

# Evaluating the Performance of Optimizing Constraint Satisfaction Techniques for Cognitive Constraint Modeling

Alina Chu (achu@umich.edu)

Department of Computer Science and Engineering; University of Michigan  
Ann Arbor, MI 48109

Richard L. Lewis (rickl@umich.edu)

Department of Psychology; University of Michigan  
Ann Arbor, MI 48109

Andrew Howes (howesa@manchester.ac.uk)

Manchester Business School; University of Manchester  
Manchester, UK.

## Abstract

Cognitive Constraint Modeling is an emerging modeling framework that allows a modeler to derive predictions of asymptotic performance from (a) a specification of architectural constraints; (b) a specification of a space of possible task strategies; and (c) an explicit objective (payoff) function. This approach has distinct advantages over traditional approaches in that it reduces the degrees of freedom inherent in specifying the details of particular strategies when explaining behavior; instead these specific strategies are selected on the basis of their maximum payoff given the architectural constraints. However, this approach potentially suffers from high computational cost and intractability on some problems because it is grounded in optimizing constraint-satisfaction techniques. To understand and address this problem, we have built a cognitive constraint problem generator which stochastically generates large populations of parametrically controlled problem instances on which we can test CCM performance. Initial results from our use of this tool have identified at least one clear property of problems—*order strength*—that has a significant and non-linear effect on performance time. These results, when extended, will pave the way to efficient constraint modeling.

## Introduction

It is a commonplace of psychology that human behavior is adaptive, but research over the last 10-15 years has revealed the astonishing degree to which strategic adaptation and variability manifests itself across behavioral timescales—even at the lowest levels of extremely rapid behavior (e.g., [Meyer and Kieras, 1997]). Cognitive architectures naturally admit of such strategic variation because they are programmable. Unfortunately, this general theoretical virtue becomes explanatory vice in trying to explain specific behaviors, because the strategies and knowledge supplied to the architecture become degrees of freedom in accounting for data [Kieras and Meyer, 1999, Newell, 1990]. This leads to the following question: How can we account for the complex details of adaptive interactive behavior—without effectively building into the model what we are trying to explain?

*Cognitive Constraint Modeling* (CCM) is an emerging framework for modeling that attempts to address this issue by allowing modelers to work in terms of formally defined *spaces* of strategies and formally defined architectural theories [Howes et al., 2004]. This is an advantage in that points in the strategy space are then

evaluated in terms of explicit theorist-defined objective functions (such as “go as fast as possible” or “minimize working memory load”) rather than being selected because of data fit.

The approach is realized computationally in a system called *CORE* (Constraint-based Optimality Reasoning Engine) [Howes et al., 2004]. Computational modeling in *CORE* has several distinguishing characteristics, but we focus on two here. First, descriptions of behavior are derived via constraint satisfaction over explicitly declared architectural, task, and strategy constraints, rather than running a simulation. Second, the complex details of behavioral control emerge in part from optimizing behavior with respect to explicit payoff functions ascribed to the human. Predictions of asymptotic performance are generated from architectural theory while minimizing specific assumptions about the details of the strategies. *CORE* has been used to model a range of different domains, including driving behavior [Brumby and Salvucci, 2006], a call center [Howes et al., 2005], interactions with an Automated Teller Machine task [Lewis et al., 2004], and interruptions during procedural cockpit tasks [Eng et al., 2006].

## The computational problem

Generating predictions from *CORE* can be computationally expensive: each *CORE* model constitutes a constraint satisfaction problem defining a potentially large search space of possible behaviors. Our practical experience with *CORE* suggests that even with relatively short stretches of behavior (under 20 seconds), it may be possible to create modeling problems that takes hours or days of CPU time to compute, as well as problems that are essentially intractable.

We do not believe this is a problem specific to our particular implementation of *CORE*. Rather, the computational complexity of these models is a fundamental problem that must be addressed for the general class of modeling methods in which the modeler is effectively specifying and exploring large spaces of possible strategies rather than specific (and possibly arbitrary) points in an implicit strategy space. This paper reports our first steps at understanding and addressing this computational problem.<sup>1</sup>

<sup>1</sup>This is not the first explicit attempt to systematically and quantitatively analyze the computational performance of a symbolic cognitive modeling technique; many current

## Our approach

Our approach to understanding and addressing this computational problem is to develop a *cognitive constraint model problem generator*, which allows us to stochastically generate large populations of parametrically controlled problem instances on which we can test the performance of CORE. Initial results from our use of this tool have identified at least one clear property of problems—*order strength*—that has a significant and non-linear effect on performance time. As we discuss below, this property has been previously identified in the literature on constraint satisfaction techniques.

In the remainder of the paper, we first summarize the basic structure of the computational problem as it arises in CORE, showing how CORE cognitive models are reduced to resource-bounded constraint satisfaction problems (RCSPs). We then briefly describe a particular class of algorithms for solving RCSPs and the time complexity of RCSPs. Next, we provide an overview of our problem generator framework, and then report on our initial results of experiments with this system. We conclude with the first set of general lessons we draw from these experiments, and suggest practical steps to take to make constraint modeling more efficient.

## From cognitive constraint models to scheduling problems

CORE takes in a specification of constraints and outputs a behavioral prediction. The architectural constraints are described in terms of cognitive, perceptual, and motor processes which use a set of processors/resources with set capabilities (following Model Human Processor [Card et al., 1983] and CPM-GOMS [John and Gray, 1995]) and high-level task grammars that define a space of possible task strategies. CORE transforms the architectural constraints and the task grammar specification into a constraint satisfaction problem, described as a set of variables, domains, and constraints. This formulation allows for the treatment of the models as scheduling problems, in particular, a Resource-Constrained Scheduling Problem (RCSP). CORE uses an off-the-shelf constraint solver: the Constraint Logic Programming over Finite Domains (CLPFD) Sicstus Prolog solver, to solve these constraint satisfaction problems. This solver utilizes the standard Branch and Bound algorithm for optimized solution search.

## Formulation as Resource-Constrained Scheduling Problem

A RCSP consists of a number of different activities, where each of the activities has a duration, and a maximum time domain in which it must be scheduled. Each activity also has a resource requirement which it uses

during its processing time. The activities must be scheduled within the total time bound (or optimized with respect to time), as well as stay within the total resource capacity at any given time. A solution schedule is an assignment of activities such that all specifications described are satisfied. There may be more than one solution per problem, depending on the tightness of the specifications.

A CORE cognitive constraint model can be interpreted as an RCSP. The variables consist of the processes, the domains of the variables consist of the duration distributions of the processes, and the constraints consist of (1) the task constraints denoting the flow of information required, (2) the capacity of the resources and the processes' allowed resource usage, (3) the architectural constraints such as the resources' capacities and amount of processing allowed at any given time, and (4) the minimization of time (or some other objective function). The space of solutions consists of the possible sets of domain values assigned to processes such that all the constraints are satisfied.

This interpretation is useful because it allows us to make contact with the large body of research already in place for RCSPs. In particular, parameters which characterize RCSPs and measures used to evaluate the performance of RCSP-solving algorithms have been developed, tested, and widely used [Kolisch et al., 1995, Kolisch et al., 1998, Patterson, 1976]. This means that input task models, the space of solutions (behavior prediction), and the performance of the constraint solver itself all may be evaluated in terms of these established RCSP metrics.

## Complexity, phrase transitions, and control parameters

RCSP itself is an NP-complete problem [Herroelen and De Reyck, 1999]. However, there are properties of NP-complete scheduling problems which can be exploited to provide a characterization of the complexity of the problem space. A particularly useful one is that many NP-complete problems exhibit *phase transitions*, which are sudden changes in computational complexity. Phase transitions are products of the way *control parameters*, parameters that characterize the problems to be solved, affect the hardness of the problem. Hard to solve instances occur around a critical value of a control parameter, and are responsible for the phase transition. [Herroelen and De Reyck, 1999]. Therefore one of the goals of the present work is to search for those control parameters that predict phase transitions in cognitive constraint models.

For the RCSP, most control parameters fall into these categories: (1) size of the problem, (2) topological structure (morphology) of the problem, or (3) availability of different resource types in the problem [Herroelen and De Reyck, 1999]. A widely-used example of these is *order strength*, which falls into the topological structure category. Order strength is intuitively the strength of the partial ordering, or the density of the network.

---

production system architectures such as Soar and EPIC [Meyer and Kieras, 1997] were made viable by early foundational work on the production rule match which led to efficient new algorithms [Forgy et al., 1984, Tambe et al., 1990].

**Definition 1** Order strength is the number of precedence relations divided by the theoretical maximum of such precedence relations:  $T/U$ , where  $U = n(n - 1)/2$ , and  $n$  is the number of activities.

In terms of the task model RCSPs, activities are processes, precedence relations are the cascades denoting the flow of information between processes, and the network is the schedule itself, with the processes as nodes and the precedence relations as topological constraints between them. Order strength has been shown to successfully characterize phase transitions for many of the main RCSP-solving algorithms such as Branch-and-Bound [Herroelen and De Reyck, 1999]. There are many other RCSP control parameters; below we present empirical evidence that order strength is significant for cognitive constraint models.

## Cognitive Constraint Problem Generator (CCPG) Framework

The motivation for the CCPG is to develop a tool which can be used to systematically test the performance of CORE on large sets of model instances that we parametrically define. This will allow us to identify characteristics of problems which may increase or decrease efficiency. In general, the CCPG should be able to create, run, and evaluate scheduling problems of varying complexity with respect to some controlled measure. Here we present results with respect to the controlled measure of order strength (refer to Definition 1). The requirements for this tool include:

1. The ability to generate scheduling problems that have similar structure to the cognitive modeling problems to which CORE is intended to be applied.
2. The ability to provide the modeler with enough input flexibility to control some aspects of the scheduling problems generated without losing the advantage of the large variability between types of problems. For example, one input parameter may be the number of processes in the generated problem: using this, the modeler can relatively control the size of the problem, however there is a huge amount of variability in the types of problems that can be generated with a certain number of processes.
3. The ability to guarantee that each generated problem in fact has a solution.

Given these requirements, we made the design choice of creating a problem generator rather than using an off-the-shelf RCSP generator, because existing generators (e.g., [Kolisch et al., 1998, Schwindt, 1995]) do not satisfy 1 and do not always satisfy 2 and 3 in tandem.

### CCPG structure and implementation

The CCPG is implemented in three main stages: the first stage generates fully constrained schedules, the second stage removes constraints systematically, creating under-constrained scheduling problems, and the third stage executes and evaluates the performance of these problems in CORE. Refer to figure 1.



Figure 1: Stages of the cognitive constraint problem generator.

### Stochastic generation of fully-constrained schedules

The first stage receives input parameters from the modeler and outputs a number of fully-constrained schedules generated from those input parameters and some randomization. The two types of input parameters are the characteristic and shaping parameters. The characteristic parameters (number of processes, number of resources, etc.) describe the basic structure of the schedules to be generated, characterizing the bounds of the schedule space to be explored. The shaping parameters control the general shape of the schedule, approximating the location of this schedule in the schedule space. The algorithm grows the schedules in a tree-like implementation, where the precedence relations between nodes of the tree correspond to the flow of information between processes. The scheduling of each of the processes is dependent on probability distributions which are affected by the shaping parameters. However, these parameter constraints are not tight enough to limit the number of schedules possible to generate.

The stochastic method allows for the generation of a huge range of similarly-structured schedules, supporting the requirement for variability in 2. To test if the generator gives sufficient parametric control so that schedules produced are similar to cognitive problems of interest (requirement 1), an assessment was carried out using a regression analysis and common metrics used to describe directed acyclic graphs. The analysis showed that the parametric control is significant in shaping the schedule structure and provided guidelines for the choosing of parameters to manipulate schedule formation. This analysis also showed that solution schedule variation is extremely large despite the parametric control (requirement 2) and that the space of possible generated solutions is much larger than the current scope of existing models so absolute verification of a mapping from the space of possible generated task models to the space of human task models is not feasible, at least at this time (requirement 1).

### Turning schedules into problems via constraint removal

In this stage of the CCPG, constraints are removed systematically from the fully constrained schedules to create under-constrained scheduling problems. This systematic removal guarantees that every problem generated is solvable (but does not guarantee what the solution is) satisfying requirement 3, and allows the varying of complexity to be controlled for evaluation with respect to the experimenter-chosen controlled measure. In this implementation, an RCSP control parameter, order strength, is chosen. To systematically vary the or-

der strength, the type of constraints removed must be ordering (flow of information) constraints (definition 1 precedence relations). This investigates the strength of the partial ordering as a property of scheduling problems which may help characterize the space of solvable task models for CORE.

## An Empirical Investigation of the Effects of Order Strength

We used the problem generator to empirically investigate the effects of order strength variation on the performance of the branch-and-bound algorithm. In what follows, we first describe the structure of the experiment and the parameters varied, and then report the results in terms of execution time and time-out rates as a function of order strength.

In light of this order-strength specific evaluation, we have found it useful to describe the results in two stages. First we establish the low run time model trends, using a low time-out value meaning the problems which take more than a certain CPU time value to run will be stopped and remain unfinished. This is a user-specified quit mechanism in Sictus Prolog.) Then we address the question of fitting higher run time models to these low run time curves, evaluating the ability of the low run time curves to predict the behavior of higher run-time models.

### Structure of the experiment

We embedded the generator in nested loops of execution which ran large numbers of models systematically through CORE under the same machine conditions and recorded the CPU time per run.

Our models include multiple fully-constrained schedules using the same exact input parameters, and also multiple fully-constrained schedules using different numbers of processes but otherwise unchanged input parameters. Each schedule was the starting point for multiple sequences of constraints removed (the sequences ranging from no constraints to all constraints removed), each additional removal amounting to a separate model. This allows for time complexity to be analyzed across (but not limited to) numbers of processes, number of constraints removed (NCRs), and order strength.

### Low run time models

Figure 2 shows the results of the analysis characterizing the behavior of low run time models: their relationship between order strength and execution time. The data in this plot is the result of a range of 5-30 process problems, each with 10 different fully-constrained schedules with 10 different sequences of constraints removed each. This is a total of 52,500 models. Each line is a local polynomial regression fitting of the data points for the models of one number of processes, and each line is labeled with that number of processes. The ‘‘Average CPU time per trial’’ is the average search time the constraint solver needs to find a solution for that model. The time-out is 100ms. (The data plotted here is only for the models

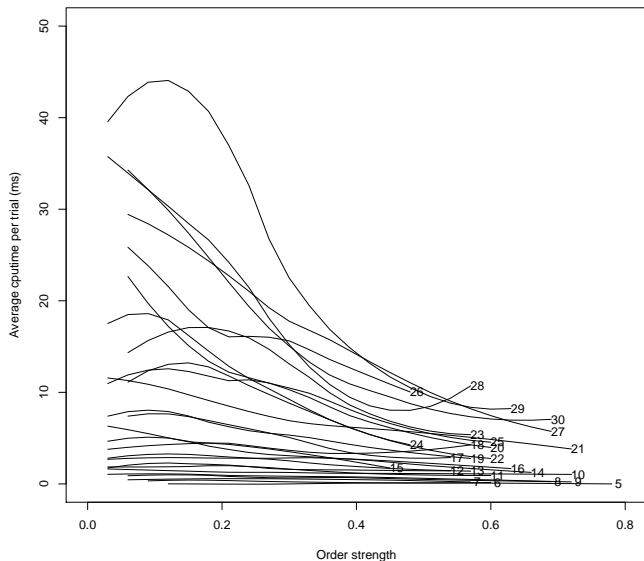


Figure 2: Time complexity of models of different numbers of processes (labeled numbers in the plot) according to order strength. For low run time models.

that finished running and did not time-out.) The reason for this tight time-out value is the result of a search through ranges of time-outs which exhibited an interesting processing time duality: their run time either took hours or was under a minute. With low run time models it is computationally feasible to run enough models to find a trend if one exists, and if that trend can predict the complexity of the high run time models, then the control measures used are good predictors of the the task models’ run time complexity.

The trends in Figure 2 curve downward, and more sharply with increasing numbers of processes. This is consistent with the expectation that increasing order strength (increasing strength of the partial ordering of a problem) will decrease the run time. The stronger the structure of the problem, the less time it will take to find a solution because the search space is smaller. With models of smaller numbers of processes, the search space is small to begin with, and so the run time is still low regardless of order strength and the lines are rather low and flat.

The slight downward turn (near 0 order strength) at the peaks of some of the lines is an effect of the order strength formulation: as the order strength approaches 0, the equation  $T/U$  ensures that the ratio of existing precedence relations with respect to the theoretical maximum of all precedence relations is getting smaller. For models with large numbers of processes, the ratio can approach 0 fairly quickly without the need for very small numbers of existing precedence relations because of the unstructured nature of the problem. The point at 0 can only occur when there are no existing precedence rela-

tions, which is fairly easy (small run time) to schedule a problem with no constraints.

### Higher run time models

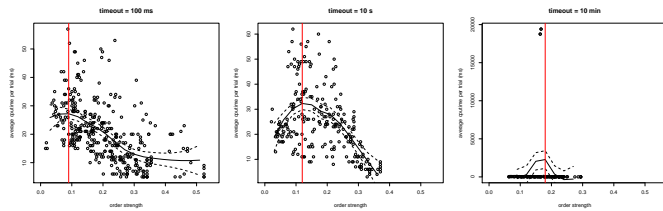


Figure 3: Time complexity across increasing run time models with respect to order strength. Raw data is behind the local polynomial regression fitted line. Dotted lines represent error within 95% confidence interval.

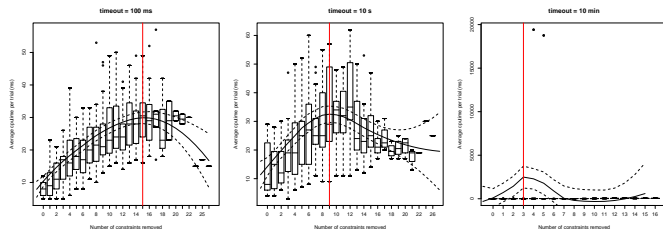


Figure 4: Time complexity across increasing run time models with respect to number of constraints removed. Raw data in boxplot form behind the local polynomial regression fitted line. Dotted lines represent error within 95% confidence interval.

Figures 3 and 4 are examples of the results comparing the behavior of the previous low run-time models to two other sets of higher run time models. The changes across the systematically varied run times was gradual; these particular sets were chosen to illustrate the differences between the higher and lower run time models. (These examples also come from a range of 5-30 process models, same as the low run time models). The trends shown by these examples are representative of the results gathered; they are shown by example to facilitate viewing of the data.

Figure 3 shows the relationship between order strength and execution time across increasing run time models. The vertical line notates the order strength at which the maximum execution time of the smoothed fit occurs, used to compare the hill-shaped trends observed in the lower run time models with the higher run-time models. The top of the hills shift between the different time-out plots, however there is a set range between 0.1 and 0.2 order strength within which the peak of the hill (across all models) lies.

The processing time duality referring to the sudden increase in computational complexity of certain models can be seen here in the 10min time-out plot. The majority of the models have an execution time similar to the

100ms and 10s models, however there are a few models which have a very sudden increase in complexity. These models *cause* the smoothed hill in that plot to be peaked at that order strength, however, when compared to the lower time-out plots, the peak is very close to the prediction of the lower run time models. Because these trends were observed across the total data set, *it suggests that order strength may be able to identify those problematic models that have run time complexity many orders of magnitude above its similar but low run time neighbor models.*

However, order strength does not explain the reason for the jump in complexity. We are currently examining “minimal pairs” of scheduling problems that are identical in number of processes and resources and differ only in the removal of a single constraint—but which show radically different complexities in that one of the problems takes at least 3 orders of magnitude longer to solve than the other. Such minimal pairs will help us to identify perhaps a different set of control parameters underlying the phase transitions.

These particulars make it worthwhile to compare the same run time data against the number of constraints removed (NCR) from the fully-constrained schedule. The difference between NCR and order strength is subtle: the order strength measures a structural aspect: the density of the problem regardless of the size, and the particular constraint removed does not matter to the NCR measure, however it does to order strength because of the transitive relations involved. Figure 4 consists of the same data as Figure 3. The same basic trends can be recognized (as expected, because of the correlation between the number of constraints removed and order strength). However the shift in the peak of the hill covers a wide range of the NCR metric and does not seem to be bounded well. This suggests that order strength might be a better indicator of those highly computationally complex run time models.

## Discussion

We have examined the computational nature of task models used in Cognitive Constraint Modeling. CCM is an approach to computational modeling which is unique in that it formulates cognitive models as optimized constraint satisfaction problems. Although it may be argued that all cognitive architectures treat modeling as a type of constraint satisfaction problem with resource constraints, it has not been as precisely formulated as it is here. CCM models may be interpreted as resource-constrained scheduling problems, allowing us to take advantage of the complexity and control measures that the RCSP literature has to offer. We have created a problem generator framework to allow for time complexity experimentation with the Branch-and-Bound algorithm used by CORE—a standard approach to algorithm investigation in the planning and constraint satisfaction fields—and we have exploited the advantages of RCSPs both in the framework and in the choice of analyses. This has allowed us to take a very close look at the structure of the problems themselves, and their effect on the com-

putational feasibility of CORE modeling. Our initial use of this tool has yielded promising results. First, it has confirmed that order strength is a predictor of problem complexity for the subclass of RCSPs that are generated in cognitive constraint modeling. Second, it has created a large set of minimal pairs of problems whose constraint removal straddles phase transitions, which should yield insight into the precise characteristics that give rise to such transitions.

The practical implications of this work are significant: If we can precisely characterize the space of the computational complexity of CORE models, we can use the information to make constraint modeling not only more efficient, but in some cases, possible at all. These changes to the constraint modeling may include changing the model specifications by adding or subtracting constraints in a way that does not compromise the theoretical goals of the modeler; use different specialized algorithms on models of high complexity (some algorithms may be better than others at particular kinds of hardness; e.g.[Patterson, 1976]); and also providing the modeler with characterization of the expected complexity of models *before* they are executed.

### Acknowledgments

This work was supported by ONR grant N00014-03-1-0087 and NASA grants NNA05CS65A and NNA04CK14A.

### References

- [Brumby and Salvucci, 2006] Brumby, D. P. and Salvucci, D. D. (2006). Towards a constraint analysis of human multitasking. Poster presented at the 7th International Conference on Cognitive Modeling.
- [Card et al., 1983] Card, K., S., Moran, P., T., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum, Hillsdale, NJ.
- [Eng et al., 2006] Eng, K., Lewis, R. L., Tollinger, I., Chu, A., Howes, A., and Vera, A. (2006). Generating automated predictions of behavior strategically adapted to specific performance objectives. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, Montreal, Canada.
- [Forgy et al., 1984] Forgy, C., Gupta, A., Newell, A., and Wedig, R. (1984). Initial assessment of architecture for production systems. In *Proceedings of the National Conference for Artificial Intelligence (AAAI)*, pages 116–120, Austin, TX.
- [Herroelen and De Reyck, 1999] Herroelen, W. and De Reyck, B. (1999). Phase transitions in project scheduling. *Operational Research Society*, 50:148–156.
- [Howes et al., 2005] Howes, A., Lewis, R. L., Vera, A. H., and Richardson, J. (2005). Information-requirements grammar: A theory of the structure of competence for interaction. In *Proceedings of the 27th Annual Meeting of the Cognitive Science Society*, Stresa, Italy.
- [Howes et al., 2004] Howes, A., Vera, A. H., Lewis, R. L., and McCurdy, M. (2004). Cognitive constraint modeling: A formal approach to supporting reasoning about behavior. In *Proceedings of the Cognitive Science Society*, Chicago.
- [John and Gray, 1995] John, B. E. and Gray, W. D. (1995). Cpm-goms: an analysis method for tasks with parallel activities. In Katz, I., Mack, R., and Marks, L., editors, *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, Denver, Colorado. ACM Press, New York, NY.
- [Kieras and Meyer, 1999] Kieras and Meyer, D. (1999). The role of cognitive task analysis in the application of predictive models of human performance. In Schraagen, J. M., Chipman, S. F., and Shalin, V. L., editors, *Cognitive Task Analysis*, pages 237–260. Lawrence Erlbaum Associates, Mahwah, NJ.
- [Kolisch et al., 1998] Kolisch, R., Schwindt, C., and Sprecher, A. (1998). Benchmark instances for project scheduling problems. In Weglarz, J., editor, *Handbook on Recent Advance in Project Scheduling*. Kluwer, Norwell, MA.
- [Kolisch et al., 1995] Kolisch, R., Sprecher, A., and Drexel, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703.
- [Lewis et al., 2004] Lewis, R. L., Vera, A. H., and Howes, A. (2004). A constraint-based approach to understanding the composition of skill. In *Proceedings of the International Conference on Cognitive Modeling (ICCM)*, Pittsburgh.
- [Meyer and Kieras, 1997] Meyer, D. E. and Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, 104:3–65.
- [Newell, 1990] Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.
- [Patterson, 1976] Patterson, J. H. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, 23:95–123.
- [Schwindt, 1995] Schwindt, C. (1995). Progen/max: a new problem generator for different resource constrained project scheduling problems with minimal and maximal time lags. Research report wior-449, Universität Karlsruhe.
- [Tambe et al., 1990] Tambe, M., Newell, A., Rosenbloom, and S., P. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5:299–348.