Automating the Generation of Production Rules for Intelligent Tutoring Systems

John Stamper

University of North Carolina at Charlotte

Key words: *Intelligent Tutoring Systems, Production Rules*

Abstract:

Intelligent Tutoring Systems that adapt to an individual student have shown great promise; yet have failed to become ubiquitous in education. The main reason these systems have been slow to be embraced is the high level of time and expert knowledge that are needed to create successful student models. Much of the time involves experts creating the production rules that will cover all of the ways a student might solve a problem. In this research, we propose a method of automatically generating production rules using previous student data on a proposed problem set. In addition, our proposed method will continue to refine the production rules as new data is available. We compare the results of our method run on several different data sets and with the production rules generated by domain experts.

1 Introduction

The ability of Intelligent Tutoring Systems (ITSs) to adapt to individual students makes ITSs effective [3,6]. These types of systems have shown a " 2-sigma" improvement in achievement over non-adaptive instruction. The most successful of these systems require the construction of complex cognitive models that are applicable only to a specific tutorial in a specific field, requiring the time of experts to create and test these models on students. Over the past few years some of the most promising ITSs utilize a cognitive model based on ACT-R Theory [2]. ACT-R Theory utilizes production rules in the model. The development of these production rules requires time and expert knowledge to create. Some systems including RIDES and DIAG [7] use examples performed or inputed by the author to develop the rules. Another system, REDEEM, was built to ameliorate the time needed to create a full blown ITS, and allow teachers to apply their own teaching strategies in an existing computer-based training (CBT) system. REDEEM has been shown to be more effective than a non-expert human tutor in improving student test scores [1], but not as effective as ITSs with complex production rules systems. Towards the goal of simplifying the creation of effective ITSs, we propose a method of automatically generating production rules using previous student data on a proposed problem set thus reducing the amount of time and expert knowledge needed to create these production rules. The system we propose is capable of continued refinement as new data is provided.

2 Background and Related Work

In this section we describe the system that generated the data used in the experiment, breifly explain the ACT-R Theory that is the basis of our production rules system, and explain the method we will use to create the production rules.

2.1 Deep Thought

Deep Thought [4] is a custom built system to provide Computer Aided Instruction (CAI) in the teaching of logic proofs. The interface is graphical and allows the students to visually connect premises and apply logic rules. One key feature of Deep Thought is the ability of the system to allow a student to work forward or backwards in the solving of a problem [5]. The Deep Thought program can be seen in Figure 1. One feature of the software is the ability to write out each step that a student makes to a log file. These log files have been compiled for several years now, and they contain a tremendous amount of interesting data that has not been thoroughly analyzed until this research. Although Deep Thought has been used sucessfully for the teaching of logic, the system is does not have any adaptive features and is not an ITS. The research in this paper represents part of our work to add true ITS capabilites to this software system.

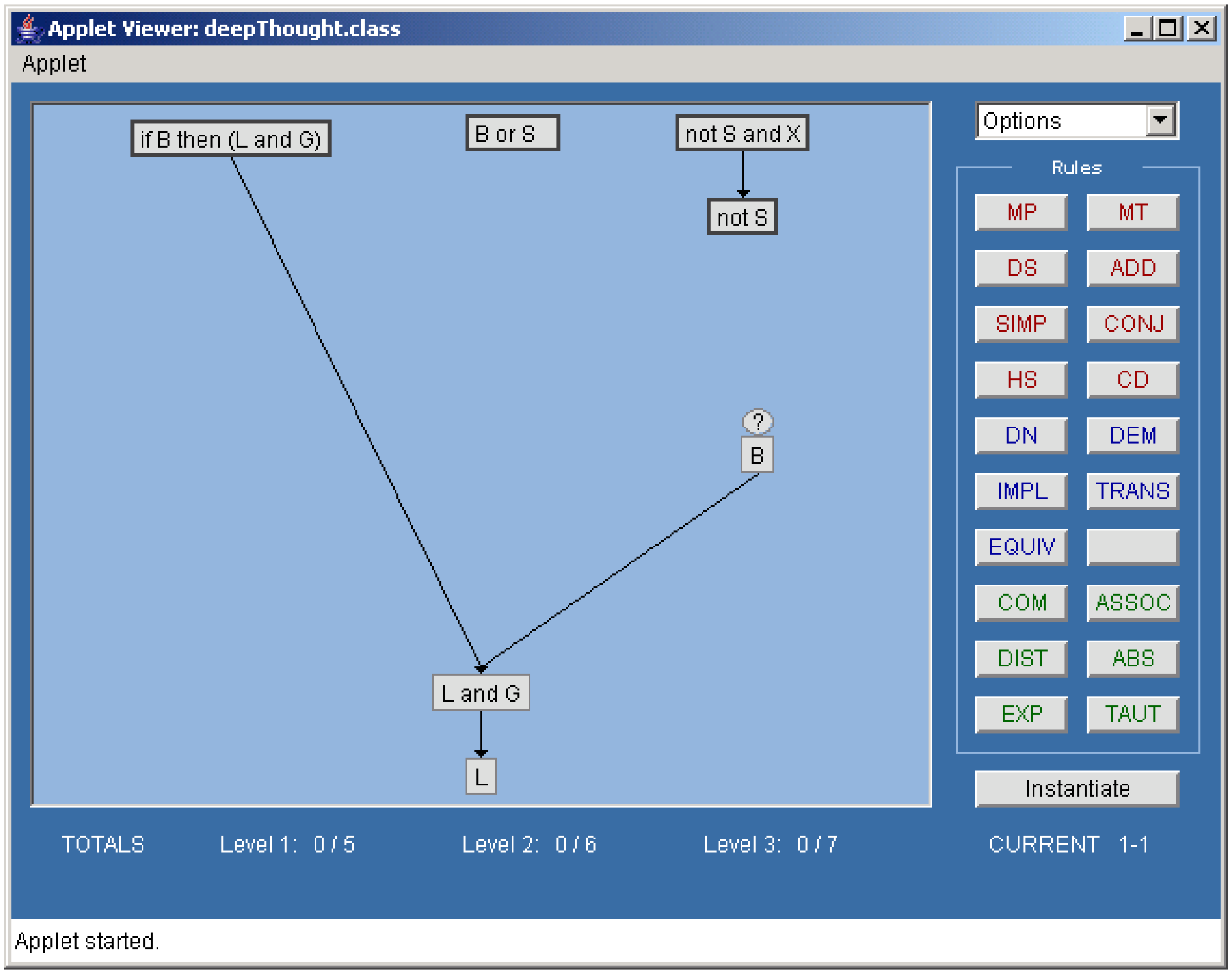

Figure 1. Deep Thought

2.2 ACT-R Theory

ACT-R Theory [2] is a cognitive architecture that has been successfully applied in the creation of Intelligent Tutoring Systems. The main component of the procedural memory module in ACT-R is a production rules system. These production rules look quite similar to a programming language composed of IF/THEN statements. The system uses pattern matching to determine the current production to be executed and the results from that execution. The creation of the production rules it the most time consuming part of developing a system based on ACT-R.

2.3 Method

Our proposed method uses a Markov decision process (MDP), which is defined via its state set S, action set A, transition probability matrices P, and reward matrices R. On executing action a in state s the probability of transitioning to state s´ is denoted P(s´ |s, a) and the expected reward associated with that transition is denoted R(s´|s, a). The production rules in ACT-R theory can be viewed as a state and an action that are described with an “If-Then” statement. The state is described in the “If” portion and the action in the “Then” portion. Our method takes the current premises and the conclusion as the state and the student’s input as the action. By building the MDP over a class’s worth of data we generate a significant number of rules and get the probabilites that students reached a given state. We can also determine which rules are good and bad by analyzing if the rules helps us get closer to the goal state. We then apply a reward for reaching each state. Good states, states that get us closer to the goal, recieve a higher reward than bad states that do not help us reach the goal. We add a large reward for arriving at the goal state. By executing value iteration using a Bellman backup “optimal” solutions can be found [8]. These “optimal” solutions represent the path to the solution that contains the fewest number of steps.

3 Experiment

In this section we describe the data, explain the experimental setup, and state our hypothesis.

3.1 Input Data

The data was obtained from three separate semesters worth of data from a Philosophy class called Deductive Logic that utilized the Deep Thought program for learning logic proofs. Students in the class used the Deep Thought program through three levels of problems – easy, intermediate, and difficult. We focused on three intermediate level problems, which we felt would produce a varied set of production rules. The data sets contained 140, 68, and 249 problem instances respectively.

3.2 Experimental Setup

Our first task was to clean the data for use in the experiment. Each of the three problems was extracted into its own file for each semester. Since Deep Thought can randomly order the premises and randomly assign different letters to the problem instances, we wrote a program to order the premises and change all the letters used in the premises to be consistent. Then the method explained in section 2.3 was applied and the production rules were created. We divided the rules into good rules and bad rules. The results were then combined across the three semesters and compared with rules generated by experts.

3.3 Hypothesis

Our first hypothesis was that the same rules would be generated from each data set even though they included different problem instances. If this is true it will we can expect that we generated a good set of rules, and that our rules can be used to predict future instances of these problems.

Our second hypothesis was that the rules generated by our method should closely resemble the rules generated by the experts. Although our rules will look different and be more specific to the problem instances we would expect that the generated rule set will closely follow the rules created by the experts.

4 Results and Discussion

The resulting MDPs generated across the three semesters worth of data were nearly identical and the optimal solutions were the same for each problem. This result confirmed hypothesis number one and shows that the method is consistent when applied to different data sets. Each data set had the same number of good rules for each problem, although the larger data sets did contain more bad rules. This makes sense since there are many possible options to apply to each step in solving a logic problem. In fact, the bad rules are often the most difficult for experts to derive and we were very happy to see the diverse set that was generated.

In comparing our method to the expert generated rules, our method produced more good and bad rules. This was expected since the experts' rules were more generalized and our rules were based on specific premises in the problems. Below are examples of an expert derived rule and a generated rule are shown below.

Expert derived rule:
IF there is a AND in a premise THEN apply simplification

An equivalent generated rule from a specific problem looks like this:
IF (Premises = (KvA)>F,R^K,~R&P) *and (Conclusion* = F*) THEN apply simp to* R^K

These rules were derived from the same problem, but the automatically generated rule is much more specific to a given problem and state in that problem. We found that we were able to group our good rules into subsets around the experts' good rules, and in this format, our optimal solutions were the same as the experts. We generated significantly more bad rules than the experts and not all of these could be grouped in subsets like the good rules. This shows that students make mistakes that the experts might not predict, and our method should better model the errors students will make.

Once our rule sets were generated and grouped, we explored them with the experts. The experts were impressed with the rules and found the percentages of each rule being used to very interesting. Several bad rules, rules that were applied incorrectly, were used at a much higher percentage than the experts would have predicted.

5 Conclusion and Future Work

This research was an initial attempt to automate the generation of production rules for an expert system to be used in an ITS. The method presented was capable of generating both good and bad rules that compared favorably to the rules generated by experts. We feel the method can be used to create production rules for other domains as well. There is still much research to be done to refine this method. In addition to expanding the research to a larger number of problems, future work will address the subsets that were created and see if we can combine a number of generated rules into one expert rule automatically.

There are also opportunities to research how this method can be used to predict what mistakes a student may make on a problem. To make these predictions we plan to create different MDP's from the test data based on student performance, and then classify future students into a specific MDP which will allow for better prediction of the path the student may take to solve a particular problem.

We also plan to add true ITS capabilites to Deep Thought by creating a cognitive model within the program that utilizes ACT-R and our method. Once in place, we plan to research the effectiveness of Deep Thought with and without the intelligent tutoring. We belive there will still be a need for expert input into rule generation, but our method will save significant time and effort.

6 Acknkowlegments

We would like to acknolwdge the following individuals for their help in carrying out this research. Dr. Marvin Croy, Professor of Philosophy at the University of North Carolina at Charlotte was instrumental in getting access to Deep Thought and the data used in this experiment. He also provided a list of expert derived rules for the selected problems. Also, Dr. Tiffany Barnes, Assistant Professor of Computer Science at the University of North Carolina at Charlotte provided expert knowledge from her teaching of logic in discrete mathematics.

References:

[1] Ainsworth, S.E., Major, N., Grimshaw, S.K., Hayes, M., Underwood, J.D., Williams, B. & Wood, D.J. 2003. REDEEM: Simple Intelligent Tutoring Systems From Usable Tools, in T. Murray, S. Blessing & S.E. Ainsworth (eds). Advanced Tools for Advanced Technology Learning Environments. pp. 205-232. Amsterdam: Kluwer Academic Publishers.
[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. 1995. Cognitive Tutors: Lessons Learned. The Journal of the Learning Sciences, 4(2), 167-207.
[3] Conati, C., A.S. Gertner, & K. VanLehn. 2002. Using Bayesian networks to manage uncertainty in student modeling. User Model. User-Adapt. Interact. 12(4): 371-417.
[4] Croy, M. 1999. Graphic Interface Design and Deductive Proof Construction, Journal of Computers in Mathematics and Science Teaching,18(4), 371-386.
[5] Croy, M. 2000. "Problem Solving, Working Backwards, and Graphic Proof Representation," Teaching Philosophy, 23(2), 169-187.
[6] Heffernan, N.T. & K.R. Koedinger. 2002. An intelligent tutoring system incorporating a model of an experienced human tutor. Intelligent Tutoring Systems, p. 596-608.
[7] Murray, Tom. 1999. Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education, 10: 98-129.
[8] Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning. Cambridge, MA: MIT Press.

Author(s):

John, Stamper
University of North Carolina at Charlotte, Department of Computer Science
Charlotte, NC 28223
jcstampe@uncc.edu