Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 29–30 (2014) 53–65

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

An interpolation approach for fitting computationally intensive models

L. Richard Moore Jr. [a,*], Glenn Gunzelmann [b,1]

[a] L3 Communications, 5950 East Sossaman Road, Suite 121, Mesa, AZ 85212, United States
[b] Cognitive Models and Agents Branch, Air Force Research Laboratory, 2620 Q St., Building 852, Wright Patterson AFB, OH 45434, United States

Received 9 July 2013; accepted 7 September 2013
Available online 25 September 2013

Abstract

Computational cognitive modeling has been established as a useful methodology for exploring and validating quantitative theories about human cognitive processing and behavior. In some cases, however, complex models can create challenges for parameter exploration and estimation due to extended execution times and limited computing capacity. To address this challenge, some modelers have turned to intelligent search algorithms and/or large-scale computational resources. For an emerging class of models, epitomized by attempts to predict the time course effects of cognitive moderators, even these techniques may not be sufficient. In this paper, we present a new methodology and associated software that allows modelers to instantiate a model proxy that can quickly interpolate predictions of model performance anywhere within a defined parameter space. The software integrates with the R statistics environment and is compatible with many of the fitting algorithms therein. To illustrate the utility of these capabilities, we describe a case study where we are using the methodology in our own research.

Keywords: Cognitive moderator; Mathematical model; Cognitive model; Model proxy

1. Introduction

Most cognitive models contain parameters that may be adjusted in order to match human performance as closely as possible. For example, models that include prior knowledge might parameterize how well learned each piece of knowledge is, as well as how easily confused it is with other knowledge. Learning models may require an estimation of learning rate. These are just a couple of examples reflecting general attributes of cognition that may be tuned for particular people, tasks, and contexts (see Wong, Cokely, & Schooler, 2010, for some examples).

The tunable parameters of a model define a *parameter space*, with dimensionality the same as the number of model parameters. In fitting parameters to models, the range of the parameter space is sometimes constrained by imposing theoretical constraints on parameter values. Fitting a model to empirical data constitutes a search within the constrained parameter space to find the parameter values that best aligns the model's performance with humans.

Parameter fitting varies widely across modeling methodologies, formalisms, and situations. For example, some models may support mathematical manipulation to determine the precise location of optima via the roots of a derivative. For some other types of models, including computational models that run in simulation, it is common practice to fit using a trial-and-error technique comparable to mentally performing a gradient descent. As the dimensionality of the parameter space increases, however, the viability of this approach is undermined.

In some cases, higher dimensional computational models may exhibit predictable relationships between the model parameters and performance measures. Such relationships allow the high dimensional parameter space to be approximated by several lower dimensional spaces, which can be more easily managed with previously mentioned techniques. Unfortunately, this is not always

* Corresponding author. Tel.: +1 480 988 3944x110.
E-mail addresses: lrmoorejr@gmail.com (L. Richard Moore Jr.), glenn.gunzelmann@us.af.mil (G. Gunzelmann).
[1] Tel.: +1 937 938 3554.

1389-0417/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cogsys.2013.09.001

the case. Software-based models that are particularly slow to execute or that have a complex high dimensional parameter space may make manual fitting solutions intractable. In these cases, modelers can turn to intelligent search algorithms, large-scale computational resources, or a combination thereof.

Intelligent search algorithms automate computational model fitting while minimizing the number of model runs. Techniques including adaptive mesh refinement (Best et al., 2009), parallel genetic algorithms (Kase & Ritter, 2009), simulated annealing (Raymond, Fornberg, Buck-Gengler, Healy, & Bourne, 2008), hill climbing (Kase, 2008), and stochastic regression trees (Moore, 2011) have all been proposed for fitting computational cognitive models in large, complex parameter spaces.

Large-scale computational resources allow model runs to be distributed among many processors simultaneously. Each tunable model parameter can be enumerated within its theoretically reasonable range using fixed increments. The resulting combinations of parameter values constitute a grid (see Fig. 1), and the points in the grid can be evenly distributed among available computational resources for parallel execution. To further reduce computational overhead, some modelers have implemented intelligent search algorithms on large-scale computing systems (Kase & Ritter, 2009; Moore, 2011), which combine the benefits of both approaches to reducing model exploration and optimization times.

For some classes of models, the computational challenge of fitting is problematic even when the combination of intelligent search algorithms and large-scale computational resources are applied. For example, there has been recent interest in models that predict or explain the mechanisms behind cognitive moderators such as caffeine (Kase, Ritter, & Schoelles, 2009), emotions (Belavkin, 2001; Marinier, Laird, & Lewis, 2009), time-on-task effects (Gonzalez, Best, Healy, Kole, & Bourne, 2011; Gunzelmann, Moore, Gluck, Van Dongen, & Dinges, 2010), sleep loss (Gunzelmann, Gross, Gluck, & Dinges, 2009), stress (Ritter, Reifers, Klein, & Schoelles, 2006), and aging (Meyer, Glass, Mueller, Seymour, & Kieras, 2001). Furthermore, Dancy, Ritter, and Berry (2012) propose a link between architectural parameters and thirst, hunger, and the startle response.

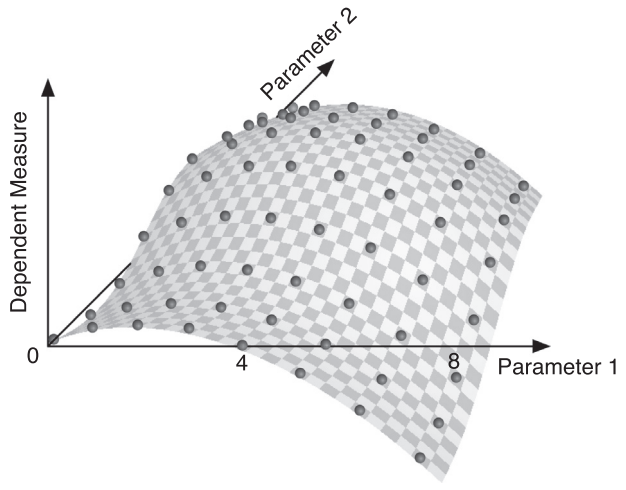

Fig. 1. A full combinatorial mesh can be sampled to construct a response surface. This space is comprised of two continuous parameters, both sampled from 0 to 7 with a step size of 1. The spheres represent the sampled points of the actual surface, which is checkered. The vertical axis represents one of the model's dependent measures as sampled at each point in the grid.

The effects of many cognitive moderators unfold according to specific, well-known mathematical signatures. For example, drug influence is usually described as an exponential decay quantified by half-life (for example, Reder et al., 2007). Giambra and Quilter (1987) used a two-term exponential function to describe the decline in sustained attention over time. The sleep research community has generated a number of mathematical models that describe the homeostatic and circadian effects of sleep loss on alertness (for examples see McCauley et al., 2009; Neri, 2004). Lastly, numerous physiological influences on cognition have been represented mathematically in systems like Hummod (Hester et al., 2011) or physiologically-based pharmacokinetic models (e.g., Gearhart, Robinson, & Jakubowski, 2009; Gerlowski & Jain, 1983).

Because the effects of cognitive moderators vary distinctly over time, a thorough behavioral model validation suggests fitting performance at multiple points in time. Testing the fit of a single parameter combination then involves evaluating the model multiple times to compare against data at critical points in the time series. This effectively multiplies the computational burden of the fitting process by the number of sessions involved, which can be problematic for slow, highly parameterized models, even with the benefit of large-scale computational resources and intelligent search algorithms.

Given this computational burden, an alternative to fitting the entire time course is appealing. For example, one could find the best fits for each session independently and subsequently report a correlative statistic across the entire time series to demonstrate a relationship between the moderator function and model parameters. This is similar to an approach we have used successfully in some of our earlier work (e.g., Gunzelmann, Gluck, Moore, & Dinges, 2012; Gunzelmann, Moore, et al., 2009). It is a reasonable approach for circumstances where the empirical noise is low and there is little colinearity between model parameters. However, if either condition is unknown or not met, this approach can be problematic.

For example, consider the case where there are multiple parameters that have some colinearity. Under these circumstances, there may be a *region* of the parameter space where similarly good fits to human performance are observed.[2] In these cases, stochasticity in the model's

[2] Colinearity among model parameters rightfully raises concerns about over-parameterization, however, it may be limited to only some sessions of the empirical data.

performance will impact the selection of optimal parameter values within the region. The resulting stochastically determined parameter values are likely to be misleading with regard to the relationship of the parameters to the moderator function, if one exists. Noise in the empirical data can disrupt regressions similarly by obscuring systematic underlying relationships.

By fitting the model against the entire time course simultaneously, the impact of these issues can be reduced while maximizing statistical power, because fitting will use the full set of empirical data rather than subsets of the data relating to particular values for the moderator. This will become particularly important as research on topics such as sleep loss, stress, alcohol and age begin to focus on individual differences.

Unfortunately there are few tools and little guidance in the literature to support such efforts. This paper begins to fill that gap by describing a fitting approach that we used for our fatigue research to fit empirical data from individual participants at the resolution of their performance in individual sessions. We begin with a description of the methodology in the next section, and later describe a specific case study.

2. Procedure

The fitting methodology described in this paper involves two stages. In the first stage the behavior of the model is captured by sampling a uniform grid as described in the introduction for large-scale computational resources. An intelligent search algorithm is then used in the second stage to mine the data and locate optima. The intelligent search algorithm will require model samples that fall between nodes on the sampled grid, and rounding may introduce undesirable error in the performance estimates of the model. Therefore, an interpolation algorithm is used to predict model behavior so the model appears to operate continuously across the discretely sampled parameter space for the search algorithm. The software that produces the interpolations is called Sif, and that will be described in more detail later. First we will discuss options for obtaining the model data across a uniform grid.

2.1. Obtaining a uniform grid of samples

Constructing an evenly sampled full mesh such as shown in Fig. 1 begins with the selection of parameter ranges, which will define the boundaries of the available parameter space, and parameter increments, which will influence the accuracy of the interpolated values. A basic understanding of the model's performance characteristics is required when selecting these values.

Appropriate spacing for the initial parameter grid is essential, and will likely require some experimentation and testing to determine the appropriate increments for each parameter. Abrupt changes in the influence of parameters, or non-monotonicity, could negatively impact the quality and accuracy of the interpolation. As a starting point, a visualization of the response surface can show the range of valid behavior and the gradients can help prescribe parameter increments (Gluck, Stanley, Moore, Reitter, & Halbrugge, 2010). Once decided, large-scale computational resources can be used to sample the parameter space at each point until the central tendency is revealed. This paper will describe several potential approaches to acquiring these data.

The first, and most likely simplest, option available to cognitive modelers is a high-performance computing grid called MindModeling (Harris, 2008; see http://mindmodeling.org to get started). MindModeling pools computational resources from individual volunteers and high performance computing clusters to sample model parameter spaces. The system was originally designed for ACT-R models, but recent additions now expand support to include models written in Lisp, Python, R, Java, or any compiled software.

MindModeling provides a web page submission system to modelers (see Fig. 2). Here the modeler specifies a list of independent variables (i.e. free parameters in the model) including the range and step size for each, the dependent variables (i.e. performance measures), administrative information, and the model itself. Once submitted, MindModeling distributes the model to available computational resources for execution. The programmatic interface between MindModeling and the model varies depending on the type of model, but it is kept as simple as possible. In some cases MindModeling will simply call a specified function with parameter values and expect the model to print out a labeled list of dependent measures in response. For stochastic models, the callback function will need to contain the necessary logic to repeatedly run the model to produce a reliable measure of central tendency before reporting aggregated results.

When the grid is complete, MindModeling provides the results in a simple text file named "results.txt" that enumerates the model's dependent measures at each point in the grid. Appendix A shows a partial listing of a results.txt file generated from MindModeling. The file is suitable for importing into most analysis tools such as R. It also produces a configuration file (hurricane_config_file.txt) that describes the format of the parameter space and is compatible with the interpolation tool described later in this paper, called Sif (see Appendix B for an example).

Distributed high performance computing (HPC) clusters provide an alternative option to modelers with access to such resources, which may be the case for many government and academic employees. Unfortunately the specific usage of such systems varies, so detailed information must be obtained from the particular system's documentation and administrators. Here we describe the general approach to using a distributed Linux cluster.

From the user's perspective, a distributed Linux HPC cluster appears much like a large network of individual Linux systems all attached to the same file server. Several "head nodes" allow interactive Linux access via command line shell. From the head nodes, users can create job scripts

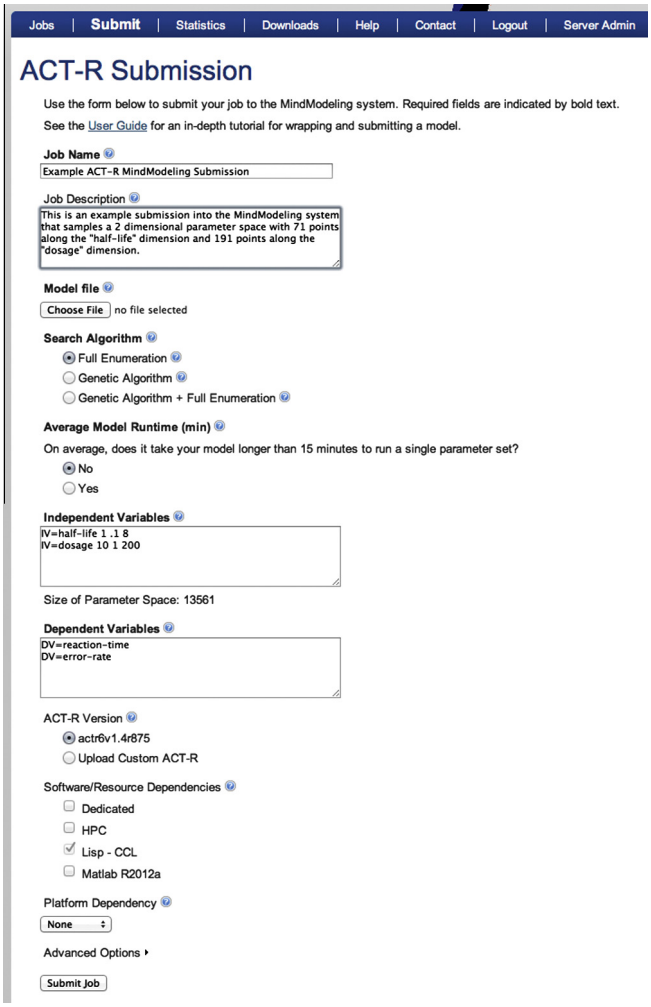

Fig. 2. An example ACT-R model submission to construct a 2 dimensional full combinatorial mesh (71 × 191) with MindModeling.

and submit them into a queue for execution. Some basic knowledge of shell scripting will be necessary. Specially formatted comment lines at the top of the job script provide details to the scheduler about things such as how many processing cores to reserve, how long they need to be reserved, and what account to bill the time against. The scheduler executes the job script on computational resources when they become available.

The simplest strategy to make use of distributed Linux clusters is to create a small program that generates and submits job script files. The generated files should contain the code to execute the model at each unique combination of parameter values in the space and save the results on the file server. The potential for parallelization is maximized when each job script runs the model for a single parameter combination in the space, but this can require millions of job scripts, which is impractical for most schedulers. Instead, multiple points in the parameter space can be managed in a single job script. The maximum number of job scripts to generate will depend upon the limitations imposed by the scheduler and the administrators of the system. Most systems allow you to see queued jobs, and this information can be used to get a feel for the distribution of job submissions that are tolerated on any particular system. As with MindModeling, the logic of running stochastic models repeatedly and aggregating the results is the responsibility of the modeler. Appendix C provides an example script that was run on a DoD distributed Linux HPC system as an example but it will require heavy adaptation for use on another cluster.

A third option also makes use of HPC systems, but it does so more robustly and usually more efficiently than the previous method. We have developed a piece of software called Cell, which is designed to perform cognitive model searches and explorations of parameter spaces on HPC systems. One capability of Cell is to sample a model into a grid space such as the one desired here. Only one job script is necessary to run Cell on an HPC, and that job script can be queued as many times as is reasonable/desired. This relieves the modeler of the burden of writing software to generate and submit job scripts and submitting them in batches as described in the previous paragraph.

The job script should be defined so that each Linux system in the cluster runs a single instance of Cell. The Cell software randomly selects a point in the mesh grid and repeatedly samples until a configured level of standard error in the model's performance is obtained. Note that unlike the other two approaches described above, the modeler does not need to run the model repeatedly and aggregate the results – that functionality is built into Cell. Upon completing a point in the parameter space, Cell will repeat the process with another randomly selected point. Data are shared across systems via log files on the shared drive that are regularly scanned for updates. (A more efficient network broadcasting approach is also available for those HPC systems that allow it.)

Because the Cell software may be running across many machines at the same time, it is inevitable that a point will be selected that has already been completed by another computational node. In this case, Cell will either agree with the earlier assessment of standard error and select a different point, or it will continue adding samples until the standard error is within the preconfigured range. (A discrepancy in standard error calculation between two computational nodes can occur if there is a communication failure when writing or reading data for that node, or if the first node failed to complete sampling the point.) With this approach, many computational nodes double-check the work of their peers. Because data are shared across systems, two machines can select the same point in the parameter space and work on it at the same time. As each machine runs out of points to work on, the process gracefully winds down (see Fig. 3).

All three procedures produce data reflecting the central tendency of model performance at each point in the parameter space. The data will be used by Sif to provide quick predictions of model behavior or parameter values within the bounds of the grid searched. The data needs to be in the form of space-separated text files such as the partial

listing in Appendix A, where each line contains a list of parameter values followed by a list of model results. As mentioned above, MindModeling produces such a file as its normal output. Cell does as well, although it spreads the data across multiple files (which is fine for Sif). When using a non-Cell based HPC approach, it is incumbent upon the modeler to correctly format the results file(s).

2.2. Using Sif

Sif must be provided with meta-information about the result file's format before it can interpret it correctly. For example, it needs to know how many parameters were used and how many measures from the model were taken so it knows how many columns of data to expect. When using MindModeling to acquire the model data, this file is generated automatically for you (hurricane_config_file.txt). Otherwise, it must be created from scratch, but it was intentionally designed for simplicity (see Appendix B). Cell uses the exactly the same format for its configuration file that Sif does, so once created for Cell it can be recycled for use by Sif without modification.

The Sif server is started from the command line with an option that specifies the configuration file to use. The grid data file(s) can be quite large, and it can take several minutes for Sif to read through them, validate completeness, and organize its internal data structures. Therefore, Sif was designed as a client/server architecture so the server that loads that model data can be started and configured once for many client requests. Once started, the server remains idle until it receives a request from a client; it must be running to process client requests.

The Sif client issues queries by providing a set of parameter values to the Sif server by means of a network socket. The server interpolates the model prediction based on the nearest sampled points in the parameter space and returns the result. The process is analogous to the hypertext transfer protocol (HTTP) used by web browsers and servers to manage web page requests. Parameter values are provided instead of URLs, and model predictions are returned instead of a web page. The Sif client comes in the form of a command line program where the query results are printed out, or it can be run as a function within R and the data are returned as an array of values (R-Sif).

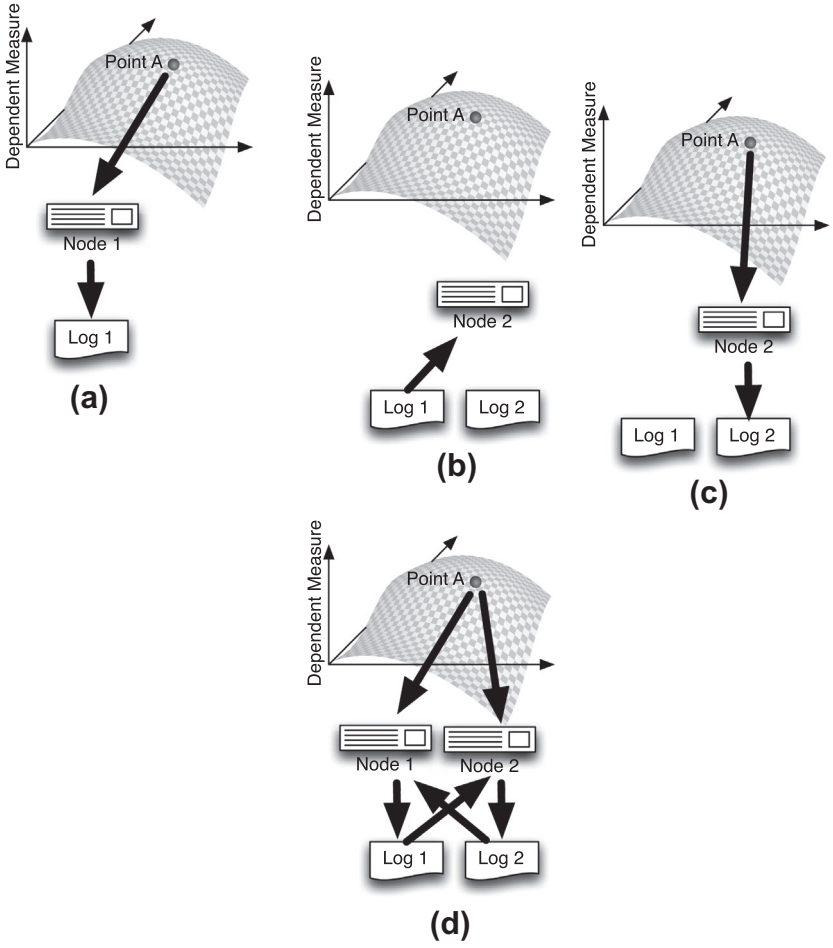

Fig. 3. (a) Node 1 selects and samples Point A until the standard error is within specifications, and then moves onto another randomly selected point. (b) Node 2 selects Point A but first examines the data collected by Node 1 to revalidate that standard error is within specifications. (c) If Node 2 finds that the standard error is not within specifications, it will continue sampling Point A until it is. (d) Occasionally Node 2 may select Point A before Node 1 has completed sampling. In this case, both nodes will sample the same point and monitor each other's progress until standard error is within specifications.

Regardless of the client used, the Sif server must already be initialized before it can service requests. Between client requests the Sif server can be stopped and restarted as desired because the clients do not maintain connections (much like a web server restarting between browser requests). The data grid can potentially be expanded during this time, if desired. The only requirements are that the new data plus the old data must still adhere to a uniformly sampled grid, and the configuration file must be updated to account for the new grid size and granularity of the combined data sets.

2.3. Sif's interpolation algorithm

The interpolation algorithm warrants a more detailed explanation. When considering a two dimensional parameter space, one might expect the response surface to be comprised of a grid of rectangles constructed out of the points of the mesh (see Fig. 4). However, this would be erroneous because each corner of the "rectangle" is measured independently and there is no guarantee that the four points are planar. Therefore an interpolation algorithm that can account for the inevitable warping in arbitrary dimensions is used to compute interpolated values:

$$V = \sum_{i=1}^{2^n} V_i \prod_{j=1}^{n} \left(1 - \frac{d_{ij}}{D_j}\right) \quad (1)$$

The equation sums the contributing influence of each measured sample, V_i, enclosing the requested point in an n dimensional space. The inner product, $\prod_{j-1}^{n}(1 - \frac{d_{ij}}{D_j})$, is the weight for each sample based on the orthogonal distance along each dimension, j. The orthogonal distance in a single dimension is simply $1 - \frac{d_{ij}}{D_j}$, where d_{ij} represents the distance from the interpolation point to the sample V_i along dimension j, and D_j represents the total distance along dimension j between the nearest surrounding samples (see Fig. 4).

Sif's computational performance is independent of grid size (i.e. O(1) in big-O notation) as long as the amount of data does not incur secondary effects such as page swapping from the operating system. Sif's performance at approximating model behavior will depend upon the grid granularity, with smaller sample increments producing more accurate results, as well as the shape of the response surface(s) at any given point.

In the next section we will introduce a case study featuring a model of the psychomotor vigilance task (PVT). The PVT is a reaction time task that has been shown to be highly sensitive to the impact of fatigue on cognitive processing (e.g., Dinges & Powell, 1985). Before delving into the details of the PVT, however, we would like to conclude this section by providing some evidence of Sif's performance. We do so by selecting two sets of parameters and comparing the PVT model's response with that predicted by Sif. The parameters were chosen such that they reside as far from the nearest grid samples as possible. In other words, they are centered within the surrounding grid

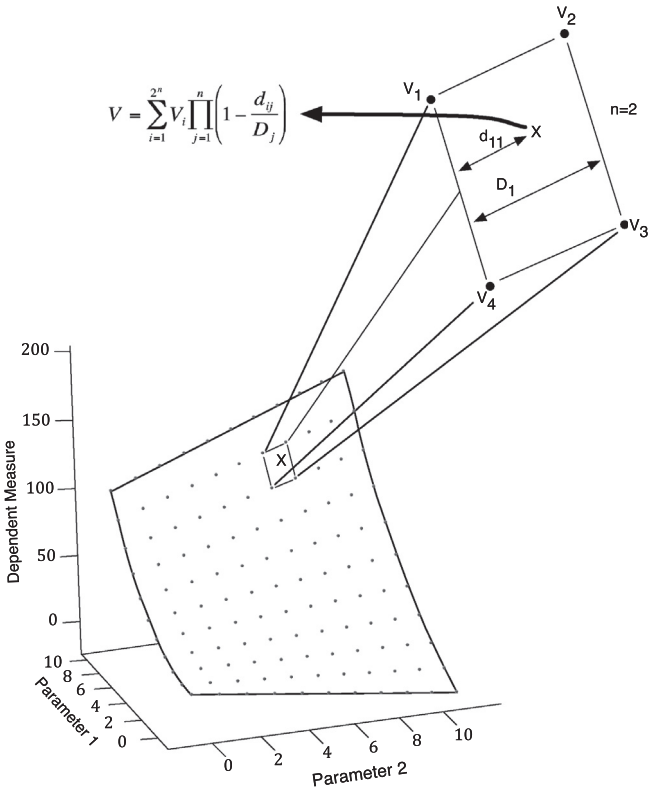

Fig. 4. The hypercubes formed by a uniformly sampled mesh are typically not planar. Interpolating a point in the middle of hypercube (represented by the "X") requires special consideration for the warping that can occur by the non-planar corners. The blow up section shows the enclosing nodes around point X, V1 through V4, and the relationships required to interpolate X in the non-planar rectangle using Eq. (1).

samples, thus requiring the most speculative interpolation possible. The results, shown in Fig. 5, are response time distributions. Both samples showed strong matches between Sif's predictions and the model's performance, with root mean squared errors of .0015 and .0021, and correlations of .999 and .997.

To contextualize the description of the software and methodology, we describe a specific case study in the next section. As foreshadowed, the case study focuses on fitting the PVT model within the context of our fatigue research (e.g., Gunzelmann, Gluck, Moore, & Dinges, 2012; Gunzelmann, Moore, Salvucci, & Gluck, 2011; Gunzelmann, Gross, et al., 2009).

3. Fatigue and the pyschomotor vigilance task

Much of our work on fatigue focuses on a model of the PVT, which is a reaction time task used frequently in research on sleep as an index of alertness (Dinges & Powell, 1985). The model is implemented in ACT-R (Anderson, 2007), and it is relatively fast compared to many other ACT-R models because of the limited knowledge required to execute the task.

Washington State University (WSU) has developed a mathematical approach to predicting fatigue in the form of abstract, quantified values (McCauley et al., 2009). Their

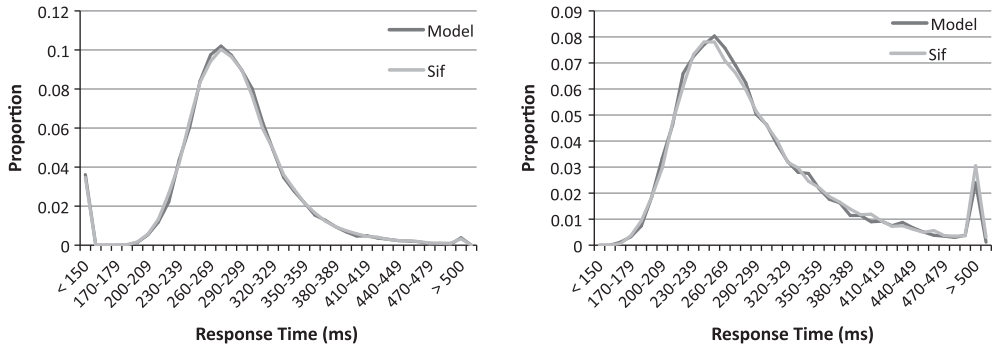

Fig. 5. Each plot represents a random sample from the PVT parameter space, showing the model's actual performance overlaid with Sif's predicted performance.

equations use sleep history combined with circadian and homeostatic effects as the basis for their predictions. We have previously demonstrated the benefits of linking fatigue predictions to parameters in ACT-R so that behavioral predictions can be made that span dependent measures and tasks (e.g., Gunzelmann et al., 2009, 2012, 2011). One advantage to binding fatigue predictions to architectural parameters is that predictions can be made in novel tasks without acquiring data under conditions of sleep loss (Gunzelmann & Gluck, 2009; Gunzelmann et al., 2011). Our goal was to determine the relationship between the WSU fatigue predictions and the ACT-R PVT model at level of fidelity such that we could make predictions about changes in individual performance across time during individual sessions distributed across days without sleep.

There are two parameters in ACT-R that we manipulate to account for degraded performance in the PVT resulting from sleep loss. These are *iu*, which represents *i*nitial production *u*tility, and *ut*, which represents production *u*tility *t*hreshold. They are both involved with the production selection and execution process.

By default, model productions are assigned a "utility" value of the amount specified by *iu*, and their utility plus a random noise component must be higher than *ut* to be eligible to fire. The closer *iu* is to *ut*, the greater the odds that an otherwise eligible production will not fire. If no production is executed, there is a gap in processing called a microlapse (Gunzelmann et al., 2009), which leads to further decrements on the production's utility. From a theoretical perspective, decreases in *iu* reflect degradations in alertness, whereas changes in *ut* capture the impact of effort, with lower values representing greater effort.

Assuming a linear relationship between the WSU fatigue predictions and the PVT model parameters, the influence of fatigue on iu and ut can be expressed as follows, where $B(t)$ is the biomathematical model prediction at time t:

$$\begin{aligned} iu &= \beta_1 B(t) + \beta_3 \\ ut &= \beta_4 B(t) + \beta_6 \end{aligned} \tag{2}$$

The complexity of the fitting challenge, however, is actually greater than this. Whereas the WSU predictions, $B(t)$, provide a general prediction of fatigue based on sleep patterns and time awake, they do not account for changes unfolding over shorter time frames, like time on task effects and the vigilance decrement (Ackerman, 2010; Warm, Parasuraman, & Matthews, 2008). For the PVT, we have shown that this can be approximated with an additional linear pressure on *iu* and *ut* as follows, where m is the number of minutes on task (Gunzelmann et al., 2010):

$$\begin{aligned} iu &= \beta_1 B(t) + \beta_2 m + \beta_3 \\ ut &= \beta_4 B(t) + \beta_5 m + \beta_6 \end{aligned} \tag{3}$$

Furthermore, we have shown that the processing cycle rate of the PVT model should be considered an individual difference (Gunzelmann, Moore, Gluck, Van Dongen, & Dinges, 2009), and so we also allow the corresponding parameter in ACT-R, *dat*, to vary across participants, though this parameter is constant across time awake and time on task. Therefore, to achieve our goal of fitting individual participants, it is necessary to find the optimal values for the 6 beta coefficients and *dat* for each participant in our experiment. In other words, fitting each individual requires searching a 7 dimensional parameter space.

The data set we are fitting against consists of 24 sessions spread across 3 days of sleep deprivation ($n = 26$), and we used a biomathematical model (McCauley et al., 2009) to generate predictions of fatigue, $B(t)$, at each session based on sleep history and time awake. Furthermore, we have aggregated the empirical data from each 10-min session into two "half-sessions" whose means are equivalent to the means of minutes 3 and 8 (see Fig. 6). This aggregation reduces noise in the data while retaining two within-session samples needed to estimate β_2 and β_5. As a result we have a total of 48 half-sessions of empirical data for each individual that we fit against.

Because the model is stochastic, multiple runs are required to reveal the central tendency for comparison to the empirical data. Typically for the PVT we use 100 runs for this purpose, so testing a single set of coefficients against 48 half-sessions would require 4800 model runs. Even for a small, fast model like the PVT this would take about an hour to run. This is avoided with Sif, which can quickly interpolate model predictions at each of the 48 sets of parameters without resorting to rounding techniques that would introduce undesired error.

The R statistical package (R Development Core Team, 2011) provides a variety of methods for searching parameter spaces. We have developed R-Sif, an R interface to execute Sif queries. R-Sif includes most of the same code that is compiled into the standard Sif client, except it is packaged as a shared library loadable by R and coupled with a few convenience functions written in R.

Once the R-Sif library is loaded and initialized, a *query.sif()* function can be used to interpolate and import data from the model parameter space into R. Each call interpolates one sample out of the model parameter space based on the architectural parameters (i.e. independent variables) supplied as an argument. The resulting dependent variables are returned as an ordered list.

The list returned from the query.sif() function can be compared to empirical data to produce a measure of fitness, whether it be Pearson's R, root mean square error, or some other calculation. Testing one set of parameters (β_1 through β_6 and *dat*) results in a predicted response time distribution that is summarized in 48 bins. These 48 predictions must be compared and aggregated with the corresponding empirical response distribution to produce one overall fitness measure. Wrapping this procedure into a function that returns the comparison result amounts to a "fitness function" which can be used by many of R's general-purpose optimization and fitting routines such as nlm() and optim(). Most optimization routines in R require a fitness function, so in many cases it is straightforward to swap out one fitting algorithm for another with no changes to the fitness function itself. Appendix D provides a more detailed explanation along with examples.

Regardless of the optimization algorithm chosen, the results will yield predictions for coefficients β_1 through β_6 and *dat*. As mentioned previously, the McCauley et al. (2009) biomathematical model provides fatigue estimates given time away and sleep history, so we are now equipped to make detailed performance predictions for any participant at any moment using Eq. (3) to parameterize our model. At this point we have achieved our goal of mapping the biomathematical model predictions and within-task fatigue predictions onto cognitive architecture parameters for any individual at any moment in time.

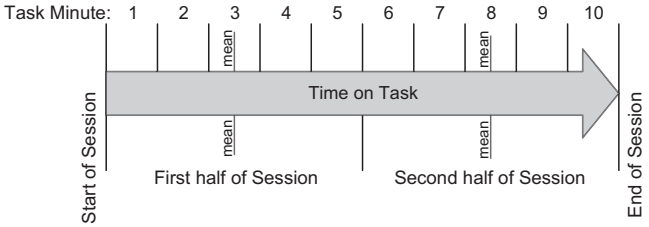

Fig. 6. Assuming a linear relationship between time on task and performance, then the mean of the first half-session distribution should equate to the mean of the minute 3 distribution, while the mean of the second half-session distribution should equate to the mean of the minute 8 distribution.

Although it is impractical to test experimentally, we can estimate the computational savings that Sif provides. Fitting a single participant using R's *genoud* function described in Appendix D required 7553 calls to the fitness function, with each call requiring 48 model predictions. Extrapolating to 26 participants and assuming conservative model run times of 1 min/sample, it would require approximately 17 years of computational time to produce fits such as those described in this paper with real model runs instead of Sif interpolations. With Sif, the computational time for 26 participants is about 8 h, which is over four orders of magnitude improvement.

4. Conclusion

Our research began with an effort to link dynamic fluctuations in alertness with a cognitive architecture (e.g., Gunzelmann et al., 2009). Although we have made excellent progress in this research, we have been constrained at times due to limitations in our methodology. Until now, we have not had tools with enough statistical power to produce a single unifying model of fatigue that spans several days, at an individual level, capable of making predictions at any point during the task. The increased statistical power of fitting all sessions simultaneously, as described in this paper, has allowed us to construct such a model. A critical step along the way was to develop the necessary tools.

The Sif software is an important development that provides a generic proxy service for any model with continuous parameters. The proxy provides good performance and based on our tests, predictions are very close to actual model runs. Of course this necessitates that the data supplied to Sif is at an appropriate level of granularity. If, for example, the mesh is too sparse, the proxy will not be able to convey higher frequency characteristics of model behavior. The modeler will need to have some basic understanding of the parameter space and its frequency characteristics in order to choose a suitable level of granularity. The modeler should also verify that the predicted model performance corresponds to the actual performance of the model for parameter values corresponding to best fits of the empirical data—we do not advocate using the interpolated data for purposes of making quantitative assessments of fit. A disparity can indicate that the grid used was too sparse.

Another important development was the R interface to Sif. By providing a direct interface between the model proxy and R, a large library of analytical tools is availed to the modeler. In this paper we showed how to leverage some of the optimization functions, but R has an incredibly rich amount of functionality and there are other libraries and functions useful for working with model data. For example, the plotting libraries can be used to produce publication-ready graphics to describe model behavior. As another example, some of the statistical tests could be used for model comparisons.

It could be argued that the creation of tools that enable more convenient and more complex model parameter fitting increases the likelihood of abuse through over-fitting. However, we would counter that the validity of this argument depends on the nature of the over-fitting. For example, an over-abundance of free parameters is perhaps the most common and serious mistake that leads to over-fitting, yet it has little to do with the technology that performs the fitting. The modeler must address this risk by seeking the most parsimonious theory possible, and some fitting algorithms and tool chains can actually help with that analysis (Gluck et al., 2010).

This paper used our model of the PVT as a case study to walk through the issues and resolution, but the methodology is applicable to fitting any model to time series data when the fluctuations in model parameter values that change over time can be guided mathematically. For example, drug influence is often characterized by metabolic half-life, which could be mapped to theoretically suitable model parameters with scaling coefficients determined by the methodology described in this paper. The HumMod simulation system (Hester et al., 2011) instantiates many such mathematical models that can be used to moderate cognitive parameters (Dancy et al., 2012).

Furthermore, the tools described in this paper can be applied more generally to any parameter fitting task that requires large numbers of model runs. For example, if a model is to be validated against many data sets (perhaps while exploring individual differences among hundreds of participants), using Sif to mine the model can be more computationally efficient than repeating individual searches.

This paper presents tools and a methodology to find relationships between mathematical and computational models. As interest in capturing the effects of cognitive moderators grows, modelers will be faced with similar challenges to the ones described in this paper. When the parameter space is unknown or does not allow for determining coefficients through a post hoc regression, a methodology such as the one described in this paper will be necessary to establish the relationship between physiological predictions and architectural parameters. All of the software discussed in this paper, including Cell, Sif, and R-Sif, is available upon request.

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, the U.S. Government, or L3 Corporation. This research was sponsored by grants 07HE01COR and 10RH04COR from the Air Force Office of Scientific Research. This research leveraged the Raptor HPC cluster at the U.S. Air Force Research Laboratory DoD Supercomputing Resource Center.

Appendix A. Partial listing of a grid data file

Wait	Hedge-up	Hedge-down	Motor-prep	Reaction-time
0	1	0.75	0.0145	375.0
0	1	0.75	0.01	475.0
0	1	0.75	0.01225	475.0
0	1	0.75625	0.03475	375.0
0	1	0.76875	0.01675	375.0
0	1	0.75625	0.019	375.0
...				

The first line of the data file can optionally contain column headers. Columns are separated by white space. In this example, the first 4 columns are model parameters, while the fifth column is the computed reaction time with those parameters.

Appendix B. Example cell/sif configuration file

```
# The following lines define 5 model parameters. The
  numbers indicate the start value,
# the end value, and the number of steps, respectively.
  Note that this is very similar to, but
# slightly different from the MindModeling submission
  page format.
IV = initial-wait 0 25 10 Constrained
IV = hedge-up 1 2.5 40 Constrained
IV = hedge-down .75 1 40 Constrained
IV = motor-feature-prep-time .01 .1 40 Constrained
IV = dat .025 .07 9 Constrained

# The following lines define what metrics are expected
  from the model.
DV = go-quant-25.0
DV = go-quant-75.0
DV = go-quant-125.0
DV = go-quant-175.0
DV = go-quant-225.0
DV = go-quant-275.0
DV = go-quant-325.0
DV = go-quant-375.0
DV = go-quant-425.0
DV = go-quant-475.0
DV = go-quant-525.0
DV = go-quant-575.0
DV = go-quant-625.0
DV = go-quant-675.0
DV = go-quant-725.0
DV = go-quant-775.0
DV = go-quant-825.0
DV = go-quant-875.0
DV = go-quant-925.0
DV = go-quant-975.0
```

(continued on next page)

```
# The following indicates
Tolerances = 0
# Indicates a full combinatorial mesh, such as shown in
  Fig. 2
Distribution = FullMeshSpace
# The directory location of the results file(s)
Logs=.
# All results files will have the following prefix
LogPrefix = results
```

Appendix C. Example HPC job submission file

```
# This single job file covers five nodes in a large
  parameter space. The file was
# automatically generated by a custom script designed
  for this particular model and
# corresponding parameter space. This is one of many
  job scripts created.

# The following lines inform the scheduler how long to
  run the job (3 h), what
# type of machine we would like (2-core SMP machines),
  and where to bill the time.

#!/bin/bash
#BSUB-W 03:00
#BSUB-o outfile
#BSUB-e errfile
#BSUB-J MakeSub-0–0
#BSUB-q standard
#BSUB-n 2
#BSUB-a SMP
#BSUB-P WPDOUSAF2772APAL

# The following runs the lisp-based cognitive model
  using steel bank common lisp (sbcl)
# with 5 different sets of parameter values (i.e. 5 nodes in
  the space).
# Six independent variable values are provided for each
  model run as command line
# parameters, starting with the 0.
~/sbcl/sbcl – core ~/sbcl/sbcl.core-quiet < mymodel.lisp
  0 0 1.000517246 2.0098935135 0.015
~/sbcl/sbcl – core ~/sbcl/sbcl.core-quiet < mymodel.lisp
  0 1 0.9866429976 1.9959898206 0.015
~/sbcl/sbcl – core ~/sbcl/sbcl.core-quiet < mymodel.lisp
  0 2 0.897053506 1.9062101985 0.015
~/sbcl/sbcl – core ~/sbcl/sbcl.core-quiet < mymodel.lisp
  0 3 0.852123408 1.861184748 0.015
~/sbcl/sbcl – core ~/sbcl/sbcl.core-quiet < mymodel.lisp
  1 0 1.000517246 2.0098935135 0.015

# Deleting this file is used as a check to make sure that
  the script completed successfully.
# If it does not disappear, an error occurred before it got
  this far, and it can just
# be resubmitted.

rm falcon-MakeSub-0-0.job
```

Appendix D. R integration details

Initialization is accomplished by sourcing RSif.R which will load the shared library and define the functions configure.sif() and query.sif(). The configure.sif() function defines the configuration file that will be used for all Sif queries, so that must be passed as a parameter. For example:

```
source("your-path/RSif.R")

configure.sif( "your-path/conf-pvtpm-fullmesh2.txt" )
```

At this point Sif can be queried. A Sif query requires two arguments:

(1) A list of architectural parameter values (i.e. independent variables), and
(2) The number of interpolated dependent measures that are expected back. For example, a single query into the parameter space may return 2 dependent variables such as reaction time and accuracy.

The function returns the results in an ordered list. For example:

```
results <- query.sif( c( ut, iu ), 39 )
```

In this example, we pass two architectural parameters into Sif (ut and iu), which would interpolate and return 39 dependent measures.[3] The number of parameters and the number of dependent measures returned should match what is defined in the Cell configuration file.

R provides many functions for general-purpose optimization and fitting problems. During the course of this research effort several were tested, but we chose the *genoud* package (GENetic Optimization Using Derivatives) (Mebane & Sekhon, 2011) because it reliably produced good results (with a relatively larger computational burden, it should be noted). However, most optimization routines in R use the same basic interface approach, so in most cases it would be straightforward to swap out one algorithm for another. Some examples of this will be provided later.

Genoud, like other optimization routines, requires the specification of a function that should be minimized or

[3] In the case of the PVT model, each of the 39 dependent measures represents a proportion of responses falling within a bin in a response time distribution.

maximized. It is through this function that the remaining model-specific logic is provided. The function will be called with a set of coefficients and is expected to return a measure of fitness. To produce this result, the function will compute the corresponding PVT model parameters at each half-session based on the supplied coefficients and compute a fitness metric with the empirical half-sessions. A simple root mean square deviation was used as the fitness metric, so the function essentially[4] becomes:

```
calculateFitMetricByHalfSession <- function( test_coefficients ) {

    # Build a frame of model data based on supplied coefficients

    modelData <- calculateModelDataByHalfSession( test_coefficients )


    # Compute and return RMSD

    sqrt(mean(( modelData - targetData ) ^ 2))

}
```

...where targetData is a frame containing an individual participant's empirical dependent measures for each half-session, and modelData is a similar frame for the model.

The calculateModelDataByHalfSession() is responsible for computing the *iu* and *ut* values for each half -session, querying Sif, and constructing a data structure for the results. This code assumes that a global variable called bmdata has been initialized to an array of predictions from the boimathematical model of alertness. The data for minute 3 is queried in the first loop, and minute 8 is appended in the second:

```
calculateModelDataByHalfSession <- function(params) {
 # The model data will be added as rows to this empty frame
 modelData <- c()

 # Calculate iu & ut for minute 3 of each session (a.k.a test bout)
 for( boutIndex in sort(unique(targetData $b))) {
     # Calculate iu <- A1 * B(t) + A2 * minute (always 3) + A3.
     iu <- params[ 1 ] * bmdata$V1[ 4 * boutIndex + 3 ] + params[ 2 ] * 2 +
 params[ 3 ]
     # Calculate ut <- A4 * B(t) + A5 * minute (always 3) + A6.
     ut <- params[ 4 ] * bmdata$V1[ 4 * boutIndex + 3 ] + params[ 5 ] * 2 +
 params[ 6 ]
     # dat is the 7th parameter, and can be used directly since it is
 unaffected by time
     dat <- params[ 7 ]

     # Query sif for dependent measures at these parameter values
     results <- query.sif( c( fpdec, ut, iu, dat ), 39 )
     # Add the row to the frame what will be returned
     modelData <- rbind( modelData, data.frame( b=boutIndex, m=2,
 fs=results[2], rt=results[ 1 ], l=results[ 38 ], nr=results[ 39 ]))
 }

 # Calculate iu & ut for minute 8 of each session (a.k.a test bout)
 for( boutIndex in sort(unique(targetData $b))) {
     # Calculate iu <- A1 * B(t) + A2 * minute (always 8) + A3.
     iu <- params[ 1 ] * bmdata$V1[ 4 * boutIndex + 3 ] + params[ 2 ] * 7 +
 params[ 3 ]
     # Calculate ut <- A4 * B(t) + A5 * minute (always 8) + A6.
     ut <- params[ 4 ] * bmdata$V1[ 4 * boutIndex + 3 ] + params[ 5 ] * 7 +
 params[ 6 ]
     # dat is the 7th parameter, and can be used directly since it is
 unaffected by time
     dat <- params[ 7 ]

     # Query sif for dependent measures at these parameter values
     results <- query.sif( c( fpdec, ut, iu, dat ), 39 )
     # Add the row to the frame what will be returned
     modelData <- rbind( modelData, data.frame( b=boutIndex, m=7,
 fs=results[2], rt=results[ 1 ], l=results[ 38 ], nr=results[ 39 ]))
 }

  return( modelData )
 }
```

Once the targetData frame is initialized, genoud can be called with the command:

[4] Colinearity among model parameters rightfully raises concerns about over-parameterization, however, it may be limited to only some sessions of the empirical data.

```
optima <- genoud( fn=calculateFitMetricByHalfSession, nvars=7 )$par
```

In this example, the fitness function is specified with the "fn" key while the number of parameters to optimize is specified with "nvars." A practical application of the genoud function may specify additional parameters to optimize or constrain the parameter search.

The calculateFitMetricByHalfSession() function can be used with a number of optimization routines in R, making algorithmic comparisons straightforward. For example, the Nelder-Mead simplex method (Nelder & Mead, 1965) might be desirable for its performance, and can be invoked via the optim function as follows:

```
bestParams <- optim( par=c(0.5, -0.005, 1.9, 0.023, 0.0013, 2.07, .05),
fn=calculateFitMetricByHalfSession)$par
```

As with the genoud function, the fitness function is specified with the "fn" keyword. Nelder–Mead requires a seed value—a starting point in the parameter space to begin its search—that is specified with the "par" argument. Another algorithm, simulated annealing, might be a reasonable compromise between performance and quality:

```
bestParams <- optim( par=c(0.5, -0.005, 1.9, 0.023, 0.0013, 2.07, .05),
fn=calculateFitMetricByHalfSession, method="SANN")$par
```

As with the genoud function, practical applications may need or benefit from specifying additional tuning parameters for the optim function.

References

Ackerman, P. L. (Ed.). (2010). *Cognitive fatigue: Multidisciplinary perspectives on current research and future application*. Washington, DC: American Psychological Association.

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

Belavkin, R. V. (2001). The role of emotion in problem solving. In *Proceedings of the AISB'01 symposium on emotion, cognition and affective computing* (pp. 49–57). Heslington, York, England.

Best, B. J., Gerhart, N., Furjanic, C., Fincham, J., Gluck, K. A., Gunzelmann, G., et al. (2009). Adaptive mesh refinement for efficient exploration of cognitive architectures and cognitive models. In *Proceedings of the ninth international conference on cognitive modeling (ICCM2009)* (Paper 252), Manchester, UK.

Dancy, C. L., Ritter, F. E., & Berry, K. (2012). Towards adding a physiological substrate to ACT-R. In *Proceedings of the 21st conference on behavior representation in modeling and simulation* (pp. 78–85). Amelia Island, FL.

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual RT task during sustained operations. *Behavior Research Methods, Instruments, & Computers, 17*(6), 652–655.

Gearhart, J. M., Robinson, P. J., & Jakubowski, E. M. (2009). Physiologically based pharmacokinetic modeling of chemical warfare agents. In R. C. Gupta (Ed.), *Handbook of the toxicology of chemical warfare agents*. London: Academic Press.

Gerlowski, L. E., & Jain, R. K. (1983). Physiologically based pharmacokinetic modeling: Principles and applications. *Journal of Pharmaceutical Sciences, 72*(10), 1103–1127.

Giambra, L. M., & Quilter, R. E. (1987). A two-term exponential functional description of the time course of sustained attention. *Human Factors, 29*, 635–643.

Gluck, K. A., Stanley, C. T., Moore, L. R., Jr., Reitter, D., & Halbrugge, M. (2010). Exploration for understanding in cognitive modeling. *Journal of Artificial General Intelligence, 2*(2), 88–107.

Gonzalez, C., Best, B., Healy, A. F., Kole, J. A., & Bourne, L. E. Jr., (2011). A cognitive modeling account of simultaneous learning and fatigue effects. *Cognitive Systems Research, 12*, 19–32.

Gunzelmann, G., Moore, L. R., Jr., Gluck, K. A., Van Dongen, H. P. A., & Dinges, D. F. (2009). Examining sources of individual variation in sustained attention (Paper No. 108). In *Proceedings of the 31st annual meeting of the cognitive science society* (pp. 608–613).

Gunzelmann, G., & Gluck, K. A. (2009). An integrative approach to understanding and predicting the consequences of fatigue on cognitive performance. *Cognitive Technology, 14*(1), 14–25.

Gunzelmann, G., Gluck, K. A., Moore, L. R., & Dinges, D. F. (2012). Impaired performance due to sleep deprivation: A role for diminished knowledge access. *Cognitive Systems Research, 13*(1), 1–11.

Gunzelmann, G., Gross, J. B., Gluck, K. A., & Dinges, D. F. (2009). Sleep deprivation and sustained attention performance: Integrating mathematical and cognitive modeling. *Cognitive Science, 33*(5), 880–910.

Gunzelmann, G., Moore, L. R., Jr., Gluck, K. A., Van Dongen, H. P. A., & Dinges, D. F. (2010). Fatigue in sustained attention: Generalizing mechanisms for time awake to time on task. In P. L. Ackerman (Ed.), *Cognitive fatigue: Multidisciplinary perspectives on current research and future application* (pp. 83–96). Washington, DC: American Psychological Association.

Gunzelmann, G., Moore, L. R., Salvucci, D. D., & Gluck, K. A. (2011). Sleep loss and driver performance: Quantitative predictions with zero free parameters. *Cognitive Systems Research, 12*(2), 154–163.

Harris, J. (2008). MindModeling@Home: A large-scale computational cognitive modeling infrastructure. In *Proceedings of the 6th annual conference on systems engineering research* <http://cser.lboro.ac.uk/CSER08/>.

Hester, R. L., Brown, A. J., Husband, L., Iliescu, R., Pruett, D., Summers, R., et al. (2011). HumMod: A modeling environment for the simulation of integrative human physiology. *Frontiers in Physiology, 2*(12).

Kase, S. E. (2008). *Parallel genetic algorithm optimization of a cognitive model: Investigating group and individual performance on a math stressor task*. Doctorial dissertation, Penn State University. <http://etda.libraries.psu.edu/theses/approved/WorldWideIndex/ETD-3236/index.html>.

Kase, S., & Ritter, F. E. (2009). An HPC and PGA approach to model calibration and validation. In *Proceedings of the 18th conference on behavior representation in modeling and simulation (BRIMS)* (Paper 10, pp. 39–46), Sundance, Utah.

Kase, S. E., Ritter, F. E., Schoelles, M. (2009). Caffeine's effect on appraisal and mental arithmetic performance: A cognitive modeling approach tells us more. In *Proceedings of the 9th international conference of cognitive modeling* (Paper 252), Manchester, United Kingdom.

Marinier, R., Laird, J. E., & Lewis, R. L. (2009). A computational unification of cognitive behavior and emotion. *Journal of Cognitive Systems Research, 10*, 48–69.

McCauley, P., Kalechev, L. V., Smith, A. D., Belenky, G., Dinges, D. F., & Van Dongen, H. P. A. (2009). A new mathematical model for the homeostatic effects of sleep loss on neurobehavioral performance. *Journal of Theoretical Biology, 256*, 227–239.

Mebane, W. R., & Sekhon, J. S. Jr., (2011). Genetic optimization using derivatives: The rgenoud package for R. *Journal of Statistical Software, 42*(11), 1–26.

Meyer, D. E., Glass, J. M., Mueller, S. T., Seymour, T. L., & Kieras, D. E. (2001). Executive-process interactive control: A unified computational theory for answering twenty questions (and more) about cognitive ageing. *European Journal of Cognitive Psychology, 13*, 123–164.

Moore, L. R. Jr., (2011). Cognitive model exploration and optimization: A new challenge for computational science. *Computational and Mathematical Organization Theory, 17*, 296–313.

Nelder, J. A., & Mead, R. (1965). A simplex algorithm for function minimization. *Computer Journal, 7*, 308–313.

Neri, D. F. (Ed.) (2004). Fatigue and performance modeling workshop, June 13–14, 2002 [Special Issue]. *Aviation, space, and environmental medicine, 75*(3, Supplement).

R Development Core Team (2011). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0. <http://www.R-project.org/>.

Raymond, W. D., Fornberg, B., Buck-Gengler, C. J., Healy, A. F., & Bourne, L. E. (2008). Matlab optimization of an IMPRINT model of human behavior. In *Proceedings of the seventeenth conference on behavior representation in modeling and simulation (BRIMS)* (pp. 26–33). Providence, Rhode Island: Simulation Interoperability Standards Organization.

Reder, L. M., Oates, J. M., Dickison, D., Anderson, J. R., Gyulai, F., Quinlan, J. J., et al. (2007). Retrograde faciliation under midazolam: The role of general and specific interference. *Psychonomic Bulletin & Review, 14*(2), 261–269.

Ritter, F. E., Reifers, A. L., Klein, A. C., & Schoelles, M. J. (2006). Lessons from defining theories of stress. In W. Gray (Ed.), *Integrated models of cognitive systems*. New York, NY: Oxford University Press.

Warm, J. S., Parasuraman, R., & Matthews, G. (2008). Vigilance requires hard mental work and is stressful. *Human Factors, 50*, 433–441.

Wong, T. J., Cokely, E. T., & Schooler, L. J. (2010). An online database of ACT-R parameters: Towards a transparent community-based approach to model development. In D. D. Salvucci & G. Gunzelmann (Eds.), *Proceedings of the 10th international conference on cognitive modeling* (pp. 282–286). Philadelphia, PA: Drexel University.