Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2008; accepted). Why Tutored Problem Solving may be better than Example Study: Theoretical Implications from a Simulated-Student Study. In Proceedings of the International Conference on Intelligent Tutoring Systems

Why Tutored Problem Solving may be better than Example Study: Theoretical Implications from a Simulated-Student Study

Noboru Matsuda[1], William W. Cohen[2], Jonathan Sewall[1], Gustavo Lacerda[2], and Kenneth R. Koedinger[1]

[1]Human-Computer Interaction Institute
[2]Machine Learning Department
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh PA 15217 USA
[mazda, wcohen, sewall, gusl, koedinger]@cs.cmu.edu

Abstract: Is learning by solving problems better than learning from worked-out examples? Using a machine-learning program that learns cognitive skills from examples or by being taught, we have conducted a study to compare three learning strategies: learning by solving problems with feedback and hints from a tutor, learning by generalizing worked-out examples exhaustively, and learning by generalizing worked-out examples only for the skills that need to be generalized. The results showed that learning by tutored problem solving outperformed other learning strategies on the test scores that were measured on each problem solving step as the average ratio of the correct to incorrect rule applications. The advantage of tutored problem solving was mostly due to the error detection and correction that was available only when skills were applied incorrectly. The current study also suggested that learning certain kinds of conditions to apply rules only for appropriate situations is quite difficult. That is, learning how to perform mathematically valid operations is easier than learning *when* to apply rules.

1 Introduction

SimStudent is a machine-learning agent that learns cognitive skills by generalizing solutions demonstrated [1] and also by being tutored as we describe in this paper. Our original motivation to develop SimStudent was to automate cognitive modeling to author a Cognitive Tutor that deploys *model tracing* to provide individualized feedback and contextualized help [2]. To perform model tracing, the Cognitive Tutor needs a *cognitive model* that represents domain principles. However, cognitive modeling is a labor-intensive task that requires significant knowledge and experience in cognitive task analysis and AI-programming. Embedded into Cognitive Tutor Authoring Tools (CTAT [3]), SimStudent acts as an intelligent building block that allows authors to perform *authoring by demonstration*, where authors merely demonstrate how to solve problems (correctly and incorrectly) instead of writing a cognitive model by hand. SimStudent generalizes demonstrations and create a set of production rules that reproduce the problem-solving steps demonstrated.

A critical research question addressed in this paper is about the efficiency of SimStudent: How can SimStudent be taught most effectively?

Originally, SimStudent was a "*passive*" learner in the sense that SimStudent attempted to generalize every problem-solving step demonstrated, but did not attempt to perform problem-solving steps on its own. SimStudent could reduce the learning load by *selectively choosing* certain steps to generalize; for instance, generalizing a step only when SimStudent does not have a production rule that reproduces the step demonstrated. Assuming that applying an existing skill is easier than learning a new skill, this learning strategy might require a relatively shorter learning time to achieve the same quality of cognitive model. A third possibility is that SimStudent could actively solve problems, rather than explaining demonstrations, and get feedback. Since the author will see SimStudent performing actions, which provides a chance to explicitly correct errors, this tutoring strategy might outperform passive or the selective learning strategies.

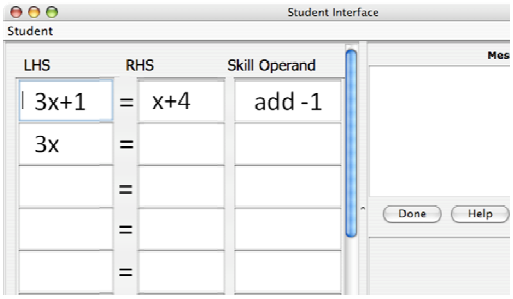

Fig. 1. The Tutor Interface for the Algebra I CTAT Tutor. Students are supposed to enter an operation for transformation first in the column labeled as "Skill Operand." Then corresponding expressions for the left- and right-hand sides must be entered.

In this paper, we compare three learning strategies to answer the following research question: *Which learning strategy is better in terms of efficiency of training and quality of resulting cognitive models?* Answering this question is not only important for authoring purposes, but it may also provide us theoretical insights into understanding human learning by inspecting SimStudent's learning processes and learning outcomes, which are not easily attainable in human subjects.

2 SimStudent: A machine-learning agent you can teach

An actual image of the Cognitive Tutor used in the current study is shown in **Fig. 1**. Suppose that an author is trying to build a Cognitive Tutor for Algebra equation solving. The author has just built the Tutor interface shown in **Fig. 1** by using CTAT. Now, the author launched SimStudent to create a cognitive model for equation solving by using the Tutor interface and solves a few problems.

2.1 An Example Cognitive Tutor: Algebra Equation Tutor

In this tutor, equations are represented with a mathematical operation to transform a given equation to another form. To transform an equation, an operation must be specified first, followed by the left-hand and right-hand sides of the resultant equation being entered in the adjacent row. **Fig. 1** shows that the author has decided to "add -1" to both sides, and the left-hand side has just been entered. In sum, a single equation-solving step (e.g., transforming "`3x+1=x+4`" into "`3x=x+3`") is modeled as *three*

steps – (1) selecting an operation for transformation, (2) entering an expression for the left-hand side, and (3) for the right-hand side. The first step is called *transformation step*, and the last two steps are called *type-in steps*. In this paper, the word "step" means one of these three steps. An operation for transformation must be specified prior to entering any expressions. The order of entering sides can be arbitrary, but both sides must be entered before selecting the next operation. The skills to select an appropriate operation are called *transformation skills*, and the skills to enter left- and right-hand sides are called *type-in skills*.

2.2 Learning Production Rules by Demonstration

Each of the production rules represents an individual skill to perform a particular step. [1] Performing a step is modeled as generating a tuple that consists of an *action* taken (e.g., "entering some text"), a place that was *selected* to take the action (e.g., "the second cell in the first column"), and the value that was *input* as a result of taking the action (e.g., the string "*3x*"). Those are called *action*, *selection*, and *input*. A tuple of <selection, action, input> is called an *SAI tuple*.

A production rule models a particular skill in terms of *what*, *when*, and *how* to generate a particular SAI tuple. In other words, a production rule shows that "To perform a step, first look at X and see if a condition Y holds. If so then do Z." The part of the production rule representing X (*what*) is called the *focus of attention* that specifies particular elements with certain constraints like "the cell in the table" shown in the Tutor interface. The part of the production rule representing Y (*when*) is called the *feature tests*. The feature tests represent a set of conditions that must hold about the focus of attention – e.g., the two cells must be in the same row, the expression in the cell must be polynomial, etc. Together, the focus of attention and the feature tests compose the left-hand side (i.e., the condition part) of a production rule. The right-hand side (i.e., the action part) of a production rule contains a sequence of *operations* that generates the value of the input in the SAI tuple.

Prior to learning, SimStudent is given a hierarchical structure of the elements in the Tutor interface with which to express the constraints among the focus of attention, a set of *feature predicates* with which to express feature tests, and a set of *operators* with which to compose a sequence of operations. SimStudent has a library of feature predicates and operators that are general for arithmetic and algebra, but the authors might need to write domain-specific background knowledge to use SimStudent for other domains.

When demonstrating a step, the author first needs to specify the focus of attention by double-clicking the elements on the Tutor interface. Then he/she performs a step, namely, takes an "action" upon a "selection" with an appropriate "input" value. Finally, the author needs to label the demonstrated step. This label is called the *skill name*.

When a step is demonstrated for a particular skill K with a focus of attention F and an SAI tuple T, the pair <F, T> becomes a *positive example* of the skill K. The pair <F, T> also becomes a *negative example* for all other skills. This indicates to "apply skill K to carry out the SAI tuple T when you see the focus of attention F, but do not

[1] Generally speaking, authors sometimes model a single observable step as a *chain of production-rule applications*. However, SimStudent generates a single production rule per observable problem-solving step.

apply any skills other than K when you see F." We call this kind of negative examples the *implicit negative examples* as opposed to the explicit negative examples used for tutored problem solving, which is described in the next section. Once a positive example is acquired, it stays as positive throughout a learning session. On the other hand, a negative example for a skill would later become a positive example if the same focus of attention is eventually used to demonstrate that skill.

When a new positive or negative example is added for a particular skill, SimStudent *learns* the skill by generalizing and/or specializing the production rule for the skill so that it applies to all positive examples and does not apply to any negative examples. The focus of attention is generalized so that they are consistent with all instances of the focus of attention appearing in the positive examples. An example generalization is to shift from "first column" to "any column." Feature tests are generalized and/or specialized so that they cover all positive examples and no negative examples that is done by Inductive Logic Programming [4] in the form of Foil [5]. The operator sequence is generalized so that it generates "input" values from the focus of attention for all SAI tuples in the positive examples.

2.3 Learning Strategies

The original version of SimStudent always learns skills whenever a step is demonstrated by generalizing existing skills or introducing a new skill. This can be seen as a model of human students diligently learning skills from worked-out examples, regardless of what they already can do (although it sounds too idealistic).

As an interesting twist (and a step towards a more realistic model), the author can also have SimStudent try to "*explain*" the step demonstrated, by identifying a previously learned skill that replicates the step demonstrated, and having SimStudent learn skills only when it fails to explain the step. This is analogous to human students learning from worked-out examples while self-explaining the solutions.

Furthermore, the author can instead *tutor* SimStudent on how to solve problems. The author provides problems to SimStudent, lets SimStudent solve them, and provides feedback on each of the attempts made. When SimStudent makes an error, the author can provide negative feedback, which will motivate SimStudent to accumulate a negative example *explicitly* – i.e, it will learn when *not* to apply a skill because it produces an incorrect output. When SimStudent has no rules indicating how to perform a step, the author provides a "hint" on what to do next; this hint is just a demonstration of how to perform the step. This is a model of learning by tutored problem solving.

In sum, we implemented these three learning strategies for SimStudent:

Diligent Learning – provides demonstrations on every step and SimStudent learns skills each time a step is demonstrated.

Example Study – provides demonstrations on every step and SimStudent attempts to identify a production rule that reproduces the step demonstrated. Only when the attempt fails, does SimStudent learn skills.

Tutored Problem Solving – provides SimStudent with problems to solve. For each step, SimStudent is asked to show *all rule applications* that can be done. For each of the rule applications, SimStudent gets flagged feedback from an *oracle*, which merely tells the correctness of the rule application. Correct rule applications become positive examples and incorrect ones become negative examples.

When there is no correct rule application for a step, SimStudent asks a what-to-do-next hint to the oracle. The oracle then demonstrates to SimStudent how to perform the step.

The oracle for the Tutored Problem Solving can be either a human or another computer program. In the current study, we used the commercially available Cognitive Tutor, Carnegie Learning Algebra I Tutor, as the oracle. The details follow.

3 Learning Strategy Study

This section describes a study conducted to evaluate the efficiency of each of the three learning strategies described in section 2.3.

3.1 Method

Three versions of SimStudent were implemented – one for each of the three learning strategies. Each SimStudent was trained with 20 problems and tested with ten problems. Since hundreds of steps must be demonstrated and tested to complete the study, it was not realistic to ask human authors to be involved in the study. Instead, we used pre-recorded and machine-generated demonstrations as described below.

The pre-recorded demonstrations were collected from a previous classroom study conducted in the PSLC LearnLab.[2] In the LearnLab study, the Carnegie Learning Algebra I Tutor was used in an urban high-school algebra class. The high-school students were asked to use the Algebra I Tutor individually. The students' activities were logged and stored into a large database, called DataShop.[3] We then extracted problems and human students' *correct* steps from DataShop for the current study. An entire (correct) solution for a particular problem made by a particular student became a single training problem for the Example Study condition and the Diligent Learning condition. The problems were randomly selected from the DataShop data.

For the Tutored Problem Solving condition, the steps were demonstrated by the Carnegie Learning Algebra I tutor. That is, when SimStudent got stuck, SimStudent asked a what-to-do-next hint to the Carnegie Learning Algebra I tutor, and the Carnegie Learning Tutor provided a precise instruction for what to do in the form of the *bottom-out hint*, which provides the same information as the SAI tuple. Whenever SimStudent performed a step, each of the rule applications was sent to the Carnegie Learning Algebra I Tutor to get a flagged feedback.

There were five sets of training problems. Thus, there was a total of 15 experimental sessions (five training sets for each of the three learning-strategy conditions).

Each time SimStudent was trained on a new training problem, the production rules learned were tested with the ten test problems. The same set of test problems was used for all of the 15 experimental sessions. The test problems were also randomly collected from the LearnLab study. For each of the steps in a test problem, we asked SimStudent which production rules can be fired. Since we wanted to know how *poorly* SimStudent solves problems in addition to how well, we recorded all possible rule applications for each step. More precisely speaking, for each step, we enumerated all

[2] www.learnlab.org
[3] www.learnlab.org/technologies/datashop

production rules whose left-hand conditions hold. The correctness of a rule application was evaluated by the Carnegie Learning Algebra I Tutor. The steps performed by SimStudent were coded as *correct* if there was at least one correct rule application attempted. Otherwise, the steps were coded as *missed*.

3.2 Evaluation Metrics

We define a dependent variable, called the *Step score*, that represents how well the production rules learned were applied on individual steps in the test problems. A step is scored as zero if it was missed (i.e., no correct rule application was made – see the definition above). Otherwise, a step was scored as a ratio of the number of correct rule applications to the total number of rule applications applicable to that particular step. For example, if there were 2 correct and 6 incorrect rule applications for the step, then the Step score for that step is 0.25. The step score ranges from 0 (no correct rules applicable) to 1 (no incorrect rules applicable, and at least one correct rule applies). We define the *Problem score* as the average *Step score* for all steps in a test problem.

In general, there are several correct and incorrect rule applications available for each step. Since SimStudent does not have any strategy to select a single rule among these conflicting rule applications, the Step score can be seen as a probability that the step is performed correctly at the first attempt.

4 Results

4.1 Overall Learning Performance

Fig. 2 shows average Problem Score for each learning-strategy condition. The X-axis shows the number of training problems learned. The Problem score was aggregated across the ten test problems and the five training sets (i.e., average of the 50 Problem scores for each condition). All three conditions showed an overall improvement on the Problem score when more training problems were learned.

The three learning conditions improved equally on the first 8 problems. After that,

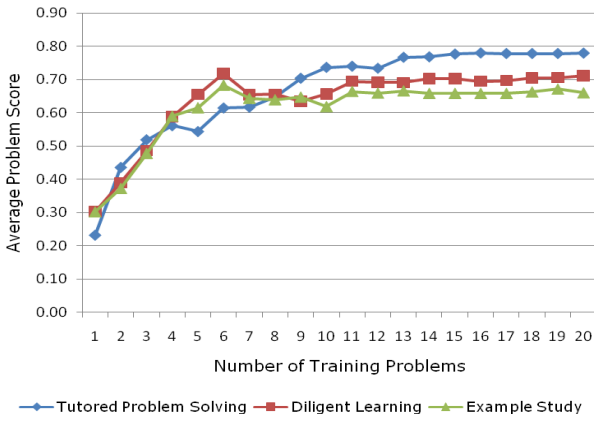

Fig. 2. Overall improvement of the Problem scores. The X-axis shows the number of training problems. The Y-axis shows the average Problem scores on the ten test problems, aggregated across five training sets.

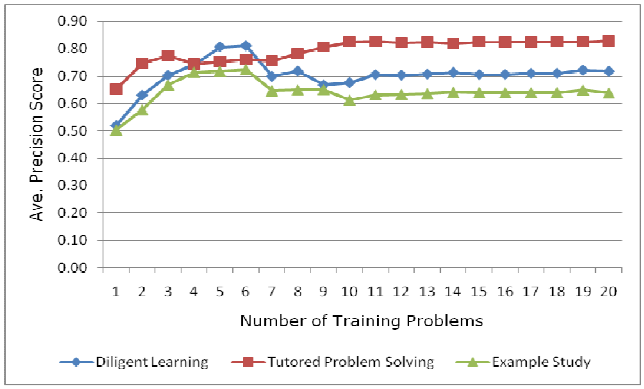

Fig. 3. Average Precision scores. The X-axis shows the number of training problems learned by the time the Precision score was measured.

the Tutored Problem Solving condition outperformed other conditions. There was a point, for all three conditions, where the improvement of the performance on the test problems diminished to almost nothing. After training on all 20 problems, the average Problem score was 0.78 for the Tutored Problem Solving, 0.72 for the Diligent Learning, and 0.66 for the Example Study. ANOVA revealed a main effect of the learning strategy; F=7.68, p<0.001. The paired t-tests showed that all three learning-strategy conditions are significantly different from each other. *The Tutored Problems-Solving condition outperformed the other two conditions on the Problem score. The Example Study was the least efficient learning strategy in terms of the Problem score.*

To further investigate why the Tutored Problem Solving condition led to better learning, we broke down the *Step* score (the basis of the Problem score) into two scores: (1) the *Precision* score showing the ratio of the number of correct to incorrect rule applications for a step, and (2) the *Recall* score showing the ratio of the number of steps that were performed correctly to the total number of steps in a test problem.

A high Precision score does not necessarily predict a high Recall score. The Precision score is not blamed for the missed steps (which by definition, are the steps where no correct rule application was made), and it gets credit for the correct rule applications, no matter how poorly other steps were missed. Similarly, a high Recall score does not guarantee a high Precision score. The Recall score does not take into account incorrect rule applications – it gets credit if a step is eventually performed correctly, no matter how poorly incorrect rule applications were made.

Fig. 3 shows the average Precision score for the ten test problems aggregated across the training sets. On the 20th training problem, there was a main effect of the learning strategy; F=24.49, p<0.001. The paired t-tests confirmed that all three conditions are significantly different from each other. The Tutored Problem Solving condition outperformed other conditions on the Precision score. This means that *the production rules learned by Tutored Problem Solving were more likely to produce correct rule applications than the rules learned by other learning strategies*.

Fig. 4 shows the average Recall score. ANOVA showed a main effect of the learning condition; F=7.68, p<0.001. The paired t-tests showed that the Tutored Problem Solving was significantly inferior to the other two conditions (both t=2.01, p<0.001), but the difference between Example Study and Diligent Learning was not significant. The Tutored Problem Solving condition was significantly inferior to other two condi-

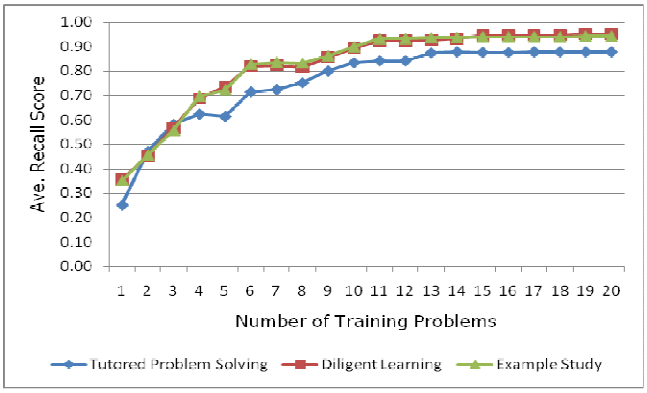

Fig. 4. Average Recall score. The X-axis shows the number of training problems.

tions, meaning *the Tutored Problem Solving condition did not learn as many production rules necessary to solve test problems as other conditions did*. On average, the Tutored Problem Solving condition learned the fewest production rules (11.6), and the Greedy condition learned the most (21.0). The Example Study condition learned 16.0 production rules on average.

4.2 Types of Errors

To see if there were any differences in the kinds of errors made by each learning condition, we categorized the errors appeared on the test problems. Regardless of the learning strategy, once the learning was saturated (i.e., after learning ten problems for Diligent Learning and Example Study, and 13 problems for Tutored Problem Solving), there were only two types of errors: (1) *Step-Skipping* error – attempting to apply a transformation skill without completing previous type-in steps, (2) *No-Progress* error – applying a transformation skill that does not make the transformed equation any closer to a solution (see section 2.1 for the definition of steps and skills).

An example of a Step-Skipping error is to apply another transformation skill to the situation shown in **Fig. 1**, and enter, say, "divide 3" into the rightmost cell on the second row when the middle cell (right-hand side of the equation) is left blank.

An example of a No-Progress error is to "subtract 2x" from `2x+3=5`. This is a mathematically valid step, but it does not make the resultant equation any closer to a solution. Thus, it is considered as a wrong step by the Algebra I Tutor.

No-Progress errors appeared in all three conditions. Quite interestingly, there were no Step-Skipping errors observed for the Tutored Problem Solving condition. Why? We hypothesized that only Tutored Problem Solving had a chance to revise incorrect skills during training, by making a Step-Skipping error and receiving negative feedback, which allowed SimStudent to accumulate a negative example to correctly learn LHS conditions. Namely, *making an explicit error and getting a flagged feedback on it (which by definition, merely tells the correctness of the step) should have positively contributed to learning*. To test this hypothesis, we controlled the creation of negative examples for the Tutored Problem Solving condition, which is described in the next section.

4.3 Control Experiment with No Explicit Negative Feedback

We have modified the Tutored Problem Solving condition, so that it does not generate negative examples for incorrect rule applications. SimStudent still received negative *feedback* for incorrect rule applications, thus another attempt was made to perform a step. This means that the modified version of Tutored Problem Solving still had the same amount of positive examples during training as the original version.

With this modification, the Tutored Problem Solving condition made the same Step-Skipping errors as the other conditions. Thus, *it was the* explicit *negative examples obtained by incorrect rule applications that caused the high Precision score for the Tutored Problem Solving condition*.

This modification did not affect the appearance of the No-Progress errors – having more negative examples did not prevent skills from being incorrectly generalized and making No-Progress errors.

5 Discussion

5.1 The Impact of Negative Feedback on Learning

The most important finding in the current study is that *the most effective way to train SimStudent is Tutored Problem Solving*. *Making errors is crucial for successful learning*. Allowing SimStudent to commit itself to apply its own skills to solve problems and giving it *negative feedback explicitly* to appropriately generalize incorrect skills leads to better learning.

It is interesting to see that Example Study and Diligent Learning are superior to Tutored Problem Solving at some point on the fifth and sixth training problems (**Fig. 2**). This was mostly due to the high Recall scores – Example Study and Diligent Learning tend to learn more rules that correctly perform steps. However, at the same time, they also have a tendency to learn incorrect rules as well. Those incorrect rules can only be eliminated through explicit negative feedback.

5.2 Difficulty in Rule Induction

Another important lesson learned is *the difficulty of inductive learning*. It turned out that learning appropriately generalized rules that do not generate No-Progress errors is challenging. Despite having explicit negative feedback on the No-Progress errors during training, the Tutored Problem Solving condition still made the No-Progress errors on the test problems.

Since No-Progress errors always generate mathematically valid steps (meaning, the RHS operator sequence is correct), the challenge is in learning LHS conditions – *learning* when *to apply a particular rule is more difficult than learning* how *to perform a step*. Since it is beyond the scope of the current paper, we do not further discuss this issue, but now *we have narrowed down the difficulty of inductive learning to learning conditions for when to apply rules*. This must be addressed further in future studies.

6 Conclusion

The empirical study showed that tutored problem-solving results in learning production rules more accurately than learning from examples. Thus, for authoring purposes, *tutoring* SimStudent instead of *demonstrating* solutions may be a better form of using SimStudent as an aid to author Cognitive Tutors, assuming that providing feedback does not cost too much for the authors. In the 20 training problems, each skill was *demonstrated* 13.5 times on average for Diligent Learning, 3.6 times for Example Learning, and 2.8 times for Tutored Problem Solving. For the Tutored Problem Solving, the tutor provided positive feedback 14.1 times and negative feedback 3.5 times on average throughout the 20 training problems. Future studies on the authoring cost analysis are necessary.

That tutored problem solving is significantly inferior to other learning conditions on the Recall score must be studied further. What about starting from the example study first and shifting to tutored problem solving later? This is a well-known learning strategy that is effective for human students [6]. All three conditions tied on the Step score for the first few training problems, and still the example study conditions were better on the Recall score on those steps. Thus, starting from an example study would allow SimStudent to acquire production rules more quickly, and switching to tutored problem-solving would provide good opportunities to correct these rules.

We also demonstrated that simulation of human learning helps us to advance knowledge in human learning. It also provides insight into future studies on inductive learning. Finding out why some features are more difficult to learn than others would open the door for future studies on human and machine learning.

References:

1. Matsuda, N., W.W. Cohen, and K.R. Koedinger, *Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors*, in *AAAI Workshop on Human Comprehensible Machine Learning (Technical Report WS-05-04)*. 2005, AAAI association: Menlo Park, CA. p. 1-8.
2. Koedinger, K.R. and A. Corbett, *Cognitive Tutors: Technology Bringing Learning Sciences to the Classroom*, in *The Cambridge Handbook of the Learning Sciences*, R.K. Sawyer, Editor. 2006, Cambridge University Press: New York, NY. p. 61-78.
3. Koedinger, K.R., V.A.W.M.M. Aleven, and N. Heffernan, *Toward a Rapid Development Environment for Cognitive Tutors*, in *Proceedings of the International Conference on Artificial Intelligence in Education*, U. Hoppe, F. Verdejo, and J. Kay, Editors. 2003, IOS Press: Amsterdam. p. 455-457.
4. Muggleton, S. and L. de Raedt, *Inductive Logic Programming: Theory and methods.* Journal of Logic Programming, 1994. **19-20**(Supplement 1): p. 629-679.
5. Quinlan, J.R., *Learning Logical Definitions from Relations.* Machine Learning, 1990. **5**(3): p. 239-266.
6. Renkl, A., et al., *From example study to problem solving: Smooth transitions help learning.* Journal of Experimental Education, 2002. **70**(4): p. 293-315.