Cognitive Model Exploration and Optimization:

A New Challenge for Computational Science

L. Richard Moore Jr.

Lockheed Martin Systems Management
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 South Kent Street
Mesa, Arizona 85212-6061
Larry.Moore@mesa.afmc.af.mil

Keywords:
adaptive mesh, exploration, searching, parameter space, predictive analytics,
volunteer computing, high performance computing

ABSTRACT: *Parameter space exploration is a common problem tackled on large-scale computational resources. The most common technique, a full combinatorial mesh, is robust but scales poorly to the computational demands of complex models with higher dimensional spaces such as those found in the cognitive and behavioral modeling community. To curtail the computational requirements, I have implemented two parallelized intelligent search and exploration algorithms, both of which are discussed and compared in this paper.*

1. Introduction

Research in cognitive science often involves the generation and analysis of computational cognitive models to explain various aspects of cognition. Typically the behavior of these models various across a continuous parameter space composed of a number of theoretically motivated parameters, but most commonly we are left to our own devices to find the right balance of parsimony and fit within that space.

We are certainly not alone. The modeling community more generally is already well aware of the challenges associated with parameter optimization. Furthermore, there appears to be a growing appreciation of the parameter space itself—a qualitative understanding of the space can provide valuable insights regarding a model's behavior, optimal parameter ranges, the number of optima, and the distance(s) from canonical values. It is this deep understanding of the model's parameter space that allows us to find a balance between parsimony, optimization and generality (Gluck, Stanley, Moore, Reitter & Halbrügge, 2010). However, this is difficult to achieve on the computational scale of a workstation, so we have turned to high performance computing (HPC) clusters and volunteer computing for large-scale computational resources.

The majority of applications on the Department of Defense HPC clusters focus on solving partial differential equations (Post, 2009). These tend to be lean, fast models with little noise. While we lack specific data regarding typical job sizes and durations, HPC maintenance is regularly scheduled at two-week intervals, so it seems reasonable to assume that most jobs fit within this window.

In contrast to HPC applications, volunteer computing projects tend to involve singularly specific, highly parallelizable tasks crunching vast quantities of data over time spans measured in months and years, such as SETI@home's analysis of interstellar radio signals and Folding@home's studies of protein folding. Both of these examples run on a common software framework called the Berkeley Open Infrastructure for Network Computing (BOINC), which enables volunteers to donate idle time from their computational resources to projects of their choice. The volunteer computing application developed by my colleagues is called MindModeling@Home, and it too runs on the BOINC infrastructure (Harris, Gluck, Mielke & Moore, 2009). Projects that work well with

BOINC tend to be long lasting and can tolerate latencies measured in days, which happen quite commonly when volunteer resources are interrupted or retasked.

Cognitive models fit somewhere between these two extremes. Our models are computationally expensive and produce stochastic results, quite unlike the partial differential equations typically solved on HPC clusters. And unlike most of the BOINC projects, we strive to analyze many different models with vastly differing performance characteristics within a calendar year. Our unique requirements present new methodological challenges for both HPC and volunteer resources. This paper describes some of the methodologies we have explored, the trade space among them, and my latest research efforts to apply HPC and volunteer resources to characterize and search parameter spaces.

2. Meshing

In its simplest form, "meshing" involves the construction of an n-dimensional grid by iterating through each parameter range by a fixed interval, and capturing the combinatorics to be used as the basis of model runs. The resulting simple orthogonal grid seems to suffice for most of our cognitive models.

Once the mesh is defined, portions can be distributed amongst computational nodes and executed completely independently. Meshing has been widely used for many years (Chen & Taylor, 1998) and it lends itself well to both HPC and volunteer resources. The complete independence among computational nodes affords the ultimate in "embarrassingly parallel"—a term commonly used to describe computational tasks that can be efficiently executed with little or no serial operations. Parallelizability is the key to realizing the full potential of large-scale computational resources.

Full combinatorial meshes have other advantages, as well. For example, there is little software overhead in computing these meshes (at least for our relatively simple requirements) and the corresponding job files for the HPC schedulers. For volunteer resources, my colleagues have developed a web interface specifically for this purpose with plans to make it available as a community resource (Harris et al, 2009).

Combinatorial meshes are also flexible. No assumptions are made about the structure or even the continuity of the parameter space. The data can be stored in any format convenient for the modeler to analyze. Analysis is straightforward, and the results can be visualized or mined indefinitely, within the limits of precision defined by the original mesh.

Another point to consider about full combinatorial meshes is that counting the results files quickly reveals the success of the jobs; one result should be present for every parameter combination. While we might shrug off a failure on our desktop as a 1 in a million fluke, when running models millions of times this seemingly innocuous failure rate becomes noticeable, and quick methods to detect and recover are desirable—in this case the modeler can simply rerun the specific mesh nodes that failed to produce results files.

How do full combinatorial meshes fare with cognitive models? In one research effort, we have developed a model that performs a Digit Symbol Substitution Task (DSST) (Moore, Gunzelmann & Gluck, 2008). This is a simple task where the model is presented with 9 digit / symbol pairs, and when prompted with a symbol the model responds with the appropriate digit. This fairly typical cognitive model has 7 relevant quantitative parameters and due to stochasticity must be resampled at least 10 times to establish a reliable measure of central tendency. With an average run time of 2 minutes, a mesh with 10 increments per variable would require 271 days to compute if run continuously on 512 cores. A computational challenge of this magnitude would overwhelm any computational resource for quite some time, and as mentioned previously there is some desire to analyze more than one model per calendar year.

There are primarily two issues that drive the computational demands of the DSST. First, the 7 parameters exhibit the "curse of dimensionality"—a phrase used to describe the exponential requirements of additional parameters in a space (Bellman, 1961). After examining the parameter space and understanding the interrelationships, dimensionality can often be reduced, but not until after an initial analysis is completed.

The second primary issue contributing to the computational requirements is the 2-minute run time required for each node in the parameter space. The DSST is a learning model—its behavior changes across sessions as it gains knowledge and experience. Therefore, to properly compare learning characteristics with human subjects, the entire learning curve must be constructed at each parameter combination across all sessions. Considering that, in this case, the model is performing the task across 32 sessions (96 simulative minutes), 2 minute run times seem quite reasonable.

Recognizing that large-scale computational resources can only take us so far, we have turned our attention to intelligent exploration and search strategies that run on both HPC and volunteer resources. Our interests are specifically focused on approaches that allow searching a

parameter space for optimal values, as well as characterizing the overall space in general.

3. Adaptive Mesh Refinement

Adaptive mesh refinement (AMR) is an intelligent search strategy that dynamically divides the overall search space into subcubes of varying size, each of which is capable of making predictions about measures in its local area of space to a predefined degree of accuracy (Berger & Oliger, 1984).

My parallelized implementation of AMR is called Quick, and it consists of about 11,000 lines of C++ code. The code has been ported to several HPC clusters, as well as our BOINC-based MindModeling volunteer computing system.

Implementing AMR—or any intelligent algorithm, for that matter—on large-scale computational resources requires a serious engineering investment. The software needs to be robust enough to recover from faults throughout the system—including models under evaluation-- and it needs to be reliable enough to run for hundreds or thousands of hours without memory leaks, crashing, etc.

To initiate an AMR using Quick, the modeler begins by defining the independent variables, their ranges, and the increment for each. The increment is identical to the increment used when constructing a full combinatorial mesh—although hypercubes produced by an AMR may span large portions of space, their boundaries are always constrained to the implicit grid lines defined by the increment. The hypercubes never overlap, and the sum of their volumes equals that of the parameter space overall.

The user also specifies the dependent measures that the model will produce, as well as a threshold value for each. The threshold is an important consideration, because ultimately it will constrain how accurate the results will be.

Once configured, the procedure to execute Quick varies between HPC and MindModeling. Running software on HPC resources is accomplished through "job" submissions. A job is defined through a simple shell script that describes the requested computational resources and the software to run. Jobs are submitted to a dedicated scheduler that executes the software when the requested resources become available. Quick begins with a single job that requests a single computer. As the AMR progresses, Quick will automatically schedule more jobs to run in parallel as aggressively as possible.

On MindModeling things behave quite differently. In this case, Quick is automatically executed on the servers at periodic intervals to determine which points in the parameter space need to be computed for the AMR. As volunteers request work, they are provided with these points to compute, and as they return results and the AMR progresses new points will be generated by Quick. Thus, parallelization is achieved at the level of sample acquisition.

Regardless of the computational context, the AMR methodology is the same. Quick begins by treating the entire parameter space as a single large single hypercube. The process begins by executing the model with parameter values at each of the corner points. AMR assumes that measurements are accurate, so we typically resample the model a fixed number of times and collapse across the dependent measures to remove stochasticity. In any n dimensional space, there will be 2^n corners to sample.

In addition to the corner points, the center of the cube is measured as well. (As with all nodes considered in the space, the center is constrained to the specified grid, so it may not reflect the precise mathematical center.) In addition to measuring the center, Quick will also make a mathematical prediction of the center, assuming that the model's behavior changes smoothly across the parameter space, yet accounting for twisting that can occur. If the difference between the measured value and the predicted value is within the specified threshold for each dependent measure, then the hypercube is considered smooth and predictable, and the process is complete. However, if any of the dependent measures exceed the threshold, the hypercube is divided into 2^n subcubes about the center point, and each subcube is analyzed using the same process just described.

When hypercubes split into subcubes, each subcube can be treated as a parameter space in its own right, albeit smaller than the true overall space. This is the key to parallelizing AMR on HPC resources, as the analysis of each subcube can be scheduled as an independent HPC job. Aside from the shape of the parameter space, these new jobs are identical to the original that started the analysis.

AMR can result in substantial computational savings, yet the quantitative quality of the results typically remains high (Best et al, 2009). The quality of the results is consistent across the space, too, so unmeasured points can be interpolated and the resulting grid can be mined just as a full combinatorial mesh. Further, because the space is mathematically defined, off-grid interpolation can also be calculated if desired. There is also something to be said

for the reduction in data that needs to be transferred to the workstation for analysis.

Nevertheless, AMR does have its drawbacks. First, the computational savings with AMR are unpredictable. This is also consistent with the Best et al (2009) work, which showed that AMR efficiency was heavily influence by threshold and implementation factors that can be difficult to predict a-priori. Furthermore, the structure of the space, (which in turn depends on the parameters and their relationships) and the number of dependent measures can also heavily influence AMR efficiency. In my experience with our models, it is not uncommon for an AMR to compute nearly all the nodes in the space, resulting in little savings.

Recall that AMR must evaluate the corners and center of each hypercube before it can move forward. In a volunteer environment such as MindModeling, this can be problematic because of the large latencies. At any moment, a volunteer might turn off their computer, or use it for something else, and processing is stalled until an explicit timeout is reached, which is usually at least a day. So while volunteer networks provide huge computational power, they are a poor match for the methodological requirements of AMR.

AMR on HPC suffers for different reasons, but with similar effects. In this case, parallelization is not usually the problem, but each parallel analysis requires a new HPC job to be scheduled. HPC schedulers vary in reliability and performance—which is in itself problematic for AMR—but they all share a first-in-first-out paradigm, so new jobs must wait for resources to be made available from jobs scheduled prior. A simple 3-dimensional parameter space with 8 divisions per parameter could potentially result in millions of job submissions, each with its own wait time in the job queue.

To test how many submissions are actually made, and their impact on the overall wall clock time, I ran six adaptive meshes on the Jaws high performance computing cluster in Maui using a model of the Psychomotor Vigilance Task (PVT). The PVT is a simple model that simulates a button press when a visual stimulus is presented at random time intervals (Gunzelmann, Gross, Gluck & Dinges, 2009). Two variants of this model were tested, and each was run using three different values for the threshold that controls the likelihood of searching deeper into the parameter space. All six meshes explored the same three-parameter space.

The mean number of HPC jobs submitted was 577. The average run time for each job was 2 minutes, and the average wait time in the scheduler queue was 5.9 minutes. One must be cautious when interpreting these results due to the small sample size and large variation in HPC usage, but in this case the mean wait time was nearly 3x longer than the mean run time per job.

Although AMR is more computationally efficient than a full combinatorial mesh on large-scale resources, it can be slower in terms of wall clock time. If you recall, our original motivation for combining intelligent search and exploration with large-scale computational resources was to improve analytical capacity with cognitive models, yet AMR does not consistently deliver.

Despite its shortcomings, AMR has clearly demonstrated that combining intelligent search with HPC and volunteer resources is indeed possible. My most recent research re-imagines optimized search specifically for the context of cognitive models on parallel computational resources.

4. Regression Trees

Recognizing that parallelization is the key to fully leveraging HPC and volunteer resources, I have developed a flexible stochastic search methodology that allows massive parallelization with virtually no interdependencies. Furthermore, recognizing the necessity for qualitatively understanding the parameter space, I have also developed accompanying visualization software that operates in real time as the space is constructed. The visualization software is called Hurricane, while the intelligent search software is called Cell.

Hurricane and Cell are written in Objective C, and at 5300 lines combined they are about half the size of Quick, testifying to their relative simplicity. They were developed on Mac OS X, and Cell specifically has been ported to Linux to support HPC and MindModeling integration. At this time Cell has been successfully ported and tested on four different HPC clusters, with MindModeling integration underway.

As was the case with Quick, Cell and Hurricane begin with a user-specified configuration including independent variables, their ranges and increment, and the dependent measures. In contrast to the AMR configuration for Quick, no threshold is required.

Like all software run on the HPC, Cell is executed through a job submission. However, because Cell is immediately parallelizable any number of job submissions can be made during startup. Typically I limit myself to 128 jobs, mostly to avoid complaints from other HPC users.

On MindModeling, a single instance of Cell runs on the server for the duration of a model run. This “listener” process analyzes incoming data, and upon request,

generates lists of points that are distributed to volunteer resources as they request work. Like Quick, Cell achieves parallelization on MindModeling by distributing model runs to volunteer resources.

Cell can analyze the parameter space in either of two ways: exploration or searching. Both approaches divide the space into a set of hypercubes that are geometrically analogous to AMR. However, rather than sampling just corners and the center, Cell samples stochastically within the hypercube space and calculates the best fitting hyperplane for each dependent measure—an analytical approach sometimes referred to as a regression tree (Alexander & Grimshaw, 1996).

Regardless of whether Cell is searching or exploring, it tries to maintain a consistent sample density among the hypercubes, regardless of size. This means that areas of the space with higher sampling will have greater numbers of hypercube divisions. The minimum number of samples targeted for each hypercube is based on the work of Knofcyznski and Mundfrom (2008), which suggests a linear relationship between the number of samples required to make a good regression prediction and the dimensionality of the space. It is not until a hypercube contains 2x this amount does it split along its longest dimension. Within the confines of a single hypercube sampling is uniform, so the split should roughly divide the samples equally between both subcubes.

The key distinction between Cell's two analytical approaches lies in the way they construct their sampling distribution. The exploration approach performs a characterization of the space—in this case the sampling distribution is positively correlated with the residual variation in each hypercube. Unexplained variation is presumably the result of noise or a poor regression fit, and in either case it is prudent to sample more, and potentially to subdivide more, to resolve the ambiguity. In this mode, the exploration process has no definitive end and runs as long as the modeler desires.

In truth, I rarely use exploration mode because our work typically involves parameter optimization as well as characterization, and search mode provides both. In this case, the user supplies additional configuration information consisting of dependent measure "target goals" to search for. In terms of cognitive modeling, this typically takes the form of human data. When supplied, the sampling distribution is skewed towards hypercubes with the lowest deviation from the human data (or whatever target goals are supplied), and so the space winds up being more intricately constructed in those areas. The search is considered complete when the best fitting hypercube cannot divide any more based on the constraining grid.

With data in hand (or even while it is being obtained in the case of running on local resources), Hurricane can be used to visualize the results, as is shown in Figure 1. Hurricane conducts the same analysis that Cell does, and produces the regression tree in the form of a 3D graph. Any two independent measures can be selected for the x and z-axis, and any dependent measure can be selected for the y-axis (vertical). The remaining independent measures can be manipulated in real time via sliders, which provides a convenient mechanism to grasp an otherwise esoteric hyperdimensional space. Hurricane can also scan the space for optimal parameter values or make predictions, which can then be imported into more generalized analytical tools like R or SPSS.

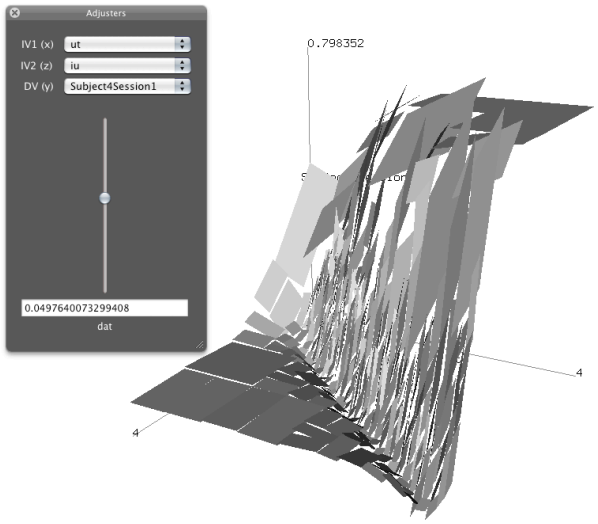

Figure 1. Hurricane visualization of a PVT parameter space. The vertical axis represents RMSD between human measures and the model, while the other two axes represent independent variables. A third independent variable can be manipulated with the slider. Best fitting parameter values are located within the trench area, which received more samples and therefore is more finely subdivided.

Searches conducted with Cell provide large computational advantages over AMR and full combinatorial meshes. This is primarily because vast sections of the space—those areas that are distant from target areas of interest—are only lightly sampled and mostly ignored once deemed suboptimal. As an example, I ran the PVT model through a full combinatorial mesh, an AMR with Quick, and a regression tree analysis with Cell. Identical grid slicing was used for all three, and they were all run on the same Mana HPC cluster in Maui.

Figure 2 shows the number of model runs required to complete an analysis of the parameter space for each methodology. In this example, the AMR—although it was configured with a liberal 5% threshold—wound up sampling most of the space anyway, while the Cell required two orders of magnitude fewer model runs.

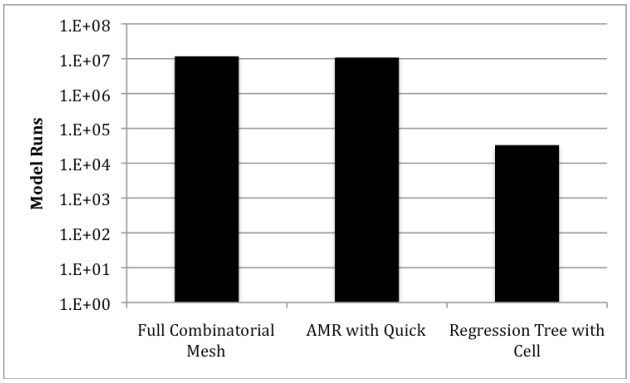

Figure 2. Comparison of computational requirements for each of the three methodologies discussed.

The amount of time required to complete the analyses is shown in Figure 3. Note that the AMR took 4.2 times longer than the full combinatorial mesh, which is almost exactly what would be expected if queue wait times were 3x the run time as discussed above. Because Cell parallelizes immediately upon startup and does not auto-schedule new jobs like Quick, most of the scheduling queue delays are avoided. For more complex searches that fail to complete within the scheduled amount time, I can simply reschedule more jobs, and each Cell instance will read the samples acquired previously from disk, and pick up where the older Cell instances left off.

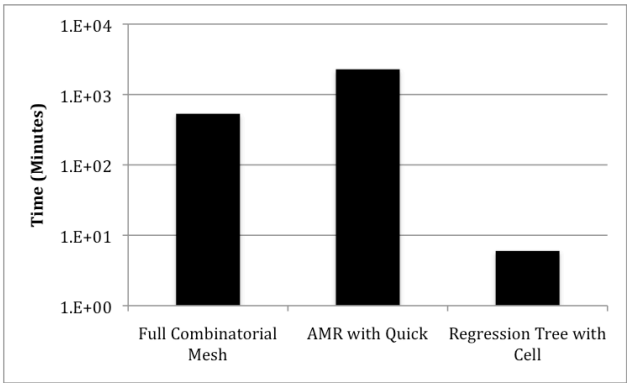

Figure 3. Comparison of wall clock time required to analyze the PVT parameter space using the three methodologies.

Speed and efficiency are important, but they are only useful if the resulting analysis is viable. Figure 4 compares the optimized parameter predictions from each of the three methodologies. To produce this table, I reran the model at the predicted optimal parameter values and computed an RMSD against the human data for each methodology. The model was run 100x to reduce noise—the same amount used during the AMR and full combinatorial runs. As expected, the full combinatorial mesh produced the best results. It was surprising to see that the regression tree methodology edged out AMR, but this is likely caused by variation in the model's performance.

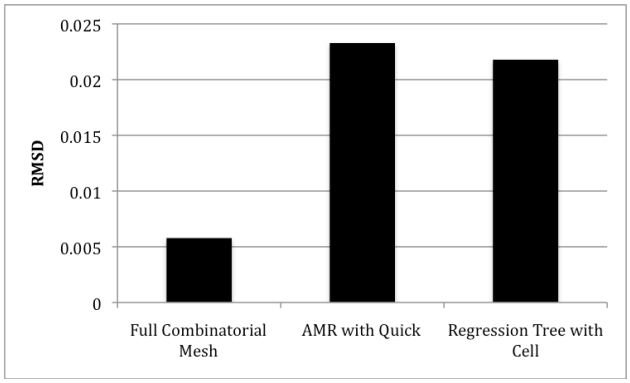

Figure 4. RMSD between best fitting parameter predictions and human data.

Many of the issues challenging parallelized AMR disappear in the context of regression tree exploration and searching. This is because Cell does not base decisions upon the outcome of specific, accurate, grid-constrained samples. Rather, the decisions are based on statistical analysis of a set of distributed, stochastic samples. As a result, any number of Cell instances can be started at once and run in parallel, each making its own decision about how to divide the space and where to sample.

Although the integration remains a work in progress, I expect that Cell will work well with volunteer resources. In this case AMR was stalled waiting for specific points to complete, but Cell, with its semi-random sampling strategy, can always generate work for volunteers. However, we will need to be careful to limit the number of outstanding points being computed at any given time. The end result of too many outstanding samples could be hundreds or thousands more samples in a hypercube than is really necessary to make a search decision. The extra data would still be useful for visualization purposes, but it would reduce the efficiency of searching.

In my ongoing efforts to combine intelligent search and exploration with large-scale computational resources, Hurricane and Cell represent best results to date. Nevertheless, they present their own new challenges. For example, the confidence of predictions based on discontinuous regression planes is inconsistent, and highly dependent upon the distance from the center of the hypercube. Predictions across the boundary of two

discontinuous hyperplanes can be disturbingly disparate compared to neighboring predictions. This not only makes visualization less appealing, but data mining outside of specified search goals can be problematic.

From an implementation perspective, Cell is more computationally intensive than AMR and full combinatorial meshes. Every incoming sample requires a search to determine its encompassing hypercube, and the introduction of new data into the hypercube will require the calculation of new regressions. To maintain pace with the incoming data stream, results must be stored in RAM rather than disk-based storage, which limits scalability. The number of samples that can be maintained in a fixed amount of RAM depends upon the amount of memory required to store a sample, which includes values for the independent measures, dependent measures, and search targets specified.

Even with in-memory data management, however, the number of regressions required can still be computationally challenging. For example, the DSST model mentioned earlier captures 9 measures across 32 sessions, amounting to 218 total independent measures, each maintaining its own regression tree. Hurricane requires about 5 hours to read in the data from this model and reconstruct the regression trees for visualization, which seems excessive, to say the least.

Despite these limitations, the regression trees seem to be another step in the right direction. Using Cell, our cognitive models scale well on HPC resources from both computational and wall clock time perspectives. Some of our faster cognitive models, in fact, can now be analyzed in a few hours on local resources, which avails large-scale computational resources for even more complex models. Additionally, Hurricane's multidimensional visualization capability has become an indispensible part of my normal workflow.

4. Discussion

In a broad sense, the engineering problem being addressed is one of computational performance and efficiency. Large-scale computational resources take us part of the way, and the remaining effort is incumbent upon us, as the resource users.

In the world of software engineering, there is a basic rule to optimization: focus on the innermost loop. In the context of this discussion, we have a parameter exploration / search methodology exercising a cognitive model, and it is the model itself that constitutes the bulk of processing in the innermost loop.

The model and its implementation are the embodiment of a theory, however, and this can severely constrain optimization options. This is certainly the case for my colleagues and I, where our models are based on a publicly available cognitive architecture (ACT-R; Anderson, 2007) that is shared among a relatively large scientific community. In our case, we routinely share models to combine and test different cognitive moderators, and it is important to maintain a consistent architecture across the community.

Therefore, we optimize our inner loop not by changing code, but by reducing the number of model runs as much as possible. AMR does this well and is used successfully in some contexts, but it appears, however, that the full utility of AMR does not necessarily transfer across domains and contexts. As cognitive and behavioral modelers begin to leverage large-scale computational resources, we must also develop suitable parallel search and exploration algorithms for our models.

This paper described our recent efforts using regression tree predictions to drive sampling distributions, and ultimately hypercube division. Like AMR, the technique reduces computational demands through a reduction in model runs, but the nature of the approach seems to be more agreeable to parallelization.

Regression trees, however, are not the only option. The dynamics of Cell are driven by two governing principles: 1) Sample more in areas of interest and 2) subdivide more in areas of higher density. The regression trees are used to determine the areas of interest, but other predictive analytical techniques can be substituted without compromising the fundamental approach. Multivariate adaptive regression splines (MARS) are one interesting possibility (Friedman, 1991).

However, models like the DSST have demonstrated that the computational demands of the analytical technique are becoming a serious consideration. While I predict that MARS will be more efficient than regression trees in terms of reducing the required number of model runs, I also expect that the analytical processing requirements will be significantly more demanding. It seems a trade space is becoming apparent between the computational demands of the model versus the computational demands of the search / exploration algorithm. For us this is not necessarily a bad trade space, because it is much less problematic to optimize a methodology as opposed to a theory, and there remain many opportunities to do so.

5. 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 Lockheed Martin Corporation. This research was sponsored by grants 07HE01COR and 10RH04COR from the Air Force Office of Scientific Research.

I would like to thank Kevin Gluck and Glenn Gunzelmann for reviews of earlier drafts of this paper, as well as the Performance and Learning Models team and Adaptive Cognitive Systems for their influences and indulgence in supporting this work.

6. References

Alexander, W. P., & 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.

Bellman, R. E. (1961). *Adaptive Control Processes*. Princeton: Princeton University Press.

Berger, M., & Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics,* 53, 484-512.

Best, B. J., Gerhart, N., Furjanic, C., Fincham, J., Gluck, K. A., Gunzelmann, G., & Krusmark, M. (2009). Adaptive mesh refinement for efficient exploration of cognitive architectures and cognitive models. In *Proceedings of the Ninth International Conference on Cognitive Modeling*, Manchester, UK.

Chen J., Taylor V. (1998). Mesh Partitioning for Distributed Systems. In *Seventh IEEE International Symposium on High Performance Distributed Computing*, 292-300.

Friedman, J. (1991). Multivariate adaptive regression splines. *The Annals of Statistics, 19,* 1-141.

Gluck, K. A., Stanley, C. T., Moore, L. R., Reitter, D., Halbrügge, M. (2010). Exploration for Understanding in Model Comparisons. Under review, *Journal of Artificial General Intelligence.*

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.

Harris, J., Gluck, K. A., Mielke, T., & Moore, L. R. (2009). MindModeling@Home … and Anywhere Else You Have Idle Processors [Abstract]. In A. Howes, D. Peebles, & R. Cooper (Eds.) *Proceedings of the Ninth International Conference on Cognitive Modeling. Manchester*, United Kingdom: University of Manchester.

Knofcyznski, G. T., & Mundfrom, D. (2008). Sample sizes when using multiple linear regression for prediction. *Educational and Psychological Measurement. 68,* 431-442.

Moore, L. R., Gunzelmann, G., & Gluck, K. A. (2008). Evaluating mechanisms of fatigue using a digit symbol substitution task [Abstract]. In N. Taatgen, H. van Rijn, L. Schomaker, & J. Nerbonne (Eds.), *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society*, Austin, TX: Cognitive Science Society.

Post, D. (2009). The Promise and Challenges for Next Generation of Computers [PowerPoint slides]. Retrieved from http://www.cc.gatech.edu/~bader/AFRL-GT-Workshop2009/AFRL-GT-Post.pdf

Author Biography

L RICHARD MOORE JR is a Research Engineer with Lockheed Martin Systems Management at the Air Force Research Laboratory, Warfighter Readiness Research Division in Mesa AZ. He completed his B.S.E. in Electrical Engineering in 1992, with an M.S. in Applied Psychology in 2008, both from Arizona State University.