Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2005). An Intelligent Authoring System with Programming by Demonstration. In *Proceedings of the Japan National Conference on Information and Systems in Education*. Kanazawa, Japan.

An Intelligent Authoring System with Programming by Demonstration

Noboru Matsuda William W. Cohen Kenneth R. Koedinger

School of Computer Science
Carnegie Mellon University

Abstract: Cognitive Tutors are known to be very effective, but with only a significant cost of cognitive modeling and programming to build a cognitive model representing domain principles and skills, which is written as a set of production rules. This study is building an intelligent authoring system that helps authors build a Cognitive Tutor. The basic idea is that instead of writing a cognitive model by hand, the authors are asked to demonstrate solutions. A machine-learning agent, called the Simulated Student, observes the demonstrations and induces a set of production rules that are generalizations of solutions demonstrated. An evaluation of the Simulated Student on an example domain of algebra equation showed that after 10 problems were demonstrated with 10 different skills (each corresponding to a production rule), 7 production rules were learned correctly. Two other production rules were plausible, which means that they produced correct actions, but since their conditions were overly general, their application might be strategically redundant.

Keyword: Cognitive Tutor, intelligent authoring system, Simulated Student, inductive logic programming, machine learning

1. Introduction

The aim of this study is building an intelligent authoring system to build Cognitive Tutors, or intelligent tutoring systems. The major challenge is an application of *programming by demonstration* (PBD) in building a computer agent that learns a cognitive model for a target domain subject by observing demonstrations performed by the authors.

While Cognitive Tutors are known to be very effective [1], a notorious amount of time for development and author training is required to build a practical tutor. Especially, it requires knowledge of the subject matter, a good understanding of the prior abilities of the students who will use the system, and extensive programming skills. Our solution is to construct a system in which an author can construct a graphical user interface (GUI) for a cognitive tutor, and then use this GUI to present examples of how the human students should solve the problems. A PBD learning system, called a *Simulated Student*, will then generalize these examples and build a set of production rules for solving problems in the task domain.

2. Building Tutors by Demonstration

Authors first build a GUI for their desired tutor. To do this, they use the Cognitive Tutor Authoring Tools [2], which basically are a collection of tools, including a GUI builder, to build a Cognitive Tutor.

This research was supported by the Pittsburgh Science of Learning Center funded by National Science Foundation award No. SBE-0354420.

Next, authors need to specify all *predicate symbols* and *operator symbols* appearing in production rules. A predicate symbol represents a test for a specific feature. An operator takes one or more arguments and returns a single value. Both predicate symbols and operators are task dependent. They may be written by advanced authors.

The authors then use the GUI and solve a number of problems just in a way that human students are supposed to perform. These demonstrations would then be fed to the Simulated Student to induce production rules that are sufficient to replicate the demonstrations. Each problem-solving step must be annotated in such a way that the steps with the same production rule should have the same name. Also, authors are required to specify all GUI elements that are involved in a production rule. Since every single GUI element is associated with a unique working memory element (WME), this step is essentially identifying the WMEs that appear in a production rule. These GUI elements (or WMEs) are called the *focus of attention*.

Each time the author demonstrates a step, the Simulated Student induces production rules with the learning technique described in Section 3. The resulted production rules are then loaded to the Cognitive Tutor with the GUI component.

After authors solve a number of problems, the Cognitive Tutor may be ready to run, that is, the induced production rules are capable of solving problems correctly. To test the production rules, authors enter a new problem into the Cognitive Tutor, and let the tutor solve it. When the tutor shows an incorrect or undesired per-

formance, authors provide feedback by first clicking a [Wrong] button and then entering a correct value into a correct GUI element. This feedback then triggers a refinement of the incorrect production rule.

Lastly and optionally, authors may directly modify production rules, which is written in Jess language [3], to obtain a desired set of production rules.

3. Learning Technique

The Simulated Students apply three different techniques for the three major components of a production rule: working memory elements (WMEs), feature tests, and operators. Given focus of attention specified by the author, searching WMEs and operators can be done by a brute-force search. To search the shortest operator sequence, the Simulated Student utilizes the iterative-deepening depth-first search.

Brute-force searching for feature tests is computationally very expensive as they can involve relationships between WMEs. We use FOIL [4] for this task. Each time a problem-solving step is demonstrated, the simulated student generates a collection of *positive* and *negative* examples as input data for FOIL. A positive example is generated for a corresponding production rule for the step demonstrated, and a negative example is generated for other production rules. Those data are accumulated over the different problems; the number of positive and negative examples continuously increases as more steps are demonstrated.

4. Evaluation of Simulated Students

To evaluate a performance of the Simulated Students, we have conducted an evaluation with algebra equation as an example subject domain.

For the sake of efficiency, the evaluation was done in such a way that demonstrations were provided with a text file. The output from the Simulated Student (i.e., production rules) was manually examined. The evaluation was run on a PC with Pentium IV 3.4GHz processor with 1GB RAM.

We used 8 feature predicates and 13 operators as background knowledge. In total 44 steps were demonstrated to solve the 10 problems shown in Table 1. These problems were solved by 10 different rules. In other words, the Simulated Students were supposed to induce 10 production rules from 44 demonstrated steps.

When all 10 problems were demonstrated, there were 7 production rules that were *correct*. Two production rules were correct but overly general hence application of those rules might not be strategically optimal (i.e., yielding redundant steps). This type of rule is called a *plausible* rule. There was one rule that was *wrong*, meaning that an application of the rule produced a wrong action. **Figure 1** shows how learning occurred over sequential demonstrations. The graph shows the number of correct ("C"), plausible ("P"), and wrong ("W") rules at each of the problems demonstrated. It also shows a "Total" number of different skills demonstrated (i.e., number of production rules to be learned). The two plausible production rules were introduced at Problem 8. Therefore, there were only three opportunities for the Simulated Student to observe application of those rules.

Table 1: Problems used for the evaluation

3x = 6, 2x = 4, 4x = 12,
x – 5 = 3, x + 2 = 6,
2 = –3x + 11,
3x – 4 = 2,
2x + 3x = 3 + 7,
3x = 2x + 4,
3x – 3 = 2x + 5

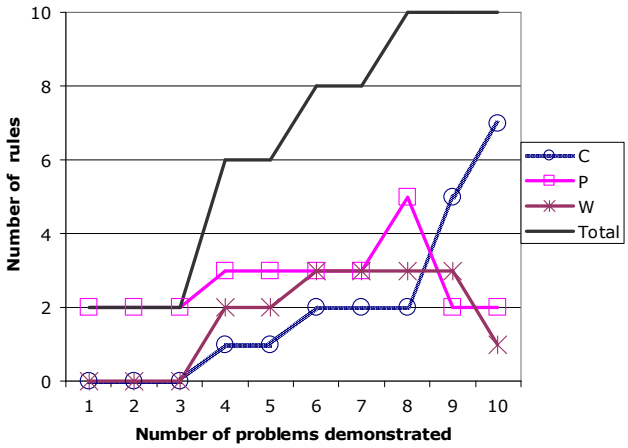

Figure 1: Results over sequential learning

5. Conclusion

The Simulated Student learns a cognitive model that can be easily communicated with the authors. Most importantly, it learns not only correct generalizations, but also incorrect ones that are consistent with those that a human student might produce as well. Such incorrect but plausible generalizations are often incorporated in Cognitive Tutors as *buggy rules*, which model typical human errors. Further study is needed to investigate the amount of demonstrations necessary to learn stable production rules as well as the effect of different curriculum on the quality of learning.

Reference:

[1] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive tutors: Lessons learned," *Journal of the Learning Sciences*, vol. 4, pp. 167-207, 1995.

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

[3] E. Friedman-Hill, *Jess in Action: Java Rule-based Systems*. Greenwich, CT: Manning, 2003.

[4] J. R. Quinlan, "Learning Logical Definitions from Relations," *Machine Learning*, vol. 5, pp. 239-266, 1990.