Best, B., Fincham, J., Gluck, K., Gunzelmann, G., & Krusmark, M. (2008). Efficient use of large-scale computational resources. In J. Hansberger (Ed.), *Proceedings of the Seventeenth Conference on Behavior Representation in Modeling and Simulation* (pp. 180-181). Orlando, FL: Simulation Interoperability Standards Organization.

Efficient Use of Large-Scale Computational Resources

Brad Best
Adaptive Cognitive Systems
1942 Broadway St., Suite 305
Boulder, CO 80302
(303) 413-3472

Kevin Gluck, Glenn Gunzelmann
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 S. Kent St.
Mesa, AZ 85212
480-988-6561 x-677

Jon Fincham
Carnegie Mellon University
Department of Psychology
5000 Forbes Ave.
Pittsburgh, PA 15213
(412) 268-3498

Michael Krusmark
L-3 Communications at
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 S. Kent St.
Mesa, AZ 85212
480-988-9773 x-679

1. Introduction

We have recently seen some of the first forays into the use of high-performance computing (HPC) resources among scientists in the cognitive modeling community (e.g., Gluck, Scheutz, Gunzelmann, Harris, & Kershner, 2007). HPC resources have been used to give cognitive modelers a tool to explore parameter spaces so they can better understand cognitive models and architectures. Much of this research has focused on using HPC resources to perform a sweep of the parameter space associated with particular task models to identify the models and associated parameter values that maximize fit to the corresponding human behavioral data. One of the lessons learned in the early stages of our use of HPC resources has been the realization that, regardless of the number of available processors, there will always be interesting research questions that will stretch or exceed the capacities of any HPC resource (i.e., available resources will always limit the number and complexity of questions that can be asked). The combinatorics of many cognitive model parameter spaces are simply prohibitive. Thus, there is a significant challenge in managing the combinatorial explosion that results from considering these larger cognitive modeling parameter spaces. The goal of the current work is to explore methods for improving the efficiency of HPC cognitive modeling work through the application of Adaptive Mesh Refinement (AMR) techniques.

2. Adaptive Mesh Refinement (AMR)

Adaptive Mesh Refinement is a general technique used to narrow the search of a broad parameter space to those areas that are "interesting" (Berger & Oliger, 1984; Rai & Anderson, 1981). This smart exploration of parameter spaces is accomplished through decomposing the experimental space hierarchically, and focusing computational resources on areas that are more informationally rich. Visualization also takes advantage of this decomposition, and the techniques often go hand-in-hand under the rubric of "adaptive mesh refinement", or AMR. AMR is characterized by a breadth first decomposition of the parameter space, leading to finer and finer refinements in areas of the space that have some measurable local curvature. Flat (linear) space is often uninteresting and sampling it with higher frequency does not provide additional information about the dynamics of the system.

2.1 AMR Applied to Cognitive Modeling

Developing code for smart search of the parameter space is complicated in the cognitive modeling domain by the stochasticity of results (i.e., different model runs may produce different results at the same parameter values), and by the variance inherent in the underlying phenomena to be modeled (human behavior). After exploring several alternatives, we have implemented a version of AMR that accounts for both stochasticity and curvature.

The essential method is multi-dimensional and *architecture agnostic*, but it is best explained starting with a single dimension for a particular architecture. Given an ACT-R parameter space and two values of that parameter for some dependent measure (e.g., reaction time at G=10 and G=20), the question that needs to be answered is whether it is profitable to also sample the parameter space at an intermediate value (G=15). Given a number of samples at G=10 and G=20, it is possible to conduct a power analysis to determine how many samples would be necessary in order to detect a difference at G=15. The null hypothesis we have implemented is that the surface between the two points is linear, and thus we expect the value at G=15 to have no statistically detectable difference from the mean of the values sampled at G=10 and G=20. This can be used to both drive the number of model runs at the point (through power analysis), and to determine when the refinement is no longer necessary (when the output for model runs using intermediate parameter values are consistent with the linear interpolations of the extreme values).

2.2 Distributing ACT-R runs

The ACT-R-AMR system is a client-server software package capable of running multiple ACT-R 6.0 models simultaneously on different CPUs for the purpose of rapidly searching a parameter space. The system combines AMR-based exploration of a parameter space with visualization of that space using the VisIt visualization software system developed by Lawrence Livermore National Laboratory (LLNL).

The distribution of ACT-R runs is accomplished through a client-server architecture. The client is a Lisp program that connects to the server, requests a unique socket, and then communicates with the server over that socket. The client socket code is based on the ACT-R socket support code, and is Lisp version agnostic (it contains code specific to each of the Lisp platforms ACT-R runs on, and so is portable across a variety of Lisp implementations including all of those that ACT-R currently supports). The server code, however, also leverages multi-threading to allow simultaneous communication with multiple clients and parallel pursuit of mesh refinement. This code is based on the Allegro multi-threading model, and has not been ported to other Lisp implementations. Thus, while the client is platform independent, the server currently requires Allegro Common Lisp.

2.3 Determining an Appropriate Number of Model Runs

Due to their stochastic nature, ACT-R cognitive models produce a distribution of results for a single set of parameters. Experimenters typically run their models repeatedly to get an accurate estimate of the mean. Sometimes researchers run the model the same number of times as there are experimental subjects (this is widely known to be a questionable, if common, heuristic) and sometimes they run the model through some relatively large, fixed number of iterations, such as 100. The approach we took was to run the model as many times as is needed to generate a point prediction. That is, we used the variance of the points at the corners of an unexplored rectangular section of the parameter space (note that these are actually n-dimensional hypercubes) to estimate the variance at the center of the space, and use this estimated variance combined with the means of the corner points to generate an expected mean and variance for the midpoint of the unexplored mesh section. This allowed the application of a statistical power analysis using a t-test, which was parameterized by the desired alpha and beta levels for the t-test. Thus, given an experimenter determined a-priori tolerance and measured local noise (variance), the ACT-R AMR system will determine the number of runs needed to detect a difference greater than the specified tolerance between the expected mean (obtained by averaging the corners of the rectangular mesh) and the actual measured mean (obtained by running the model at the mesh midpoint).

The core of the ACT_R_AMR system is the implementation of a power-analysis based adaptive search mechanism, embedded in the ACT-R-AMR software, that concentrates effort of the system in areas where there is unpredicted local curvature. This allows for more focused and computationally efficient parameter searches, and ultimately allows for the pursuit of a larger number of ambitious questions with whatever computational resources are available.

3. Efficiency Improvements

To determine whether the AMR algorithm increased the efficiency of a parameter space search, an ACT-R model of a simple research task was run on a full set of parameter values and a space generated by the AMR algorithm. Results suggest that qualitatively informative results were obtained by sampling only 1% of the full space. Moreover, visualizations of the AMR generated space were of sufficient detail for the researcher to understand the dynamics of the model across the full parameter space.

4. Conclusions

The kind of combinatorial explosion that is possible out of the many available parameter combinations in a cognitive modeling system, such as ACT-R, can quickly overwhelm any computing resource. Thus, the road forward must include approaches that reduce the need to sample some of the ranges of those variables through intelligent search. The ACT-R AMR system described here is one implementation that achieves this goal.

References

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

Rai, M. M., & Anderson, D. (1981). Grid evolution in time asymptotic problems. Journal of Computational Physics, 43, 327-344.

Gluck, K., Scheutz, M., Gunzelmann, G., Harris, J., & Kershner, J. (2007). Combinatorics meets processing power: Large-scale computational resources for BRIMS. In *Proceedings of the Sixteenth Conference on Behavior Representation in Modeling and Simulation* (pp. 73-83). Orlando, FL: Simulation Interoperability Standards Organization.