Towards a Computational Model of Why Some Students Learn Faster than Others

Nan Li, Noboru Matsuda, William W. Cohen, and Kenneth R. Koedinger
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213 USA
nli1@cs.cmu.edu, mazda@cs.cmu.edu, wcohen@cs.cmu.edu, koedinger@cs.cmu.edu

Abstract

Learners that have better metacognition acquire knowledge faster than others who do not. If we had better models of such learning, we would be able to build a better metacognitive educational system. In this paper, we propose a computational model that uses a probabilistic context free grammar induction algorithm yielding metacognitive learning by acquiring deep features to assist future learning. We discuss the challenges of integrating this model into a synthetic student, and possible future studies in using this model to better understand human learning. Preliminary results suggest that both stronger prior knowledge and a better learning strategy can speed up the learning process. Some model variations generate human-like error pattern.

Introduction

Computer-based learning environments are designed to facilitate learning processes. Traditional systems are usually not sensitive to individual user differences, and this may hurt the effectiveness of the systems. Metacognitive educational systems, on the other hand, provide a much better learning experience by dynamically adapting to users based on the "mental states" of the systems. In order to build such systems, it is important that we are able to model such "mental state". It is well known that learners of better metacognition often acquire knowledge faster than others who do not. If we had better models of such learning, we would be able to build a better metacognitive educational system.

Previous work (Chi *et al.* 1981) showed that one of the key factors that differentiates experts and novices is that experts view the world in terms of deep functional features, while novices see in terms of shallow perceptual features. The deep features are important prior knowledge in achieving effective learning. However, how these deep features are acquired is not clear. A computational model of deep feature learning that fits student learning data would be a significant achievement in theoretical integration within the learning sciences, and reveal insights on improving metacognitive systems.

In this paper, we propose a novel approach to modeling deep feature acquisition through the use of machine learning techniques. Initially, we assume that the input of the system is a set of feature recognition records. We propose a computational model using grammar induction algorithms to acquire deep features encoded in these records, and report preliminary results. Later, we discuss the challenges on integrating this computational model into a machine learning agent, SimStudent, (Matsuda *et al.* 2009), where no feature recognition records are present. The benefit of this integration is twofold. First, we could carry out more extensive studies in simulating student learning, which is important in building learning sciences. Second, SimStudent succeeded at learning domains such as algebra, but was given algebra-specific (strong) background knowledge like coefficient. The extended synthetic student could potentially learn algebra equation solving without any algebra-specific (weak) background knowledge. After that, we discuss future studies on matching the proposed model with real student data after integration.

Table 1: Probabilistic context free grammar for coefficient in algebra

Primitive symbols: $-, x$;
Non-primitive symbols: $Expression, SignedNumber, Variable, MinusSign, Number$;
$Expression \rightarrow 1.0,\ [SignedNumber]\ Variable$
$Variable \rightarrow 1.0,\ x$
$SignedNumber \rightarrow 0.5,\ MinusSign\ Number$
$SignedNumber \rightarrow 0.5,\ Number$
$MinusSign \rightarrow 1.0,\ -$

A Computational Model of Deep Feature Learning

The input of the system is a set of feature recognition records consisting of an original problem (e.g. an expression), and the feature to be recognized from the problem (e.g. a coefficient in the problem). For example, in the algebra domain, the feature -3 is recognized from the problem $-3x$ as the coefficient. Many algebra student errors are caused by incorrect parsing. A common error made by students is considering 3 as the coefficient rather than -3. Previous work (Matsuda *et al.* 2009) has shown that students with no prior knowledge of deep features such as "coefficient" learn

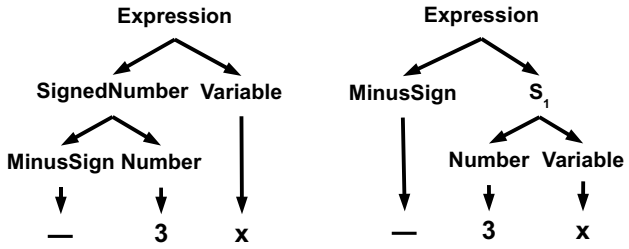


Figure 1: Correct and incorrect parse trees for $-3x$.

slowly.

After a careful examination of the problem, we discovered an interesting insight. The feature learning task can be viewed as a grammar induction problem. As shown in Table 1, it is clear that equations can be formulated as a context free grammar. The parse structure of $-3x$ based on the grammar is shown at the left side of Figure 1. More interestingly, the perspective of viewing feature learning tasks as grammar induction problems also explains the cause of student errors, which is incorrect parsing of the input as demonstrated at the right side of Figure 1. To build our learning model based on this observation, we extended the learning algorithm proposed by Li et al. (2009), since it acquires PCFG from observation sequences without any prior structural knowledge. The system consists of two parts, a greedy structure hypothesizer, which creates non-primitive symbols and associated reduction rules as needed to cover all the training examples, and a Viterbi training step, which iteratively refines the probabilities of the reduction rules. More details are described in (Li *et al.* 2009).

Feature Learning

The system first acquires the grammar with Li et al.'s algorithm. After that, our learning system tries to identify an intermediate symbol in one of the rules as the target feature. This process is done in three steps.

The system first builds the parse trees for all of the observation sequences based on the acquired rules. For instance, in algebra, suppose we have acquired the PCFG shown in Table 1. The associated parse tree of $-3x$ is shown at the left side of Figure 1. Next, for each sequence, the learner traverses the parse tree to identify the intermediate symbol associated with the feature subsequence, and the rule to which the intermediate symbol belongs. In the case of our example, the intermediate symbol is $SignedNumber$, and the rule is $Expression \rightarrow 1.0,\ SignedNumber\ Variable$. For some of the sequences, the feature may not be generated by a single symbol, which usually happens when the acquired PCFG is not in the right structure. In this case, the system will ignore the current sequence. Last, the system records the frequency of each symbol rule pair, and picks the pair that matches the most training records as the learned feature. For instance, if most of the input records match with $SignedNumber$ in $Expression \rightarrow 1.0,\ SignedNumber\ Variable$, it will be considered as the target feature pattern.

After learning the feature, when a new problem comes, the system will first build the parse tree of the new problem based on the acquired grammar. Then, the system extracts the subsequence associated with the feature symbol from the parse tree, and returns it as the feature.

Transfer Learning Using Prior Knowledge

Previous research has shown that prior knowledge affects the effectiveness of later learning tasks (Booth *et al.* 2007; Matsuda *et al.* 2009). Learners that actively acquire high-level knowledge (e.g. non-primitive symbols) maintain stronger prior knowledge than others who do not. Therefore, we designed a learning mechanism that transfers the acquired grammar and the application frequencies of the rules from previous tasks to future tasks to accelerate the learning process.

More specifically, during the acquisition of grammar in previous tasks, the learner records the acquired grammar and the number of times each grammar rule appeared in a parse tree. When a new learning task comes in, the learning algorithm first uses the known grammar to build the most probable parse trees for each new record in a bottom up fashion, until there is no rule that can further merge two parse trees to a single tree. Then, the system switches to the original greedy structure hypothesizer and acquires new rules based on the partially parsed sequences. In the next phase, Viterbi training, the learning algorithm counts the applied rule frequency not only with the training problems, but also with the recorded frequency from previous tasks.

Note that it is possible that after acquiring new rules with new examples, during the Viterbi training, the parse trees for the training examples in the previous tasks have changed, and the recorded frequencies are no longer true. However, as long as the acquired grammar is true for previous tasks, we believe the frequency numbers still serve as a good indication of the probabilities associated with grammar rules. Moreover, by recording the frequencies instead of rebuilding the parse trees for all previous training examples in each cycle, we are able to save both space and time for learning. It is also cognitively plausible since people usually do not have a large memory capacity to remember all past experiences during learning.

Effective Learning using a Semantic Non-Terminal Constraint

In spite of stronger prior knowledge, we also believe that better learning mechanism could yield faster learning experience. Hence, we further extended our learning mechanism to make use of a "semantic non-terminal constraint" embedded in training data during learning.

In the original learning algorithm, during the process of grammar induction, the learner acquires whatever grammar that could generate the training traces without differentiating the subsequence associated with the feature from other subsequences in the training example. It is possible that two grammars can generate the same set of traces, but only one has the feature symbol embedded in it. But we cannot be sure that the original learner would learn the right one. Forcing all the feature subsequences to correspond to some target

Table 2: Method summary

2-by-2 learners:	L00,no transfer, no non-terminal constraint
	L01, no transfer, with non-terminal constraint
	L10, transfer, no non-terminal constraint
	L11, transfer, with non-terminal constraint
Three tasks:	T1, learn signed number
	T2, learn to find coefficient from expression
	T3, learn to find constant from equation
Three curricula:	T1 → T2
	T2 → T3
	T1 → T2 → T3
Number of training condition:	10
Training size in all but last tasks:	10
Training size in the last task:	1, 2, 3, 4, 5
Testing size:	100

feature provides us with a constraint on the grammar, which is, there should be a non-terminal symbol associated with the target feature.

Therefore, we extended the learner to use this semantic non-terminal constraint. Before learning the grammar rule for the whole sequence, the learner first extracts all the feature subsequences from training problems, and acquires a feature grammar for it. Then, the learning mechanism replaces all the feature subsequences in the training records with a *semantic terminal symbol*, and learns a new grammar for the updated sequences. Finally, two grammars are combined, and the semantic terminal is relabeled as a non-terminal symbol and associated with the start symbol for the feature grammar

Preliminary Results

In order to understand the behavior of the proposed model, we carried out a preliminary experiment before the integration. The method we used is shown in Table 2. There were 2-by-2 (4) alternative versions of the proposed learning model in the study. We designed three curricula using three tasks, where knowledge learned from previous tasks is helpful in accomplishing later tasks. There were also 10 training sequences to control for a difference in training problems. In all but the last task, each learner was given 10 training problems following the curriculum. For the last task, each learner was given one to five training records. The learning gain was measured by the accuracy of the recognized features for the last task under each training condition.

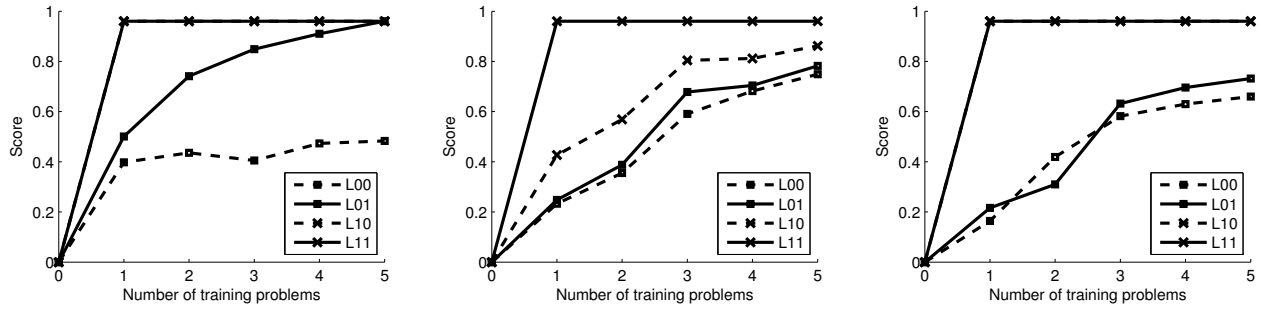
We also compared the type of errors made by the model with the real student errors. We divided student responses from a study of 71 high school students used Carnegie Learning Algebra I Tutor into four types, 1) do not know, where a student does not know how to proceed, 2) correct, where a student gives the correct answer, 3) common error bug, where a student considers $Integer$ as the coefficient of $-Integer\ x$, 4) others, all other responses. During testing, we computed the percentage of each response type in both real student data and learner generated results, and compared whether the pattern of learner generated results fit to that of real student data qualitatively.

Impact of Deep Feature Learning on the Rate of Learning: As shown in Figure 2(a), with curriculum one, all four learners acquired better knowledge with more training examples. Both learners with transfer learning, L11 and L10, have the steepest learning curve. The learner with better learning strategy, L01's learning curve is not as steep as L10 and L11. This suggests that with transfer learning, learners are able to acquire knowledge quicker than those without transfer learning. L00's learning curve is the least steep one. Comparing the base learner, L00, and the learner with non-terminal constraint, L01, we can see that a better learning strategy yields a steeper learning curve. Similar results were also observed with curriculum two and curriculum three. We can also see that in all three curricula, the transfer learner, L10, always outperforms the learner with semantic non-terminal constraint, L01. This suggests that prior knowledge is more effective in accelerating future learning than better learning strategies.

Impact of Deep Learning on Error Matching: In real student data, among all of the errors, the most common error made by students was dividing both sides by A instead of $-A$. The error rate was 6.15%. Comparing the behavior of the proposed models, we see no learners were able to recognize any coefficient without any training data, which is appropriate since students would not be able to answer any questions before being taught. After being trained with one to five problems, L00 generated the most common error in testing. Besides that all other incorrect answers belong to category "do not know". All the other learners did not make any mistakes on the test questions.

Integrating Deep Feature Learning into SimStudent

In spite of the promising result, the above model operated as a stand-alone module that was not part of an integrated system, which limits the capability of the system. In this section, we explore the challenges and opportunities in adapting this mechanism into a larger system. In particular, we chose



(a) (b) (c)

Figure 2: Learning curves for four learners in curriculum (a) from task one to task two. (b) from task two to task three. (c) from task one to task two to task three.

SimStudent (Matsuda *et al.* 2009), a state-of-art machine learning agent that acquires procedural knowledge from examples and through problem-solving experiences. This integration could potentially reduce the amount of prior knowledge needed for SimStudent, and also enables us to carry out more extensive experiment in understanding the model behavior.

A Brief Review of SimStudent

Before discussing the integration, let's briefly review SimStudent. SimStudent is a machine-learning agent that inductively learns skills to solve problems from examples. It is a realization of programming by demonstration (Lau and Weld 1998) with an underlying technique of inductive logic programming (Muggleton and de Raedt 1994). The examples can be given at once as demonstrated solutions (i.e., worked-out examples) or interactively as guided problem-solving feedback from a tutor (i.e., tutored problem solving). An example given to SimStudent shows a rule application. Each example can be labeled according to the skill applied (e.g., "divide") to reduce the cost of a search. The example typically provides another piece of information, called the focus of attention (FoA) showing where to pay attention when the skill is applied. For example, to demonstrate to "divide both sides of -3x=6 by -3", the tutor may label the step as "divide" and specify "-3x" and "6" (i.e., each side of the equation) as FoA. When learning from demonstrated solutions, SimStudent is engaged in passive learning. Each step in the demonstrated worked-out examples serves as a positive example for the specified skill application. At the same time, it also serves as an /implicit/ negative example for all other skills that have been demonstrated thus far. When SimStudent is engaged in tutored problem solving, the tutor provides both positive and /explicit /negative examples. Positive examples include not only demonstrated solutions, but also generated steps that receive positive feedback from the tutor. Explicit negative examples are generated steps that receive negative feedback from the tutor.

Production Rules: SimStudent generates production rules as a result of learning. Each production rule models a single skill application, although a skill may be modeled as a set of disjunctive rules. A production rule indicates /when/ to apply a rule to what information found /where/ in the interface and /how/ the problem state should be changed. For example, the rule to "divide both sides of -3x=6 by -3" would read "given a left-hand side (i.e., -3x) and a right-hand side (6) of the equation (the where-part), when the left-hand side is a variable term and the right-hand side is a constant term (the when-part), then get the coefficient of term on the left-hand side and divide both sides by the coefficient (the how-part)." Accordingly, the production rule consists of three major parts - (1) the WME-path (the where-part) representing a hierarchical structure of the working memory elements that are realized in a problem given to solve, (2) the feature test (the when-part) representing conditions to apply the production rule, and (3) the operator sequence (the how-part) representing actions to be taken.

Background Knowledge: To learn the WME-path correctly, SimStudent must be given hierarchical information of the working memory elements. The hierarchy of WME reflects strong domain knowledge such as "an equation has left- and right-hand side." Learning the feature test and the operator sequence also requires "strong" domain knowledge such as variable term, constant term, and coefficient as mentioned above. It has been empirically tested that the strength of the background knowledge affects the rate and the accuracy of SimStudent's learning (Matsuda *et al.* 2009). Especially, to model (human) students' learning, especially to model errors that students make, we need to provide "weak", more perceptually grounded background knowledge such as perceiving "3" as a number before a variable, instead of a coefficient. Since the "weak" background knowledge, by definition, applies broader context, modeling SimStudent with "weak" background knowledge also expands its generality. The proposed attempt to integrate the deep feature learning to SimStudent is to model the process of learning "strong" domain knowledge from "weak" background knowledge (and transfer) as well as to model more errors that students make due to an inappropriate application of such "weak" background knowledge.

Integrating Deep Feature Learning

As we can see from the above description, SimStudent requires a set of strong (domain specific) background knowledge as input to learn procedural knowledge to solve the problems. The proposed model, on the other hand, only needs weak knowledge such as what is an integer. [1] If we could integrate this model into SimStudent, we would be able to achieve a system that requires only weak prior knowledge as input.

Integrating Learning Process into SimStudent: Despite the promising future, challenges remain. First, SimStudent learns knowledge by demonstration, and thus no explicit labeling of deep features would be provided. One possible way of achieving the integration is asking the tutor/teacher to circle the deep features during each step, and label them with the name of each feature. This strategy, if used under the learning by teaching situation, forces the tutor to self-explain his/her behavior. This could potentially improve the learning speed of the tutor.

Another possible choice is when there is no tutor providing the labeling information, SimStudent should be able to identify the deep features in the problem as well as the labels associated with them automatically, based on the demonstration steps. This is a harder but more interesting problem. To do this, after each demonstration step, SimStudent tries to match the input string with the current FoAs. If some part of the string matches with any of the current FoAs, we consider the partial string is potentially a deep feature associated with that FoA. For example, for problem $-3x = 6$, the first input message is $divide\ -3$. In this case, the current FoAs include $-3x$ and 6. Since -3 in $divide\ -3$ matches with $-3x$, we consider -3 as a candidate deep feature embedded in $-3x$. Then, SimStudent is able to identify possible deep features in the task. Nevertheless, this is not sufficient. There are usually more than one type of deep features in the domain (e.g. coefficient, constant). Hence, SimStudent should be able to cluster deep features into several groups, each associated with some specific deep feature. This could be done by a clustering algorithm using demonstration signals such as the name of the rule learned with that demonstration, the name of the operator associated with the deep feature. Other signals such as hint messages also provide assistant information in labeling deep features.

Extending the Model to Support Noisy Input from SimStudent: One potential risk the learning algorithm might face in this case is that the proposed model assumes all of the input records are correct records, which is no longer true. Since SimStudent is now automatically generating input records based on demonstration steps, it is likely that some of the generated records are incorrect. Although the proposed model is robust to some extent due to its statistical nature, the proposed model might still suffer due to the noise in the input records. In response to this, we plan to further extend the proposed model to support negative feedback, where the model will not only try to acquire the grammar that is most likely to generate the positive records, but also is most unlikely to generate the negative records. The source of the negative feedback comes from a wrong application of a production rule. More specifically, in the original model, the greedy structure hypothesizer always stops whenever there is at least one parse tree for each training record. This greedy strategy can get the learning algorithm stuck at a local optimum point. When there is a negative feedback available on the current grammar, instead of just considering the most probable parse tree as the parsing structure of the input message, the learning model should also consider the second or third most probable parse tree as the candidate parsing of the input record. This process continues until the learning algorithm finds a grammar that is consistent with both the positive and negative training records. In short, with the deep feature acquisition strategy described above, the extended SimStudent does not require more demonstration information than before, but could potentially reduce the amount of prior knowledge needed.

Extending SimStudent to Use the Acquired Knowledge: After the grammar has been learned, how to enable SimStudent to use this knowledge to assist its learning process as well as future execution also present challenges to integration. To this end, we propose to extend the WME path structure in the LHS of the production rule. Instead of always having a cell as the leaf node of a WME path, we further extend the original WME path to include the most probable parse tree of the cell based on the acquired grammar. Also, not only the cells are considered as FoAs, the intermediate symbols and terminal symbols(subcells) in their parse trees are also set to be current FoAs. For instance, -3 and x would also be added to the FoAs. Then, SimStudent would be able to acquire knowledge based on the extended WME path. In the $-3x = 6$ example, a production rule learned will not need to first extract the coefficient from $-3x$. Instead, it will only need to divide both sides with -3. If there is no prior knowledge given on extracting coefficient, the original SimStudent might fail in learning the right production rule, whereas the extended SimStudent could still acquire the production rule based on the extended WME path. The facts associated with the subcells will also be generated and updated in each demonstration step. After the rule is learned, whenever the rule is satisfied, SimStudent would apply the rule. A failure on the rule application also serves as a negative feedback discussed in the previous paragraph. With the integration just described, the extended SimStudent should be able to yield faster learning based on prior learning, and to acquire procedural knowledge given only weak prior knowledge.

Future Studies after Integration

After integration, both the learning capability of the proposed model and the extensiveness of the studies we could carry out would be improved. We first tested the learning capability of the integrated model with a simple example, where the computational model was first trained on the coefficient task. After the model has learned the correct grammar of coefficient, we extended WME path to support the acquired grammar, and trained the extended SimStudent on solving equations of the form, $Integer\ x = Integer$, and

[1] If given enough training examples, the proposed model may not need any prior knowledge.

then tested on a problem $3x = 6$. The extended SimStudent successfully acquired the production rules, and solved the testing problem without asking for help. The acquired production rule for dividing both sides had only one weak operator, whereas the original SimStudent required a strong operator. This preliminary result suggests that the extended SimStudent may be able to acquire new domain knowledge without being given the strong background knowledge currently needed by SimStudent.

We plan to carry out more thorough studies to get a better understanding of the grammar learning extension of SimStudent. More specifically, we would like to answer two questions, 1) How well does SimStudent behavior match real student behavior? 2) If the proposed model is a good simulation of real students, what insights could we gain from it? To address the first question, we will use real student data gathered from 71 high school students used Carnegie Learning Algebra I Tutor. Based on the data, we will divide student into slow and fast learners, and record their learning curves. We will also implement multiple versions of the synthetic students with different features embedded. Then, we will train the synthetic students with the same set of problems used on real students, and record the learning curves as well as the errors made over time. We will then match the recorded behavior of the synthetic students with the real student data, and check which version of the model corresponds to the fast student learners, whereas which version of the model behaves similar to the slow student learners. If the model matches appropriately with real student data, it suggests that the proposed model is a good model of real students.

As for the second question, if the extended SimStudent is a good model, we would be able to carry out controlled simulation study with the extended SimStudent in understanding what causes the difference of real students based on the model variations. We will compare and contrast the different features used in the fast synthetic students and the slow synthetic students. With this simulation study, we hope to isolate factors that cause some students to learn faster than others.

Related Work

The objective of this paper is to propose a computational model that demonstrates how deep feature acquisition could yield faster future learning. While there has been considerable amount of work on accelerated future learning (Bransford and Schwartz 1999; Hausmann and VanLehn 2007; Pavlik *et al.* 2008), these projects have focused on demonstrating how instructional treatments lead to accelerated future learning. To the best of our knowledge, there has been no research on modeling how accelerated future learning is generated.

More generally, there has been considerable work on learner modeling. Previous research on competitive chunking (Servan-schreiber and Anderson 1990) models how grammatical knowledge of chunks were acquired, but was not focused on accelerated future learning. Orbán et al. (Orbán *et al.*) investigated a chunking mechanism in humans with a visual pattern-learning paradigm, and developed a Bayesian chunk learner. Research on automatic bug library construction (Suarez and Sison 2008; Baffes and Mooney 1996) uses machine learning techniques to capture student errors. But again, these projects did not attempt to model faster future learning.

Conclusion

In this paper, we presented a computational model of student learning that acquires deep features to assist future learning, and proposed a plan for integrating this model into a machine learning agent. We provided a preliminary evaluation of the proposed model. The preliminary results showed how both stronger prior knowledge and a better learning strategy in deep feature acquisition can assist future learning. Results also indicated that stronger prior knowledge produces faster learning outcomes compared with a better learning strategy. Some model variations were able to generate human-like errors, while others learn more quickly than students do. Last, we discussed the set of more extensive studies we could carry out once the integration is done.

Acknowledgments: This study is supported by National Science Foundation Award No. DRL-0910176 and by Department of Education (IES) Award No. R305A090519. This work is also supported in part by the Pittsburgh Science of Learning Center, which is funded by the National Science Foundation Award No. SBE-0836012.

References

Paul Baffes and Raymond Mooney. Refinement-based student modeling and automated bug library construction. *J. Artif. Intell. Educ.*, 7(1):75–116, 1996.

Julie L. Booth, Kenneth R. Koedinger, and Robert S. Siegler. The effect of prior conceptual knowledge on procedural performance and learning in algebra. In *Poster presented at the 29th annual meeting of the Cognitive Science Society*, Nashville, TN, 2007.

John D. Bransford and Daniel L. Schwartz. Rethinking transfer: A simple proposal with multiple implications. *Review of Research in Education*, 24:61–100, 1999.

Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2):121–152, June 1981.

Robert G.M. Hausmann and Kurt VanLehn. Explaining self-explaining: A contrast between content and generation. *Artificial intelligence in education: Building technology rich learning contexts that work*, 158:417–424, 2007.

Tessa Lau and Daniel S. Weld. Programming by demonstration: An inductive learning formulation. In *Proceedings of the 1999 international conference on intelligence user interfaces*, pages 145–152, 1998.

Nan Li, Subbarao Kambhampati, and Sungwook Yoon. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009.

Noboru Matsuda, Andrew Lee, William W. Cohen, and Kenneth R. Koedinger. A computational model of how learner errors arise from weak prior knowledge. In *Proceedings of Conference of the Cognitive Science Society*, 2009.

Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

Gergö Orbán, József Fisher, Richard N. Alin, and Máté Lengye. Baysian learning of visual chunks by human observers. In *Proceedings of the National Academy of Sciences, 105*, pages 2745–2750.

Philip Pavlik, Jr., Thomas Bolster, Sue-Mei Wu, Ken Koedinger, and Brian Macwhinney. Using optimally selected drill practice to train basic facts. In *ITS '08: Proceedings of the 9th international conference on Intelligent Tutoring Systems*, pages 593–602, Berlin, Heidelberg, 2008. Springer-Verlag.

Emile Servan-schreiber and John R. Anderson. Learning artificial grammars with competitive chunking. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16:592–608, 1990.

Merlin Suarez and Raymund Sison. Automatic construction of a bug library for object-oriented novice java programmer errors. In *ITS '08: Proceedings of the 9th international conference on Intelligent Tutoring Systems*, pages 184–193, Berlin, Heidelberg, 2008. Springer-Verlag.