What characterizes a better demonstration for cognitive modeling by demonstration? [*]

Noboru Matsuda, William W. Cohen, Ken Koedinger
School of Computer Science
Carnegie Mellon University, Pittsburgh, PA

Abstract

A *simulated student* is a machine learning agent that learns a set of cognitive skills by observing solutions demonstrated by human experts. The learned cognitive skills are converted into a *cognitive model* for a *Cognitive Tutor* that is a computerized tutor that teaches human students the cognitive skills. In this paper, we analyze the characteristics of the effective demonstrations that lead to quicker and more accurate learning. Results from empirical studies show that expressive demonstrations (as opposed to abbreviated demonstrations that involve implicit mental operations) are better for both speed and accuracy of learning. We also found that providing multiple demonstrations of the same cognitive skill with differing surface features accelerates learning. These findings imply that the ordering of training sequence as well as the level of detail in demonstration determines the efficiency with which a simulated student generates a cognitive model.

1 Introduction

This study investigates characteristics of effective demonstrations for machine-learning agents. In particular, we consider *Simulated Students*, which learn a *cognitive model* by demonstration [Cypher, 1993].

A cognitive model is the essential component for a *Cognitive Tutor* [Anderson *et al.*, 1990; Anderson *et al.*, 1995] since it is used to *model trace* (i.e., to evaluate) a human student's problem-solving steps. Traditionally, building a cognitive model has required significant skills in cognitive task analysis and AI programming, so that novice authors have great difficulty to create them without help.

Developed as a plug-in intelligent component for the Cognitive Tutor Authoring Tools [Koedinger *et al.*, 2003], our Simulated Student learns cognitive skills from the model solutions. That is, with the assistance of a Simulated Student, an author can simply demonstrate solutions without prior knowledge of programming. The Simulated Student then generates a set of *production rules* that becomes a cognitive model that replicates the demonstration. Because this model is closely related to the model developed by the human student that will use the Cognitive Tutor, we call this learning agent a *Simulated Student*.

Several researchers have used machine learning for instructional design, for example, [Mertz, 1997; Tecuci & Keeling, 1999]. Others have applied machine learning specifically for cognitive modeling for Cognitive Tutors [Blessing, 1997; Jarvis *et al.*, 2004]. Our approach generates a cognitive model that is more accurate and more general than the previous studies, however, because it captures meaningful features of the demonstration (details in 3.3).

An interesting question is then what makes demonstration more effective and efficient. Specifically, we have explored how the following parameters alter the effectiveness of learning: the amount of information demonstrated, the type of problems, and the order of problems demonstrated. We call those problems used for demonstration the *training problems*.

One way to evaluate effectiveness is to measure the amount of work necessary to obtain a quality cognitive model. We have operationalized the amount of work as the number of steps demonstrated, and the quality of the cognitive model as the accuracy of model tracing. We have used equation solving as an example domain for the study.

Our major findings: (1) providing an expressive demonstration (as opposed to an abbreviated demonstration, which skips steps) is more likely to avoid learning incorrect production rules when the demonstration bears a coincidental ambiguity in the problem representation (e.g., the number '3' in $3x=9$ resulting in $x=3$, which could be a coefficient or a quotient); (2) providing an expressive demonstration, however, does not prevent incorrect learning when the demonstration has ambiguity in structural features (e.g., '-3' in $2x+4x-3$ could be a last term or a constant term); (3) the order of training problems has a significant impact on the efficiency of learning – better learning involves ordering the training problems in such a way that, for each production rule, the instances of rule applications appear differently at a surface features (e.g., applying the same production rule to an equation with and without a fraction term).

These results imply that to best facilitate authoring with the Simulated Student, the author should provide a demonstration with detailed steps. It also implies that there is an

[*].

"optimal" sequence of training problems that generates a better cognitive model with fewer demonstrations.

The structure of the paper is as follows. In section 2, we describe the basic architecture of the Cognitive Tutor and its authoring tools, the Cognitive Tutor Authoring Tools. We then introduce the Simulated Student in section 3. Details of two evaluation studies as well as the lessons learned are described in sections 4 and 5, followed by concluding remarks in section 6.

2 Cognitive Tutors and CTAT

2.1 Cognitive Tutors and Example-Tracing Tutors

Cognitive Tutors are an effective [Anderson *et al.*, 1995] and widely-used [Koedinger *et al.*, 1997] type of computerized tutor. A Cognitive Tutor's teaching strategy is as follows: ask (human) students to solve problems; verify the correctness of each step they make; and provide feedback on each step. Feedback tells the student whether a step is correct or not. When the step is incorrect, the tutor may also provide more informative feedback – for example, the tutor explains why the step is wrong or what should have been done instead. If the student can not make a correct step, then he or she may ask for a hint. The tutor then provides a hint message associated with the next correct step.

This teaching strategy is supported thoroughly by a cognitive model, which is a set of production rules. Each production rule represents either a correct or incorrect step that could be performed by a student.[1] When a student performs a step, the tutor tries to find a production rule that matches the step in order to judge its correctness.[2] This procedure is called *model tracing*. The tutor requires the students to perform each step correctly before allowing the student to proceed to the next step. This way, the tutor ensures that it (the tutor) follows the student's path toward a solution.

One can build a simplified version of a Cognitive Tutor by relaxing a requirement on model tracing. Instead of having a fully functional model tracing capability with a fully general cognitive model, the automated tutor would recognize student steps only in the specific problems demonstrated by an author. The Cognitive Tutor Authoring Tools (CTAT) is a suite of software that allows authors to build this kind of tutor, called an *Example-Tracing Tutor* [Koedinger et al., 2003].

2.2 Cognitive Tutor Authoring Tools (CTAT)

CTAT comprises three major components: an external graphical user interface (GUI) builder, an integrated development environment to build and test a Cognitive Tutor, and an external editor to manually build a cognitive model.

With CTAT, the author first builds a GUI for the desired tutor that exposes the individual steps in the solution. The author then uses the tutor's GUI and demonstrates correct and incorrect actions for many problems. A CTAT tool called the *Behavior Recorder* embedded into CTAT stores and displays the author's demonstrations. It also allows the author to mark each of the steps as correct or incorrect and add hint messages so that when human students use the Example-Tracing Tutor, the tutor can provide appropriate feedback on the students' steps.

To make the Example-Tracing Tutor a fully-equipped Cognitive Tutor that can model trace human student's steps on more general classes of problems, the author must build a cognitive model by hand. CTAT has tools to help the author with this task, but it is still quite challenging for those authors who are not trained, hence the motivation for building the Simulated Students. Our Simulated Students addresses this difficulty by extending CTAT to enable the Cognitive Tutor to model-trace generalizations of the author's demonstrated solutions.

The next section shows how the simulated students can automatically build a cognitive model from demonstrations.

3 The Simulated Student

We will use the Equation Tutor shown in Figure 1 as an illustration. An equation is represented by two cells in separate single-column tables. Given an equation in the top row of the tables, human students are expected to fill in a single cell at a time. For example, at the moment shown in Figure 1, entering 3*x*+4–4 in the second cell in the left column is the desired next step, followed by entering –5–*x*–4 in the second cell in the right column.

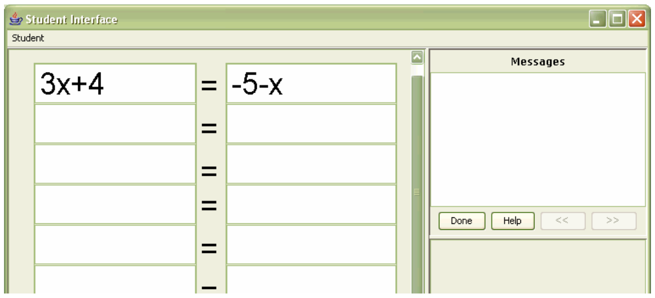

Figure 1: The Equation Tutor

In the rest of this section, we first describe how the author demonstrates problem-solving steps for the Simulated Student. We then explain the basic structure of the production rules learned by the Simulated Student. Finally, we discuss the learning algorithm implemented in the Simulated Student.

3.1 Authoring cognitive model by demonstration

The Simulated Student builds a cognitive model incrementally and interactively. For each step in demonstration, the author must provide the following additional information:

[1] In general, a cognitive model could involve production rules that do not correspond to any visible steps. Those are the production rules that are used only for the internal chain of reasoning to gain efficiency in the system development, which is not relevant in the current study. Hence we do not deal with such kind of internal production rules in this study.

[2] The Cognitive Tutor assumes that the cognitive model is omniscient hence if there is no matching production rule, the step performed is incorrect.

1. All GUI elements (called the *focus of attention*) whose contents are to be used in performing the step must be indicated. This is done by double-clicking on each GUI element.
2. A name for the step (called a *skill name*) must be given. This name must be unique for each skill, and the same name must be given each time that skill is demonstrated.

When an author demonstrats a step, the Simulated Student first tries to model trace the step with the existing cognitive model (which is initially empty). Feedback on the attempt is then provided; that is, the author is informed which production rule was matched (if any) when model tracing the step just demonstrated. The author would then acknowledge the feedback by entering a correct skill name (in the case that a wrong production rule or no production rule matched), or by validating a correct skill name (in the case that the model trace was correct).

The Simulated Student learns or re-learns a single production rule for a step demonstrated only when model tracing fails. The cognitive model thus grows upon failure.[3]

3.2 Structure of a production rule

A production rule represents an action to be performed in terms of *what* kind of operations should be done upon *which* GUI element and *when*. The last two issues are relating to the *conditions* in the left-hand side (LHS) of the production rule, whereas the first issue is for the *operator sequence* (or the *actions*, if you will), in the right-hand side (RHS).

More precisely, the which-part represents particular GUI elements that are internally represented as *working memory elements* (WMEs). Every GUI element has a corresponding internal WME representation. The WMEs are semantically organized in a frame-like structure, represented as a semantic network. So, for example, in the tutor GUI shown in Figure 1, a problem has two tables, a table has a column, and a column has a list of cells. The which-path identifies a particular WME as a node in the semantic network – hence we call it the *WME-path*.

The when-part is represented as a list of the WME status hence called the *LHS feature constraints*. It captures the *features* shown in the GUI that must be held for the production rule to be fired. Some example features in the Equation Tutor are "an expression *is a polynomial*," "an expression *has a variable term* in it," etc.

The what-part consists of a *sequence of RHS operators* that must be applied to the specified WMEs to make an action for a step. An operator may have multiple input values but always returns a single output value. An output from an operator can be passed as an input for another operator. Therefore, an action for a single step can be decomposed as a sequence of simple operators. For example, a production rule to add -4 from the left-hand side of an equation (as the first step in adding -4 from both sides) could apply to the situation shown in Figure 1. The operator sequence for this action could be:

```
?last-term ← (get-last-term "3x+4")
?rev-term ← (reverse-sign ?last-term)
?new-exp ← (add "3x+4" ?rev-term)
```

3.3 The learning algorithm

The three components of the production rule – WME-path, LHS feature constraints, and the RHS operator sequence – are learned with the different learning techniques.

The WME-path consists only of focus of attention, which is explicitly specified by the author when a step is demonstrated. The learning for each WME-path is a straightforward generalization. In the most specific case, a WME-path represents an absolute object in the absolute position, e.g., the 3rd cell in the 2nd column of the 1st table. In the most general case, the WME-path represents any object in any places, e.g., any cell in any column in any table. A WME-path of intermediate generality has those only some of its elements specified, e.g., any cell in the 2nd column in the 1st table. The generality of a WME-path is predefined. On the first demonstration for a particular skill, the WME-path is simply a set of paths for each element of focus of attention. When more demonstrations are made for the same skill, then the WME-path is generalized along predefined generalities.

LHS features are learned by FOIL [Quinlan, 1990]. For each production rule, FOIL learns an applicability of the production rule in terms of the features that can be read off the GUI elements. When a step is demonstrated for a particular production rule, then the specified focus of attention is stored as a positive example of the applicability for that particular production rule. As for the relations appearing in the body of the learned concept, when a step is demonstrated, all of the feature predicates given a priori as background knowledge are tested against the focus of attention. The results become the positive and negative examples for those relations. Furthermore, an instance of demonstration on a particular production rule becomes a negative example for all other production rules demonstrated so far.

The learning for the RHS operator sequence is the most simple and straightforward. Given a set of instances of demonstration on a particular rule application, the Simulated Student tries to find the shortest sequence of operators that agrees with all instances in the sense that it always produces the action from the focus of attention for each of the demonstration instances. This is done by the iterative-deepening depth-first search.

The next two sections show empirical studies to evaluate the effectiveness of demonstrations to address the central research question of what defines a better demonstration.

4 Curriculum Evaluation Study

4.1 Purpose of the study

The primary purpose of the first study is to examine how differences in the sequence of training problems used in the demonstration affect learning. For human students, teaching

[3] This functionality can be turned off. A study comparing the timing of learning is shown in section 4.3.

"easy" problems prior to "hard" problems might make more sense, presumably because it imposes less cognitive load. What about a machine learner, which does not suffer from such a limitation of cognitive resources?

Note that in many cases, the author will be sequencing examples with a human student in mind. For purely engineering purposes, therefore, it is useful to understand the degree to which the sequence of examples needed by the simulated student coincides with our intuitions about human learning [VanLehn, 1987]. Also, if example ordering causes a significant difference in learning performance, we must identify the sequence of training problems that works best for the simulated student.

As mentioned above, the Simulated Student re-learns a production rule when a production rule turns out to be incorrect. More precisely, the Simulated Student first attempts to model trace a step demonstrated, and only when the model tracing fails is a corresponding production rule re-learned. In this study, we also explore the effect of forcing the Simulated Student to relearn rules on *every* instance of a rule application in a demonstration.

4.2 Method

There were eight training problems used for the study as shown in Table 1. There were ten production rules involved in the demonstration (shown in the top row in the table). The numbers in the table show the locations in the solution step where a corresponding rule application occurred. Double digits (e.g., 3 7) indicate that the corresponding rule was applied twice on the specified steps. The numbers in the margin show the total number of rule application made for each problem and rule. A total of 54 instances of rule application was demonstrated.

Problem	do-arith-lhs	do-arith-rhs	done	add-lhs	add-rhs	div-lhs	div-rhs	multi-lhs	multi-rhs	copy-rhs	
(x + 5)/6 = 7	3 7	4 8	9	5	6			1	2		9
x/4 + 5 = 8	3 7	4 8	9	1	2			5	6		9
4x + 5 = 13	3 7	4 8	9	1	2	5	6				9
3x + 4x = 21	1 5	6	7			3	4			2	7
-x = 5	3	4	5			1	2				5
x/7 = 6	3	4	5					1	2		5
8x = 16	3	4	5			1	2				5
x + 4 = 9	3	4	5	1	2						5
	12	11	8	4	4	4	4	3	3	1	54

Table 1: The training problems used in the studies

For validation of the learned production rules, seven test problems were used that were solved in 67 steps with the same ten production rules used in the demonstration. Each time a training problem had been completely demonstrated, a validation test was run over the seven test problems, and solution steps were model-traced. The accuracy of a production rule was measured as the ratio m/N where N is the total number of times the rule should be applied in the seven test problems, and m is the number of steps that were correctly model-traced.

There were ten feature predicates and 24 operators given as background knowledge.

4.3 Results and Discussion

Impact of the training sequence on the learning outcome

When the training problems are ordered according to the number of the production rules applied to solve a problem, the order in which training problems are demonstrated affects learning – ordering training problems from "difficult" problems (more production rules involved to solve a problem) to "easy" problems (fewer production rules) leads to better learning in terms of both speed and accuracy. Figure 2 shows two learning curves comparing the Hard-to-Easy (H2E) and Easy-to-Hard (E2H) conditions. The X-axis shows a number of times a particular rule application is demonstrated. The Y-axis is an accuracy of model tracing, aggregated and averaged for each of the learning occurrences across all production rules.

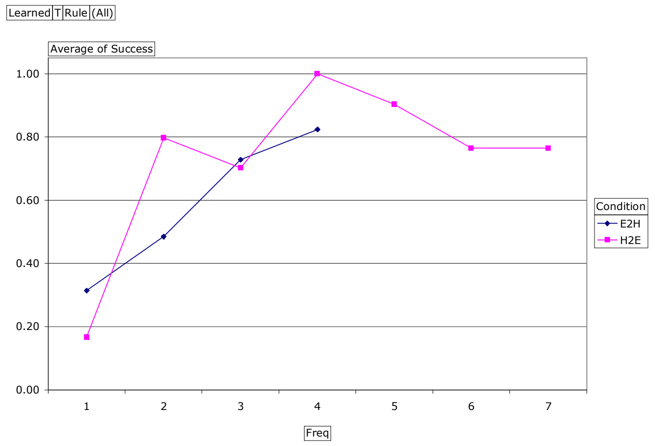

Figure 2: Comparing order of training problems

As shown in the figure, the Easy-to-Hard condition required considerably more training instances than learning hard problems first. Notice that learning occurred only when a production rule turned out to be wrong. Hence in the Easy-to-Hard conditions, the production rules learned were more likely to be wrong. At the end of the training sessions, however, there were no differences in the quality of the production rules in both conditions.[4]

Can we optimize the training problems?

To investigate more closely how the order of the training problems affects learning, we randomly ordered eight training problems and evaluated the learning outcomes in the same way described above. It turned out that for a particular production rule to be learned with a small number of demonstration instances, the training problems need to be ordered so that the demonstration instance for a particular production rule happened consecutively across the training problems where the rule applications differed at the WME-path level. In the equation-solving domain, the difference in the rule

[4] There was a technical glitch in the current implementation that affected the accuracy of model tracing for the current study. The drop at the end of the Hard-to-Easy condition was partly due to this problem.

application at the WME-path level corresponds to the difference in the place in the table where the rule is applied – e.g., the 1st line vs. the 3rd line.

The above observation implies that one can design an effective demonstration in terms of the sequence of the training problems that leads to a better learning outcome. To our surprise, we could successfully identify a sequence of only four training problems (shown in Table 2) that leads to a set of production rules with the equivalent quality in the accuracy of model tracing (Figure 3).

Type	do-arith-lhs	do-arith-rhs	done	add-lhs	add-rhs	div-lhs	div-rhs	multi-lhs	multi-rhs	copy-rhs
x/4 + 5 = 8	3 5	3 5	5	2	2			4	4	
(x + 5)/6 = 7	3 5	3 5	5	4	4			2	2	
3x + 4x = 21	2 4	4	4			3	3			2
-x = 5	3	3	3			2	2			

Table 2: A "better" sequence of training problems

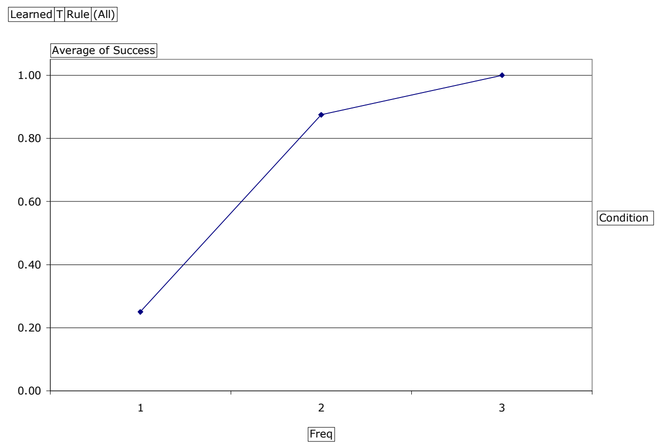

Figure 3: Learning curve with the "better" four training problems

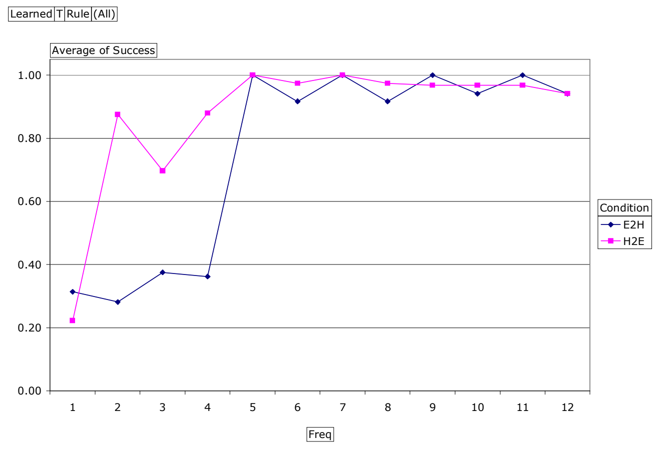

Figure 4: Learning curve when forced to relearn on each instance of the demonstration steps

Impact of the timing of re-learning

As shown in Figure 4, even when the Simulated Student was forced to re-learn rules on every step in the demonstration, there was not a significant difference in the performance of learning in terms of speed and accuracy – the learning converged within 5 demonstration instances. There was a notable drop at the beginning of the Easy-to-Hard condition, but it was due to a coincidental similarity in the rule applications demonstrated. As shown in the Table 1, the first 4 instances for the three rules – `do-arith-lhs`, `do-arith-rhs`, and `done` – were demonstrated in the exact same ways in terms of the surface similarity (they were applied in the exact same place in the solution steps).

In summary, the order of training problems has a significant effect that leads to better learning in terms of speed and accuracy. The timing of learning may not have any effect at all, so learning only on failure is sufficient for practical purposes.

5 Level of Detail in Demonstration

5.1 Purpose of the study

The purpose of the second study is to see if the difference in the level of detail in demonstration could affect learning. The level of detail is defined as the number of actions explicitly demonstrated.

5.2 Method

There were two different types of demonstrations used in the study:

1) The *abbreviated demonstration* showed only a result of an algebraic manipulation. In other words, simplification was done implicitly. For example, when dividing both sides of an equation with a number, the abbreviated demonstration showed only the result of division, say, $4x$=8 gets x=2.
2) The *expressive demonstration* showed intermediate steps or algebraic manipulations. For example, dividing the both sides of $4x$=8 with 4 got $4x/4$=8/4, which then simplified to x=2 in the next step.

The same procedure for the validation test and the same dependent variable to measure the accuracy of model tracing were used as in the first study. Also, the same set of feature predicates and operators were used.

5.3 Results and Discussions

It turned out that ambiguity in demonstration could lead to learning incorrect production rules. We have also found that changing the level of detail in demonstrations could resolve certain types of ambiguity problems. In this section, we first introduce the ambiguity problem, then discuss the effect of changing the level of detail in demonstration.

Ambiguity in demonstration

There are two kinds of *ambiguity* in a demonstration that could hinder learning.

Operator Ambiguity: When $3x$=9 leading to x=3 was demonstrated as the first instance of a rule to divide both side with the same number, the Simulated Student generated the production rule that leads Ax=B to x=A instead of x=B/A.

This type of ambiguity happens when there are multiple ways to sequence operators to replicate the demonstration (e.g., get a coefficient of the left-hand side vs. divide right-hand side with the coefficient).

Structure Ambiguity: When the equation $2x–4x+3=4$ leading to $2x–4x=4–3$ was demonstrated for a production rule to transfer a constant term to the other side, the Simulated Student generated a RHS operator sequence that moves the *last term* on the left-hand side of the equation. This rule is overly specific and might result in semi-optimal solution, say, leading $2x+3–4x=4$ to 2x+3=4+4x.

This type of ambiguity happens when there are multiple ways to identify the object to be manipulated (e.g., the last term vs. the first constant term).

Impact of the level of detail in a demonstration

With the abbreviated demonstration, incorrect (overly specific) production rules were occasionally generated when the training problems had both operator and/or structure ambiguities, but these incorrect rules were eventually fixed when more demonstration instances were provided that served as the counter examples to disambiguate previous instances. For example, $4x=8$ leading to $x=2$ served to disambiguate the $3x=9$ example noted above.

The expressive demonstration, on the other hand, was less affected by operator ambiguity. For example, given that $3x=9$ leads to $3x/3 = 9/3$, the Simulated Student successfully generated a production rule that divides the left-hand side of the equation with the coefficient of the variable term in the original equation.

The difference in the degree of detail in demonstration also affects the complexity of the search. With abbreviated demonstrations, the search complexity increased drastically because the Simulated Student had to search all implicit operators that were abridged in the demonstration. As a result, when the number of RHS operators increases, the search becomes impractical as shown in Table 3. The table shows a time in second to complete a search, and the number of nodes expanded in the search space. A dash mark '–' shows that there was no rule generated at the specific operator length.

In summary, the detailed demonstrations are more practical for two reasons: (1) it decreases the chance of learning incorrect rules from operator ambiguity; and (2) the resulting production rules tend to have fewer RHS operators and search is less expensive.

Table 3: Comparison of the search complexity with different degree of detail in demonstration

Number of operators in RHS	Detailed demonstration		Abbreviated demonstration	
	Time [sec]	Space	Time [sec]	Space
1	0.00	1	0.00	1
2	0.12	19	0.04	6
3	10.21	1452	40.31	3563
4	–	–	2396.52	212780

6 Conclusion

The Simulated Students are a useful tool for authors of Cognitive Tutors, especially for the novice authors who are not familiar with cognitive modeling or AI programming.

The empirical studies showed that there is a significant effect in ordering training problems where an effective ordering involves instances of rule application that differ at the level of focus of attention, namely, the topological configuration to apply a production rule. It would be interesting to see if this observation also holds in learning for human students.

The problem of ambiguity is interesting, because similar performance can be observed in human learning. This suggests a task for future work: since both types of ambiguity are caused by having multiple interpretation of the demonstration, the Simulated Student could detect an ambiguity and ask the author for more information, in order to disambiguate the problem automatically.

References

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence, 42*(1), 7-49.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *Journal of the Learning Sciences, 4*(2), 167-207.

Blessing, S. B. (1997). A Programming by Demonstration Authoring Tool for Model-Tracing Tutors. *International Journal of Artificial Intelligence in Education, 8*, 233-261.

Cypher, A. (Ed.). (1993). *Watch what I do: Programming by Demonstration*. Cambridge, MA: MIT Press.

Jarvis, M. P., Nuzzo-Jones, G., & Heffernan, N. T. (2004). Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. In J. C. Lester (Ed.), *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 541-553). Heidelberg, Berlin: Springer.

Koedinger, K. R., Aleven, V. A. W. M. M., & Heffernan, N. (2003). Toward a Rapid Development Environment for Cognitive Tutors. In U. Hoppe, F. Verdejo & J. Kay (Eds.), *Proceedigns of the International Conference on Artificial Intelligence in Education* (pp. 455-457). Amsterdam: IOS Press.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring Goes to School in the Big City. *Journal of Artificial Intelligence in Education, 8*, 30-43.

Mertz, J. S. (1997). Using A Simulated Student for Instructional Design. *International Journal of Artificial Intelligence in Education, 8*, 116-141.

Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning, 5*(3), 239-266.

Tecuci, G., & Keeling, H. (1999). Developing an Intelligent Educational Agent with Disciple. *International Journal of Artificial Intelligence in Education, 10*, 221-237.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31*(1), 1-40.