distribution skews towards “areas of interest.” This is achieved through a process that begins by locally regressing each of the measures, including the implicit ones, against the parameter values within each hypercube region. Typically we use a simple linear regression, as more complex variations fail to offer significant gains. Local regressions are not ideal from an analytical perspective as they produce discontinuities across hypercube boundaries, however, they are redeemed by the computational efficiency afforded in regressing smaller sample sizes.

The regression coefficients in each hypercube enable predictions of measures in that area. Of specific interest are the implicit measure predictions, which indicate deviation between model performance and specified targets. Because the regressions are angled hyper-planes, we can quickly determine the lowest corner through the gradient. The lowest corner represents the optimum performance of the model in a particular hypercube. The hypercubes can be scanned very quickly to determine which encapsulate optimal parameter values overall at any given moment. 90% of the sampling distribution is allotted towards the best fitting cubes, with the remaining 10% scattered randomly among the remaining cubes.

If optimization targets were not specified in the configuration file, the regressions also calculate residual variation, which can be used to skew the sampling distribution towards these areas of ambiguity. In this case, the proportion of samples allotted to each cube is determined by the ratio of its own variation to the sum of residual variation in all the hyper-cubes.

Each Cell instance operates independently, analyzing the space, dividing hypercubes, creating sampling distributions, and running the model. However, as the model produces new data, the information is broadcast on the network to any interested listeners. In addition to broadcasting its own data, Cell actively listens for data from other instances, so all Cell instances share in the information generated by others. Note that no structural information needs to be shared, it is merely measured samples from the model runs that are shared. The structure is driven through statistical processes, and any interruption or loss of data among Cells is recoverable over time as each will ultimately arrive at the same conclusions. Data are also logged to a network file system, so new Cell instances can quickly “catch up” to existing instances by scanning the files. Tornado, too, can scan the log files and listen in on activity to provide real time visualization and analysis. This concept of data “raining” on the network forms the crux of the engineering design, and so explains the meteorologically inspired naming.

The result as visualized in Tornado is a surface shaped by flat planes that provide a qualitative feel for the parameter space. Tornado provides a projection of the n-dimensional space

onto 3D via user selectable variables as well as a series of sliders that can be used to manipulate the angle of projection and explore additional dimensions of the space in real time. The planes tend to be roughly cut outside areas of interest, with much more detail and finer structure near targeted areas. Tornado provides options to produce quantitative results in the form of traditional grids, which is often convenient for subsequent analytical tools, or global optima.

Currently, the Tornado visualization software runs exclusively on Mac OS X. Cell runs on the Mac and Linux operating systems, and can be used on whatever computing resources are available, from a single laptop or desktop machine, to a small research cluster, to large scale computing resources. Other recent publications provide further information on Cell's relationship with different exploration and search methods (Moore, 2010), as well as Cell's use with volunteer computing resources (Moore, Kopala, Mielke, Krusmark, and Gluck, 2010). In this paper, however, we now turn our attention to the use of this technology to support exploration of the Reitter and Halbrügge models from the DSF Challenge.

3. Case Studies: The Reitter and Halbrügge DSF Models

Marc Halbrügge and David Reitter, who had two of the top three models in the DSF Challenge, volunteered to allow their models to be used in additional analyses that would leverage the power of high performance computing resources to try to gain a deeper understanding of their models' behavior in a relatively short timeframe. Specifically, high performance computing (HPC) resources were used to explore their models' sensitivity to changes in architectural parameter values and to evaluate the necessity of knowledge-level strategy implementations in their models. These analyses require an exploration of the space of architectural parameter values, allowing those values to venture away from their canonical defaults. The Cell software was used to guide this exploration, improving the efficiency of the process and maximizing the information value of each processing hour.

3.1 Sensitivity analysis: architectural parameters

The sensitivity analysis model runs were performed on the Mana Cluster at the U.S. Air Force Research Laboratory's Maui High Performance Computing Center and also the Michael J. Muuss Cluster at the U.S. Army Research Laboratory's Aberdeen Proving Ground. Access to these HPC clusters allowed us to complete approximately 40 million model runs in the three months between the release of the call for papers for this special issue and the submission deadline. That is 123 thousand processor hours in a combined total of 240 hours of wall clock time on the clusters, for an average usage rate of 512 processors per cluster hour.

Model explorations were run across all nine conditions from the DSF model comparison challenge – the four 'development' conditions (Dutt and Gonzalez, 2007; Gonzalez and Dutt, 2007) used in the dataset provided to modelers for purposes of model development, as well as the five 'transfer' conditions, the data from which were not provided to the modelers before submission for the competition.

The fitness functions for Cell's explorations of these spaces were the root mean squared errors (RMSEs) of model performance to human performance for each of the conditions (lower fitness values are better here). Performance for a given trial was calculated by taking the absolute deviation between the current level in the tank and goal level for every subject/model run and then collapsing across subjects/model runs by taking the median (models were run 15 times and then collapsed). The median was used as a measure of central tendency to handle large outliers

present in the data. Each model run calculated 100 trials analogous to the subject session, so the RMSE for each condition was calculated by comparing the collapsed 100 trials generated by the model to the collapsed 100 trials observed from participants.

3.1.1 REITTER'S MODEL

Reitter's (under review, this volume) model was implemented using the ACT-R cognitive architecture (Anderson, 2007). It uses instance-based learning and declarative memory (DM) retrieval mechanisms to find the strategy (from a base set of strategies) that works best for a given task. The current best strategy is the one that is retrieved from a declarative memory request, and the likelihood that a strategy will be retrieved is directly related to that strategy's activation (higher activation leads to higher likelihood). To calculate activation, Reitter's model supplements the traditional ACT-R base-level learning mechanisms with a blending component (Wallach and Lebiere, 2003) that allows rewards for past episodes of experience with choosing a strategy to weigh in on that strategy's activation in DM. Using these components, the decay rate (:bll) and instantaneous noise (:ans) for each strategy can influence the activation of each memorized episode. Activation of the episodes along with a 'blend temperature' term determine the probability that the strategy will be retrieved, so each of these three parameters can influence which strategy is ultimately chosen for the task. To explore the sensitivity of Reitter's model performance to relevant architectural parameters, we manipulated the base-level learning decay rate (.01-.99), activation noise (.01-.99), and blend temperature (.01-10) for both the development and transfer datasets. Analyses showed that blend temperature had an insignificant effect on model performance, so results across this dimension were collapsed, leaving a standard two-dimensional surface plot.

3.1.2 HALBRÜGGE'S MODEL

Halbrügge's (under review, this volume) model was also implemented using the ACT-R theory of cognition (Anderson, 2007), leveraging the architecture's production utility and learning mechanisms to find the strategy (from a base set of strategies) that works best for each task condition. The current best strategy is the one with the highest utility, and utilities change over time depending on how well each strategy has been working. Formally, the Difference Learning Equation (e.g., Bush and Mosteller, 1955; Rescorla and Wagner, 1972) determines these utility values, and a key parameter in this equation is the learning rate (:alpha). The ACT-R theory also adds noise to utility values (:egs), and more noise generally leads to more explorative behavior, so the variance of the noise distribution can influence which strategy is chosen as well. To explore the sensitivity of Halbrügge's model performance to relevant architectural parameters, we manipulated the learning rate (.01-.99) and utility noise (.01-.99), across all development and transfer conditions.

3.1.3 ARCHITECTURAL PARAMETER SENSITIVITY ANALYSIS RESULTS

Results from both Reitter's and Halbrügge's models in all nine conditions are shown in Table 1, resulting in 18 RMSE surface plots. For each of these 18 explorations we provide the best fitting architectural parameter values.

A general comment on the set of results in Table 1 is that model RMSE with the human data is determined in a complicated space of interactions at the multi-dimensional intersection of chosen knowledge-level (strategy) implementation, architectural parameter values, and task

condition. In some cases, there is evidence that the quality of the models' replication of the human data is indeed sensitive to both relevant architectural parameters. An example of this is available in the results from Reitter's model for the sequence-4 condition in the transfer dataset, which is enlarged for inspectability below in Figure 1. Note that performance is dependent both on base-level learning decay rate (:bll) and activation noise (:ans).

In other cases, there is evidence that a model's RMSE with the human data is mostly or entirely insensitive to the particular architectural parameter values chosen. An example of this is available in Figure 2, which is the fitness surface for Halbrügge's model across all the conditions in the development dataset. Note that performance does show some sensitivity to the learning rate (:alpha), yet is insensitive to utility noise (:egs). This insensitivity does not hold across all DSF challenge conditions, however, and certainly is not true of utility noise in general.

Having completed Cell explorations with both models in each of the DSF challenge conditions, we now have 18 estimates for best fitting parameter combinations. This enables us to answer some questions that we otherwise would not be in a position to address. For instance, how variable are the best fitting parameter estimates across DSF conditions? How do the best fitting values compare to architectural defaults and/or the values chosen by the modelers for their model implementations? When using best fitting parameter values in each condition, how well do the models replicate human performance in each condition? What do Cell's results say about the assumption of the DSF challenge that models over-optimized for the data in the development conditions may not generalize well to the transfer conditions?

First, we address the variability of the best fitting architectural parameters, which is substantial. In Reitter's model, Cell found :ans values ranging from .07 to .93 and :bll values ranging from .09 to .93. In Halbrügge's model, Cell found :egs and :alpha values ranging from .07 to .68. This high variability and the association of parameter combinations with DSF task conditions risk obscuring the fact that these parameters are intended to represent cognitive traits within the human participants in the DSF study. In this particular study these conditions served as a between subjects manipulation, meaning that each of the DSF task conditions involved a different sample of human subjects. Some variation in architectural parameters is to be expected across samples of different people. However, the high level of variability in Table 1 likely reflects confounding interactions with knowledge-level (strategy) details, other architectural components, and task characteristics.

By default, both activation noise and base level learning are disabled in the ACT-R architecture. This is a modeling convenience, rather than a theoretical claim. The position in the ACT-R theory is clearly that there is transient noise in the human memory system and that memories decay. For purposes of the challenge, Reitter used .25 as the activation noise (:ans) parameter and .5 as the base level learning decay rate parameter. These are quite reasonable, commonly used, perhaps even canonical parameter values. The optima predictions in Table 1 suggest that these values for :ans and :bll are too low and too high, respectively, in the development conditions, whereas the same values end up being too high and too low (note the switch), respectively, in the transfer conditions (after removing the degenerate Delay 3 condition from the descriptive analysis). If we take into consideration the parameter surfaces, however, (the canonical values are marked with a stippled vertical line), it becomes evident that the canonical values, although quantitatively disparate from the optima, still linger in the better performing (lower RMSE) areas of the fit surface. In the interest of parsimony, the model might do reasonably well with the typical values, but understanding the trade space only becomes evident when we take the entire parameter space into consideration.

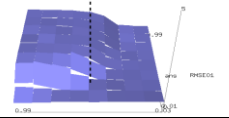
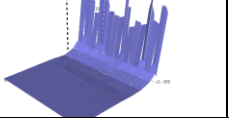
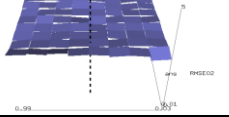
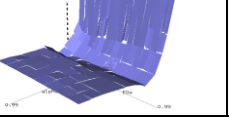
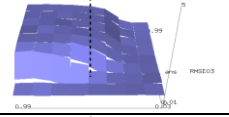
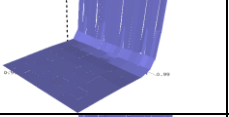
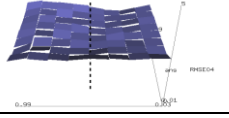
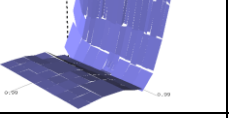
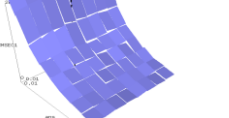
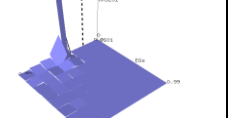
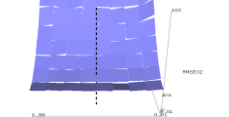
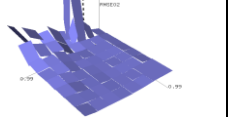
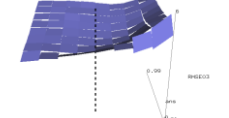
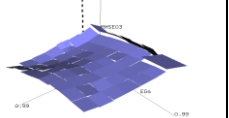
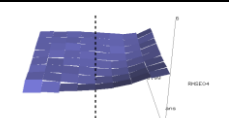
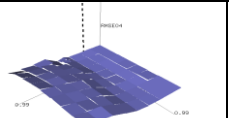
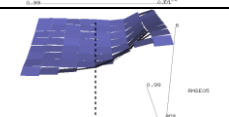
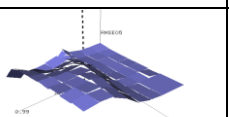
		Reitter's Model		Halbrügge's Model			
Condition		ans	bll	egs	alpha		
Development Conditions	Linear increasing		0.32	0.09		0.19	0.32
	Linear decreasing		0.93	0.33		0.61	0.24
	Nonlinear increasing		0.81	0.09		0.071	0.19
	Nonlinear decreasing		0.93	0.33		0.52	0.24
Development Mean		.75	.21	.35	.25		
Transfer Conditions	Delay 3		0.93	0.81		0.68	0.19
	Delay 2		0.07	0.56		0.19	0.07
	Sequence 2		0.07	0.93		0.07	0.68
	Sequence 2 w/Noise		0.07	0.68		0.07	0.07
	Sequence 4		0.07	0.81		0.19	0.07
Transfer Mean		.24	.76	.24	.22		
Grand Mean		.47	.51	.29	.23		

Table 1. Surface plots of RMSE between the model and aggregate human data for each condition, along with optimal parameter values, as calculated by Cell. RMSE is shown vertically and parameters range from .01 to .99 on all plots except Reitter's development plots where the base level learning (:bll) range starts at .03 to trim noise. Also on Reitter's model runs, :bll is shown horizontally while activation noise (:ans) provides depth, except for the Delay 3 condition where :ans is projecting inward right. On all Halbrügge's plots, :alpha (learning rate) projects inward left, while expected gain noise (:egs) projects inward right. The stippled lines show canonical ACT-R parameter locations.

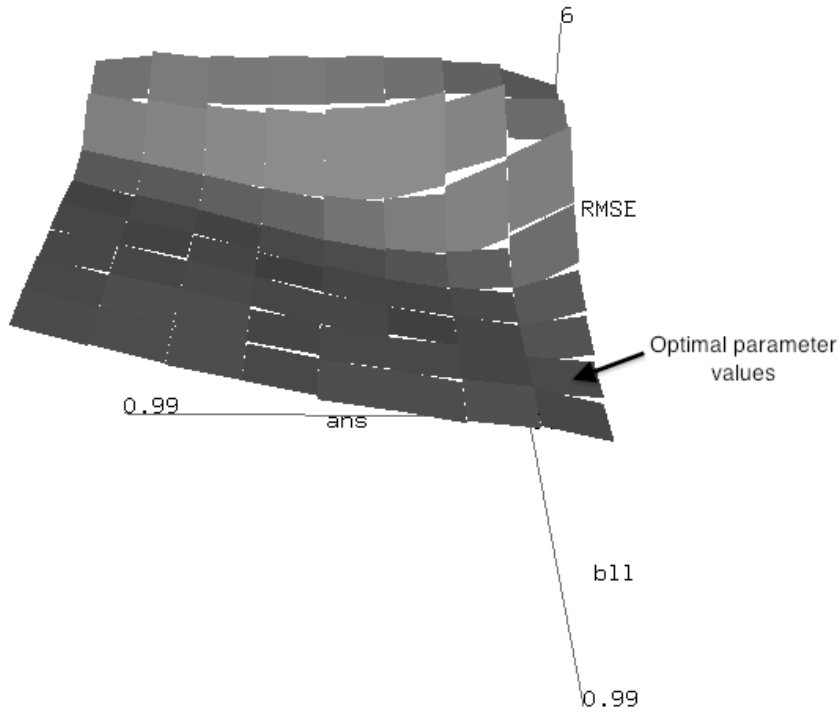


Figure 1: Reitter's model: surface plot of RMSE fitness function for the sequence-4 condition in the transfer dataset (regions of lower elevation correspond to better model performance)

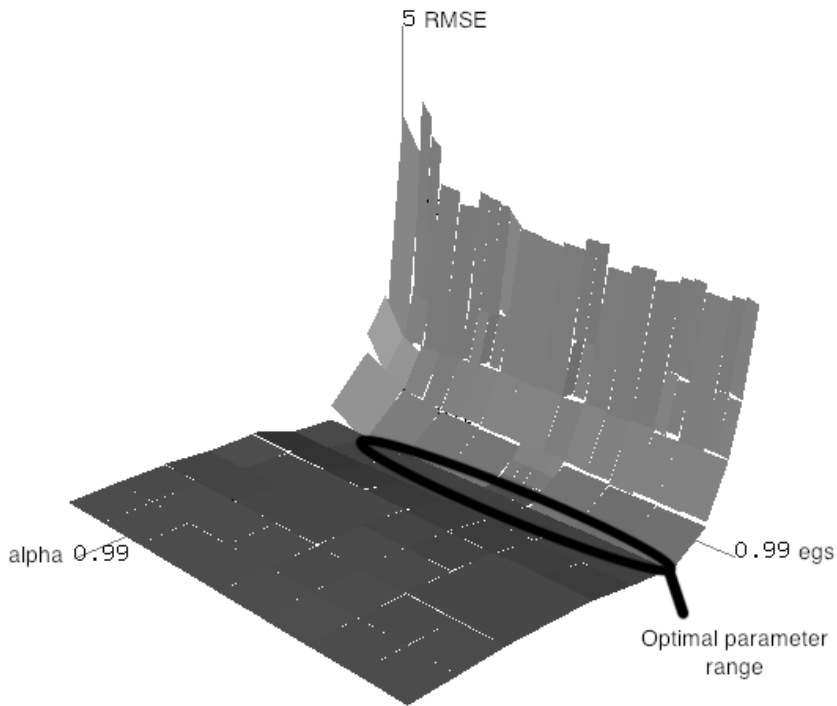


Figure 2: Halbrügge's model: surface plot of collapsed RMSE fitness function for the development dataset (regions of lower elevation correspond to better model fit to the human data)

On the procedural parameter side, Halbrügge set :egs to .2 (it defaults to *nil*) and used the architectural default value of .2 for :alpha. On average, and despite some deviations in specific conditions, these procedural values end up working out quite well in the DSF conditions. In fact, if we take the response surfaces into consideration, we can see that Halbrügge’s model is relatively insensitive to parameter values in general.

Having fully explored these parameter sensitivity spaces, we might wonder how well the two models actually replicate human performance in each of the conditions. Table 2 displays fit metrics for both models in all nine conditions, using the optimal parameter combinations found by Cell.

		Reitter’s Model		Halbrügge’s Model	
		RMSE	r	RMSE	r
Development	Linear increasing	0.58	0.95	0.13	0.96
	Linear decreasing	2.19	0.89	2.02	0.88
	Nonlinear increasing	0.78	0.60	0.49	0.71
	Nonlinear decreasing	1.21	0.90	1.33	0.92
Transfer	Delay 3	345,199	0.16	14.43	0.66
	Delay 2	28.25	0.39	1.99	0.84
	Sequence 2	3.71	- 0.02	2.65	0.58
	Sequence 2 w/Noise	2.36	0.05	1.88	0.60
	Sequence 4	3.59	- 0.01	2.32	0.67

Table 2. Root Mean Squared Error (RMSE) and correlations (r) between model and human data at the optimal parameter values reported in Table 1.

Finally, what are the implications of these results regarding issues of over-fitting and generalization? It is clear in Table 1 that Reitter’s model is at greatest risk of being misled by an overfit to the development conditions. Carrying the average best fit parameter values from the development conditions to the transfer conditions would result in a model with higher noise and lower decay than Cell claims is desirable in the transfer conditions. The magnitude of the potential damage appears to be much lower in Halbrügge’s model, but even here it appears likely that drawing directly from the development conditions to set architectural parameters would result in overestimates of the noise and learning rate parameters. Interestingly, in all cases, the models are better off in the transfer conditions using the modeler-selected, typical/canonical parameter values than they are using parameter values set on the basis of best fits to the development conditions.

3.2 Necessity analysis: model strategies

The previous section explored the Halbrügge and Reitter models with sensitivity analyses across key architectural parameters. This section explores those same models with necessity analyses. It turns out both of the models choose between two different types of strategies when doing the DSF task, so we wondered if having both strategies present was actually necessary.

When evaluating the necessity of a single model component, it’s not enough to merely remove that component from the model and then rerun, using the same parameter values for the other components as before. Due to interactions among components, best-fitting parameter values

for a model with all components enabled may be quite different than the best-fitting parameter values for a model with one of its components removed. When testing for necessity by removing components, we want to give the model the opportunity to perform as well as it possibly can, given the environment in which it is placed. Then, if the model still cannot perform as well with a certain component removed as it can with all components present, then that component is truly necessary to produce the data the model is producing. Consequently, we varied architectural parameters alongside removing knowledge-level model components to strongly test for the necessity of each component.

3.2.1 HALBRÜGGE'S MODEL

Halbrügge's model has three different strategies concerning what the model thinks the environment flow rate function is (zero, constant, or changing linearly), and these are crossed with two different strategies (practical and analytical) concerning how the model will try to get the water level back to the goal, given the predicted environment flow rate.

Both the analytical and practical approaches offset the predicted environment flow rate and correct for deviations between the current and goal water levels to try to get the water level back to the goal. However, these strategies differ in how the current water level in the tank is actually determined. The analytical approach is to read the water level off the meter, which is always accurate, and can go above and below tank capacity (i.e., above ten gallons and below zero gallons). Alternatively, the practical approach is to realize that the tank only holds ten gallons, and it's not possible for a tank to hold a negative amount of water, so water levels in the tank above ten gallons should be truncated to ten gallons, and water levels below zero gallons should be truncated to zero gallons.

This means that the practical and analytical strategies are only different when water levels in the tank are either above ten gallons or in the negative. The majority of the data had water levels ranging between zero and ten gallons. Since these strategies are equivalent within this range, we wondered if having both the practical and analytical strategy present was necessary for the model to perform the task. Towards this end, utility noise (:*egs*) and learning rate (:*alpha*) were varied from .01 to .99 while running the model through the development and transfer datasets with practical-only, analytical-only, and both strategies enabled.

A large amount of overlapping variance (i.e., equivalent model performance) between the strategy runs was found for the majority of the conditions – particularly conditions in the development dataset – which makes sense given that these conditions did not have a large amount of data (from participants or the model) with substantially large or small values in the tank.

However, large tank deviations were observed for participants in the difficult delay-3 condition, and the performance of Halbrügge's model does change quite drastically in this condition depending on if the practical or analytical strategy is being used. Included in Figure 3 is the performance (i.e., RMSE) surface of Halbrügge's model when each of the two strategies is enabled, across the entire range of utility noise (:*egs*) and learning rate (:*alpha*) combinations. Note particularly how the performance surface is orders of magnitude worse (i.e., higher) when only the analytical strategy is being used (compared to using only the practical strategy) for all combinations of utility noise and learning rate. Because there is no combination of utility noise and learning rate that allows the model with only the analytical strategy enabled to have an average absolute error in the tank less than 1 million, the analytical strategy alone cannot handle the difficult delay-3 condition.

The practical strategy can handle the delay-3 condition, whereas the analytical strategy cannot, because the amount of correction available to the practical strategy is limited to ten gallons, whereas the analytical strategy has no limit on the amount of correction. The practical

strategy can be characterized as adopting the perspective that you can't put more into a tank than it can hold. For the delay-3 condition, input is delayed until two trials after it is given, so over correction becomes fairly likely. However, if the amount of correction possible with the model is constrained, as in the case of the practical strategy, the desirable result is the avoidance of over-corrective oscillations that spiral out of control.

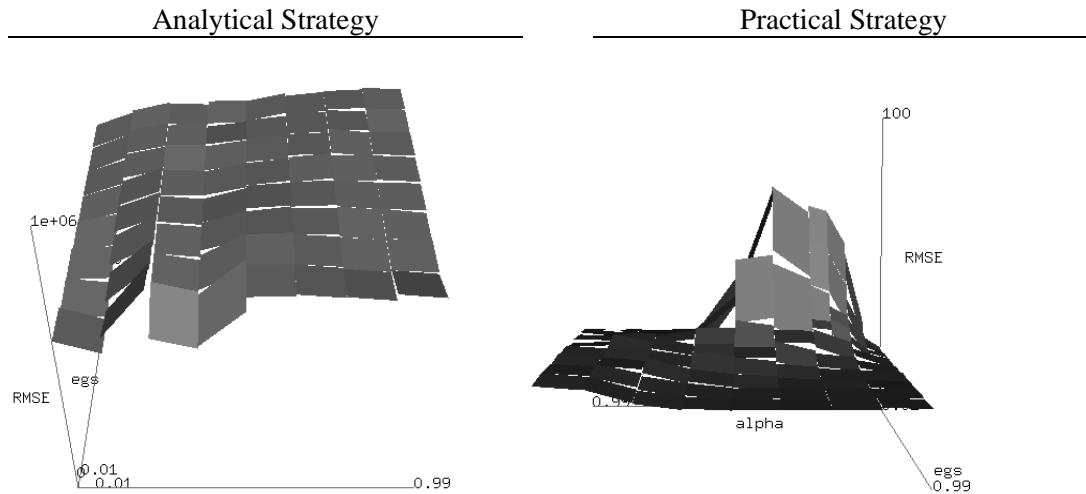


Figure 3: Performance surface of Halbrügge's model for the delay-3 condition, with the analytical (left) and practical (right) strategies enabled

Given that the two strategies are mathematically equivalent when large tank deviations are not observed (all conditions except for the delay-3 condition), and only the practical strategy can handle the delay-3 condition, it's questionable whether the analytical strategy is necessary at all, or if the practical strategy alone would suffice. However, to strongly test this claim, one must show that model performance for the delay-3 condition with just the practical strategy is equivalent to model performance when both strategies are enabled (show that the analytical strategy isn't necessary), and that model performance decreases when the practical strategy is removed (show that the practical strategy is necessary).

Towards this end, the model's best performance (i.e., best-fitting combination of utility noise and learning rate) with each and both strategies enabled was run through a Monte Carlo simulation ($N=500$ for each strategy run). The trial-level plots are included in Figure 4 and fit statistics are in Table 3. First, note that model performance is much worse when the practical strategy is removed (compare the left and right plots, noting the difference in scaling), which means that the practical strategy is necessary in the delay-3 condition. This is to be expected, since no combination of utility noise and learning rate allowed the analytical strategy alone to handle the delay-3 condition (see Figure 3).

However, also note that model performance degrades slightly when the analytical strategy is removed (compare the left and middle plots), which means that the analytical strategy is also necessary in the delay-3 condition. So, although the analytical strategy alone causes the model to spiral out of control, using this suboptimal strategy every once in a while alongside the practical strategy allows the model to fit human subjects better than the practical strategy alone, regardless of the utility noise and learning rate used.

Therefore, both the practical and analytical strategy are necessary for the delay-3 condition, and it's not the case that only the practical strategy is sufficient to produce the model-human fits that Halbrügge's model is capable of generating for all the conditions (particularly the delay-3

condition). Nonetheless, because the practical strategy is the only strategy that can handle the delay-3 condition (which most participants can handle), and both strategies are equivalent for all other conditions, having the practical strategy enabled is much more necessary than having the analytical strategy enabled (by ~5 orders of magnitude, if judging by the delay-3 condition alone).

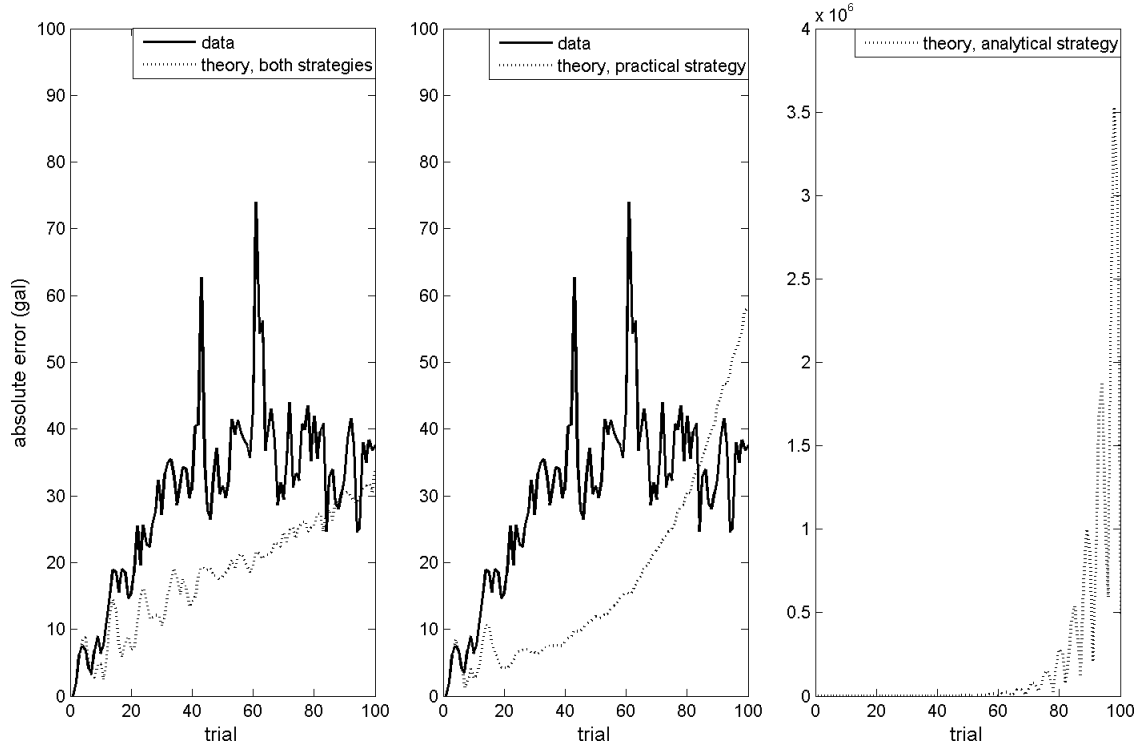


Figure 4: Trial-level plots of Halbrügge's model for the delay-3 condition, at best-fitting utility noise (:*egs*) and learning rate (:*alpha*) values, with practical (:*egs*=.93, :*alpha*=.44), analytical (:*egs*=.19, :*alpha*=.07), and both (:*egs*=.68, :*alpha*=.17) strategies enabled (*N*(subjects)=60, *N*(model)=500 for each strategy run)

	Strategy		
	Both	Practical Only	Analytical Only
RMSE	15.00	19.71	624,739.20
r	.68	.39	.13

Table 3. Root Mean Squared Error (RMSE) and correlation (*r*) fit metrics for the performance plots in Figure 4.

3.2.2 REITTER'S MODEL

Reitter's model chooses between a 'calculation' and 'estimation' strategy to determine the user flow rate for the next trial. Both strategies offset the predicted environment flow rate and correct the deviation between the current and goal water levels to try to get the water level back to the goal. Additionally, both strategies predict the environment flow rate for the next trial by adding the 'trend' of how the flow rate is changing from trial to trial to the flow rate of the previous trial. However, these strategies differ in how this trend is actually calculated.

The calculation approach is to take the difference between the environment flow rate of the last two previous trials (no retrieval errors, no noise), and use this value as the trend. This is equivalent to Halbrügge's 'changing linearly' strategy, except that Reitter's model simulates cognitive arithmetic to arrive at the estimate. Alternatively, the estimation approach stores the environment flow rates for all previous trials in declarative memory, and then tries to retrieve the flow rate for the 2nd previous trial when calculating the trend (the flow rate for the 1st previous trial can be read directly off the display). Noise in declarative memory retrieval mechanisms could cause a much older environment flow rate to be returned, leading to a misretrieval and a miscalculation of the trend between the previous two trials. These trends are then stored in declarative memory (whether correct or incorrect), and a future trend is estimated as a blend of the trends from all the previously experienced trials, weighted by each memory's activation (trends from more recent trials weighted more).

We suspected that the estimation strategy would be necessary for conditions where multiple past trials should be taken into account (e.g., the delay and sequence conditions in the transfer dataset). However, we weren't as certain that the calculation strategy would be necessary for a condition in the development dataset, or if the estimation strategy alone would suffice. Towards this end, the transient noise (.01-.99), base-level learning decay rate (.01-.99), and blend temperature (.01-10) were varied while running the model through the development and transfer datasets with calculation, estimation, and both strategies enabled. Once again, blend temperature did not seem to drastically influence model performance – at least not on the order that transient noise and decay rate were influencing performance – so results across blend temperature were collapsed, leaving a 2-parameter analysis at the default .5 blend temperature.

As anticipated, the estimation strategy did better than the calculation strategy for conditions where multiple past trials should be taken into account (i.e., the delay and sequence conditions in the transfer dataset). However both strategies performed well in the development dataset, so we questioned if having both strategies present was necessary for the model in this dataset, or if the estimation strategy alone would suffice for all the conditions in the DSF Challenge (i.e., both datasets).

Included in Figure 5 is the aggregated performance (i.e., RMSE) surface of Reitter's model collapsed across all four conditions in the development dataset, when each of the two strategies is enabled, across the entire range of transient noise (:ans) and decay rate (:bll) combinations. Note particularly how the average height of both surfaces is about the same (similar overall performance), but that the shape of the surfaces is slightly different. The calculation surface is relatively flat across all transient noise and decay rate combinations, suggesting that model performance for the calculation strategy is roughly independent of transient noise and decay rate values. This makes sense, given that the calculation strategy does not use ACT-R retrieval mechanisms when determining the trend of the environment flow rate (no retrieval errors, no noise).

However, the height of the estimation surface is influenced both by transient noise and decay rate values, and performance is best (i.e., surface height is lowest) when both values are low. This is due to the fact that manipulating these values alters the signal to noise ratio (S/N) of the activation of a chunk in declarative memory. Increasing transient noise increases the noise, whereas decreasing the decay rate increases the signal. This S/N ratio determines the likelihood of retrieving the past environment flow rate. Higher S/N leads to higher likelihood of correct retrievals. Lower S/N leads to higher likelihood of misretrievals, which results in an accumulation of incorrect memories, which in turn leads to a propagation of error when those trends are blended over to determine the trend for the next trial. This means that low S/N leads to worse performance and high S/N leads to better performance.

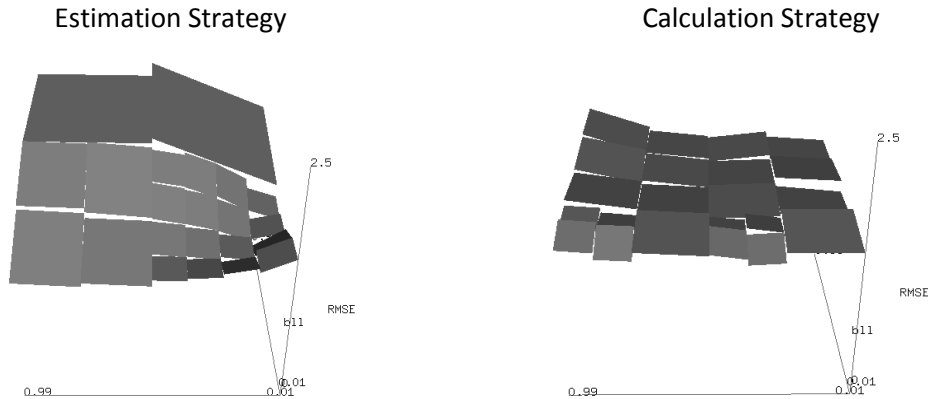


Figure 5: Aggregated performance surface of Reitter's model collapsed across all conditions in the development dataset, with the estimation (left) and calculation (right) strategies enabled

Further, the lowest regions of the estimation surface (i.e., low values for transient noise and decay rate, high S/N) actually cross below the height of the calculation surface, demonstrating that the particular combination of transient noise and decay rate values used can determine which strategy (estimation or calculation) performs better on conditions in the development dataset.

Although the average height of the calculation and estimation surfaces in Figure 5 is roughly the same, having different shapes suggests that the strategies may be getting similar performance (i.e., similar RMSE) in completely different ways (e.g., better fits in earlier trials and worse fits in later trials or vice-versa). To explore this possibility, the model's best performance configuration (i.e., best-fitting combination of transient noise and decay rate) with each and both strategies enabled was run through a Monte Carlo simulation ($N=500$ for each strategy run) for each condition in the development dataset (12 runs total). The trial-level plots are included in Figure 6 and the fit statistics are in Table 4. Due to the fact that the model's performance in the calculation strategy did not depend on the particular value of transient noise and decay rate (calculation surface is flat in Figure 5), the parameter values used by Reitter were used for these four runs (one run for each condition) as the best-fitting parameter values.

The estimation strategy performs similarly to human in the earlier trials, but the fit gets worse for later trials (too high an absolute error). Alternatively, the calculation strategy fits the model quite well in later trials, but not so well in earlier trials (too low an absolute error). Because the overall fit (i.e., RMSE) of either strategy is about the same, the performance surface of each strategy has roughly the same height. However, the trend of deviations between tank and goal at the trial level for these strategies is completely different. The estimation strategy is a better fit to human data for earlier trials, while the calculation strategy fits better for later trials.

Further, the fit when both strategies are enabled is better than using either the estimation or calculation strategy alone. Reitter's model has a metacognitive mechanism that monitors the success of each strategy on previous trials and uses the more successful strategy for the next trial. This allows (when both strategies are enabled) the model to shift from using the less accurate (i.e., higher absolute error) estimation strategy for earlier trials to using the more accurate (i.e., lower absolute error) calculation strategy for later trials. This transition from estimation to calculation strategy as trials progress fits the participant data quite nicely, where using either strategy alone causes issues with the fit (estimation fits worse for later trials, calculation fits worse for earlier trials). Therefore, having both strategies present in the model is indeed necessary to fit participant data for conditions in the development dataset, and each strategy is about as

necessary as the other, because participants appear to transition from using the estimation to calculation strategy as the trials progress.

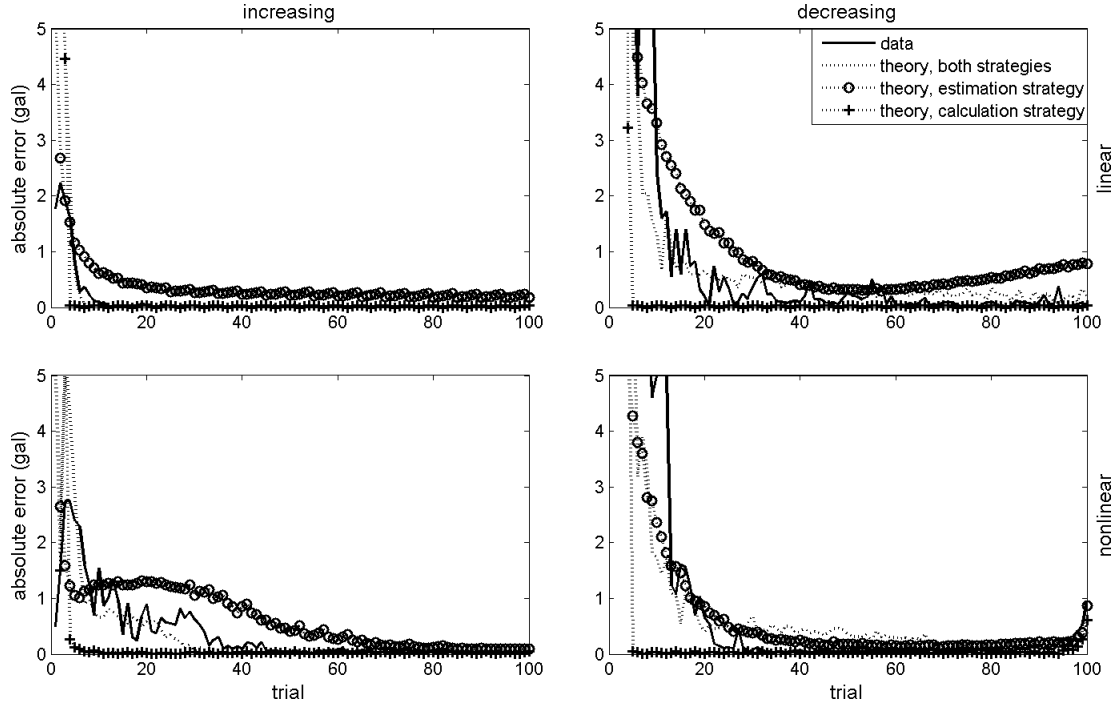


Figure 6: Trial-level plots of Reitter's model for all conditions in the development dataset, at best-fitting decay rate (:bll) and transient noise (:ans) values, with calculation (:bll=.5, .5, .5, .5, :ans=.25, .25, .25, .25), estimation (:bll=.19, .07, .32, .19, :ans=.07, .44, .07, .44), and both (:bll=.09, .33, .09, .33, :ans=.32, .93, .81, .93) strategies enabled. $N(\text{subjects})=15, 11, 18, \& 17$ for conditions left to right & top to bottom. $N(\text{model})=500$ for each strategy run for each condition.

Linear					
		Increasing		Decreasing	
		RMSE	r	RMSE	r
Both		.61	.95	2.39	.87
Estimation		.44	.86	2.89	.78
Calculation		.61	.87	2.83	.75
Nonlinear					
		Increasing		Decreasing	
		RMSE	R	RMSE	r
Both		.59	.77	1.12	.92
Estimation		.69	.53	1.53	.90
Calculation		.78	.37	2.55	.75

Table 4. Root Mean Squared Error (RMSE) and correlation (r) fit metrics for the performance plots in Figure 6.

4. General Discussion and Conclusion

Exploration via sensitivity and necessity analyses can improve our understanding of the sources of power and the performance dynamics of cognitive models and other forms of artificially intelligent systems. We performed such analyses with two of the best models from the DSF Model Comparison Challenge. Using large scale computational resources combined with the intelligent search and visualization tools, Cell and Tornado, we performed a sensitivity analysis (varying parameter values) alongside a necessity analysis (removing/adding architectural or strategic components), just as Estes recommended. The results allowed us to investigate not only the degree to which optimal parameter values change from one condition to the next, but also how those results vary as a function of the specific architectural components that are enabled. Comparing the optimal parameter values when both strategies are enabled (see Table 1) to optimal values when each of the strategies is removed (see Figures 4 and 6) shows the magnitude of the variance present in optimal values returned by the search algorithm (Cell) for both Reitter's and Halbrügge's models.

One thing evidenced by the complex results is that model "appropriateness" is not a binary state, but rather a continuum. In the sensitivity analysis, for example, it was clear that while both models had selected reasonable architectural parameter values, Halbrügge's model was relatively insensitive, performing well across a wide range of values. Reitter's optimal parameter values, on the other hand, were varied and contradicting across conditions. The trade space was complex and difficult to grasp without visualizing the response surfaces. And while both models demonstrated necessity for their components, Reitter's model relied more heavily on both strategies than did Halbrügge's.

Striving toward a more appropriate model raises important questions concerning what model parameter values should (and do) represent. Large scale computational resources and efficient search algorithms empower the modeler to find optimal parameters, and more importantly to understand the structure and interrelationships among those parameters. At that point, one can then reason about what precisely these values represent (e.g., the gravitational constant, Planck's constant, the charge of an electron, knowledge decay rate) that transcends the specific task that's studied, and is more of (for cognitive modelers) a product of the general cognitive processes that are used to perform any task. In short, we're looking for canonical defaults within a very large space.

With an efficient search algorithm and HPC resources at your fingertips, it is tempting to take a model, use a standard fitness function, and blindly optimize all of the model's components. However, this approach can be problematic because it does not answer why a particular point in the space is the optimal model configuration. Instead, we advocate using HPC resources and exploration and search algorithms as tools to quickly and efficiently find model configurations that make interesting (possibly counterintuitive) predictions, expanding the space of manipulations to which the architecture and model are exposed. In this way, efficient, large-scale model exploration can more quickly advance our understanding of both human and artificial cognitive systems.

Acknowledgements

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This research was sponsored by the U.S. Air Force Research Laboratory, 711th Human Performance Wing, Human Effectiveness Directorate, Warfighter Readiness Research Division.

References

- Alexander, W. P., and Grimshaw, S. D. 1996. Treed regression. *Journal of Computational and Graphical Statistics*. 5: 156-175.
- Anderson, J. R. 2007. *How can the human mind occur in the physical universe?* Oxford University Press, Oxford, UK.
- Bush, R. and Mosteller, F. 1955. *Stochastic Models for Learning*. John Wiley & Sons, New York.
- Dutt, V., and Gonzalez, C. 2007. Slope of inflow impacts dynamic decision making. In *Proceedings of the 25th International Conference of the System Dynamics Society* (pp. 79). Boston, MA: System Dynamics Society.
- Estes, W. K. 2002. Traps in the route to models of memory and decision. *Psychonomic Bulletin & Review*, 9(1), 3-25.
- Friedman, J. 1991. Multivariate adaptive regression splines. *The Annals of Statistics*. 19: 1-141.
- Gluck, K. A., Scheutz, M., Gunzelmann, G., Harris, J., and Kershner, J. 2007. Combinatorics meets processing power: Large-scale computational resources for BRIMS. In *Proceedings of the Sixteenth Conference on Behavior Representation in Modeling and Simulation*, 73-83, Orlando, FL: Simulation Interoperability Standards Organization.
- Gonzalez, C., and Dutt, V. 2007. Learning to control a dynamic task: A system dynamics cognitive model of the slope effect. In *Proceedings of the 8th International Conference on Cognitive Modeling*, 61-66. Ann Arbor, MI.
- Halbrügge, M. in press. Keep it simple – A case study of model development in the context of the Dynamic Stock and Flows (DSF) task. *Journal of Artificial General Intelligence*.
- Kase, S.E., Ritter, F.E., and Schoelles, M. 2007. Using HPC and PGAs to optimize noisy computational models of cognition. In *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*.
- Kintsch, W., Healy, A. F., Hegarty, M., Pennington, B. F., Salthouse, T. 1999. Models of working memory: Eight questions and some general issues. In A. Miyake and P. Shah (Eds.) *Models of working memory: Mechanisms of active maintenance and executive control*. (pp. 412-441). New York: Cambridge University Press.
- Knofczynski, G. T., and Mundfrom, D. 2008. Sample sizes when using multiple linear regression for prediction. *Educational and Psychological Measurement*. 68:431-442.
- Langley, P., Laird, J. E., and Rogers, S. 2008. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2), 141-160.
- McClelland, J. L. 2009. The place of modeling in cognitive science. *Topics in Cognitive Science*, 1, 11-38.

- Moore, L. R. 2010. Cognitive Model Exploration and Optimization: A New Challenge for Computational Science. In T. Jastrzembski (Ed.), *Proceedings of the 19th Behavior Representation in Modeling and Simulation (BRIMS) Conference*. Charleston, SC.
- Moore, L. R., Kopala, M., Mielke, T., Krusmark, M., and Gluck, K. A. (2010). Simultaneous Performance Exploration and Optimized Search with Volunteer Computing. In *Proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC)*. Chicago, IL.
- Pew, R. W., Gluck, K. A., and Deutsch, S. 2005. Accomplishments, challenges, and future directions for human behavior representation. In K. A. Gluck and R. W. Pew (Eds.) *Modeling human behavior with integrated cognitive architectures: Comparison, evaluation, and validation* (pp. 397-414). Mahwah, NJ: Lawrence Erlbaum Associates.
- Reitter, D. In press. Metacognition and multiple strategies in a cognitive model of online control. *Journal of Artificial General Intelligence*.
- Rescorla, R. A., and Wagner, A. R. 1972. A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black and W. F. Prokasy (Eds.) *Classical conditioning II. Current research and theory*, 64-99. New York: Appleton-Century-Crofts.
- Wallach, D., and 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.