A pilot study on logic proof tutoring using hints generated from historical student data

Tiffany Barnes[1], John Stamper[1], Lorrie Lehman[1], and Marvin Croy[2]
{tbarnes2, jcstampe, ljlehman, mjcroy}@uncc.edu
[1]Computer Science Department, [2]Philosophy Department,
University of North Carolina at Charlotte

Abstract. We have proposed a novel application of Markov decision processes (MDPs), a reinforcement learning technique, to automatically generate hints using historical student data. Using this technique, we have modified a an existing, non-adaptive logic proof tutor called Deep Thought with a Hint Factory that provides hints on the next step a student might take. This paper presents the results of our pilot study using Deep Thought with the Hint Factory, which demonstrate that hints generated from historical data can support students in writing logic proofs.

1 Introduction and Related Work

Our goal is to automate the creation of intelligent tutoruing systems (ITSs) using educational data mining. We present experimental results of using the Hint Factory, a novel technique that uses a Markov Decision Process (MDP), created automatically from past student data, to generate specific contextualized hints to transform existing computer aided instruction (CAI) tools into intelligent tutoring systems (ITSs).

Example-based authoring tools such as CTAT use demonstrated examples to learn ITS production rules [3]. In these tools, teachers work problems in what they predict to be frequent correct and incorrect approaches, and then annotate the learned rules with appropriate hints and feedback. This system has also been used with data to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) [4]. However, in both of these approaches, considerable time must be spent in identifying student approaches and creating appropriate hints.

In the Logic-ITA tutor, educational data mining was used to create hints that warn students of likely mistakes [6]. In contrast, we use a domain-independent method to build student models from data and automatically generate contextual, goal-oriented hints. In our prior work, we have visualized problem attempts to find students' areas of difficulty [8] and predicted availability and specificity of hints [1]. In this work, we present our experimental study of automated hint generation in the Deep Thought tutor using educational data mining.

2 The Hint Factory

The Hint Factory consists of the MDP generator and the hint provider. The MDP generator is an offline process, but the hint provider must be integrated with the CAI. In this experiment we modified the deductive logic Deep Thought tutor to provide hints while students work problems. The modifications needed were minimal, including tracking the student actions, passing them to the hint provider, and adding a hint button. Some work must be done to word the hints, but need only be done once.

The MDP Generator uses historical student data to generate a Markov Decision Process (MDP) that represents a student model, containing all previously seen problem states and student actions. Each action is annotated with a transition probability P and each state is assigned a value based on the MDP reward function R. On executing action a in state s the probability of transitioning to state s' is P(s' | s, a) and the expected reward associated with that transition is R(s'| s, a). Our method takes the current premises and the conclusion as the state, and the student's input as the action. Therefore, each proof attempt can be seen as a graph with a sequence of states (each describing the solution up to the current point), connected by actions. Specifically, a state is represented by the list of premises generated in the student attempt, and actions are the axioms (rules) used at each step.

We combine all student solution graphs into a single graph. We then use reinforcement learning to find an optimal solution to the MDP. In this work, we set a large reward for the goal state (100) and penalties for incorrect states (10) and for each action (1). We apply the value iteration reinforcement learning technique using a Bellman backup to assign reward values to all states in the MDP. An iterative gradient algorithm is used to calculate state values until convergence. The reward values for each state then indicate proximity to the goal, while probabilities of each transition show the frequency of taking an action in a certain state. Using the MDP, we create a hint file for each problem in the Deep Thought tutor. The hint file consists of all problem states generated in the MDP with available hints, i.e. non-errors with paths to the goal. Table 1 shows the number of student attempts used to create the MDPs, including the average, minimum, maximum, and expert lengths for each problem. The problems are listed in order of difficulty.

Figure 1 shows the graphical interface for Deep Thought, a custom CAI for logic proofs [2]. Our new hint button appears, as shown at the lower right in Figure 1, when a student loads a problem with hints. The button is bright yellow to make it more visible. When the hint button is pressed, the hint provider searches for the current state and, if a successor state exists, the successor with the highest value is used to generate a hint sequence. We work with instructors to determine the wording and order of hints. However, these are easily changed, and experiments can verify the appropriateness and effectiveness of the chosen hints.

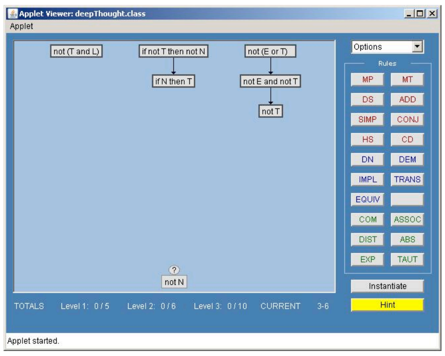

Figure 1: Problem 3.6 in Deep Thought.

Table 1: Average statistics for completed attempts used to create MDPs. Length includes errors; Expert reflects the length of the shortest expert proof.

Problem	**3.6**	**3.8**	**3.2**	**3.5**
Attempts	26	25	16	22
Length	8.0	11.9	11.3	18.8
Correct	5.5	11.2	9.0	16.7
Errors	1.1	1.1	0.9	3.1
Time	3:23	6:14	4:25	9:58
Expert	3	6	6	8

Our hints were designed based on instructor tutoring, research on hint strategies, and consistency with existing tutors. Both of our logic instructors prefer hints that help students set intermediate goals, as in [5]. We also use common pointing hints that help focus attention and bottom-out hints that reveal the answer. For a given state, we generate a hint sequence of four parts as shown in Table 2. Each time the hint button is pressed, the next hint is given. Once a student performs a correct step, the hint sequence is reset. For each hint request, we record the time, hint sequence number, and the total number of hint requests.

Table 2: Hint sequence for problem in Figure 1, derived from example student data

Hint #	Hint Text	Hint Type
1	Try to derive not N working forward	Indicate goal expression
2	Highlight if not T then not N and not T to derive it	Indicate the premises to select
3	Click on the rule Modus Ponens (MP)	Indicate the rule to use
4	Highlight if not T then not N and not T and click on Modus Ponens (MP) to get not N	Bottom-out hint

4 Pilot Study

The main goal of this experiment was to test the capability of the Hint Factory to generate hints for actual students. Our secondary goal was exploratory, to determine how students used the provided hints, to inform our future work. Once the Hint Factory was added to Deep Thought, we generated MDPs and hint files for several Deep Thought problems. Since the current semester was already underway, we chose four level 3 problems from Deep Thought, 3.2, 3.5, 3.6, and 3.8. In case of unexpected errors, we enabled the instructor to quickly disable hints in Deep Thought. Fortunately, the software worked well and this was not necessary.

MDPs were generated using Deep Thought data from two 2007 Deductive Logic philosophy courses: spring (30 students) and summer (20 students). The data were cleaned by removing all incomplete proofs and log files with missing data. Forty students in the spring 2008 course were assigned to work these four problems.We hypothesized that, with hints, a higher percentage of students (by class and by attempts) would complete the proofs. Class participation and completion rates for the experimental class were much higher than the source class. For 2008, the attempt and completion rates were 88 and 83%, respectively, out of 40 students. For 2007, these rates were at most 48%, out of 50 students. This may be due to a novelty effect. Figure 2 shows the percent of solution attempts that are complete for the source (2007) and hints (2008) groups. For all problems but 3.5, there was a slightly higher percent complete with hints available. Problem 3.5 showed much higher completion rates for the hints group. Figure 3 shows a comparison in behavior during complete and partial solutions. Partial problems were longer by both time and the number of actions. Complete solutions have fewer errors and deletions and more hint usage.

Figure 4 shows the number of hints for each problem, broken down by color into the distribution of hint sequence length. We hypothesized that, as learning occurs, the usage of shorter hint sequences should increase. We do not have reliable data on the sequence of problems that students performed, so we have ordered problems by difficulty. We see here that students did use more hints as we move from less to more difficult problems, and the number and proportion of hint sequences of length 1 seems to go up from 3.6 to 3.8 and from 3.2 to 3.5. In our future work we will investigate the particular effects of using hints in transfer of learning.

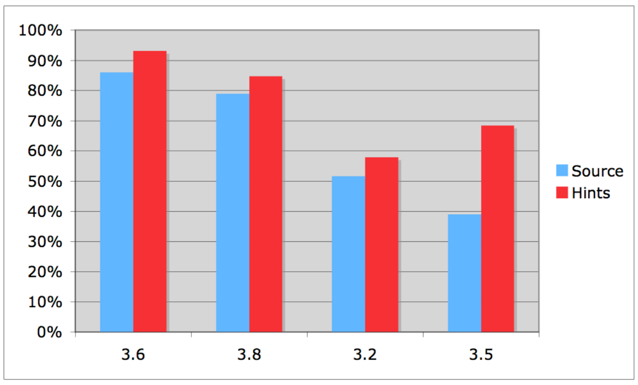

Figure 2: Percent attempt completion between the source and hints groups

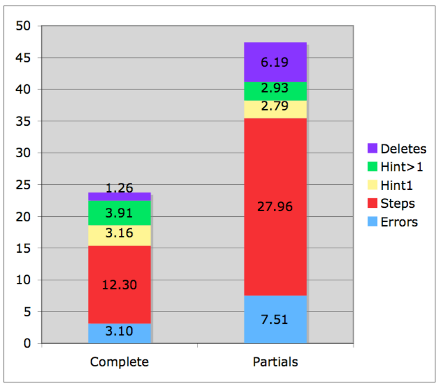

Figure 3: Comparison of behavior between complete and partial solutions

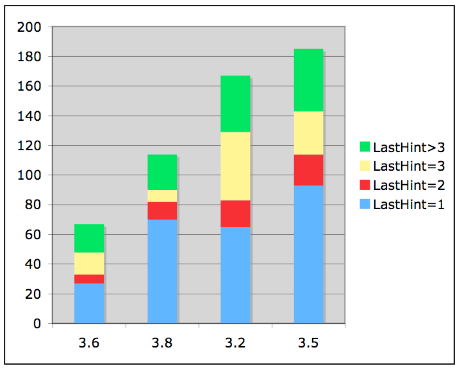

Figure 4: Distribution of hint sequences by seq. length

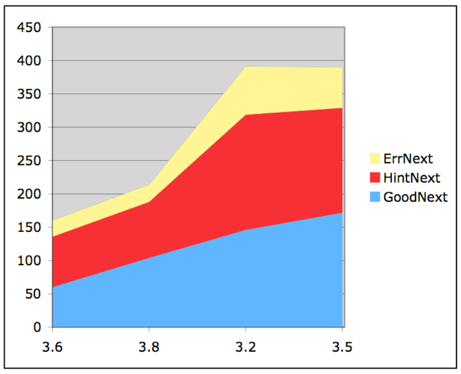

Figure 5: Number of steps after a hint that are correct (good), hints, or errors

Another way to measure the effectiveness of hints is to examine behavior just after receipt of a hint. We therefore investigated the number of errors, correct or good steps, and hint requests immediately following a hint, as shown in Figure 5. The proportion of good steps just after a hint goes consistently up, while there is a jump in the number hints and errors requested between problems 3.8 and 3.2. When we examine the difference between 3.2 and 3.5, we see more good steps, and slightly more hints and fewer errors just after a hint. Along with its higher completion rate, this suggests that the hints may be more effective for 3.5.

Table 4 shows the hint usage and availability for all 2008 completed and partial attempts. "Moves" is the total number of non-error student actions in the interface. In our prior feasibility study, we predicted that proof MDPs built using 16-26 attempts on problem 3.5 have a probability of providing hints 56-62% of the time [1]. In this experiment, if a student had pressed the hint button after every move taken, a hint would have been available about 48% of the time. This is lower than our prediction. However, the existence of the hint button may have changed student behavior.

We were encouraged by the comparison of this rate with the hint availability when students requested hints. Table 4 shows that a hint was delivered for 91% of hint requests. This suggests that hints are needed precisely where we have data in our MDPs from previous semesters. We plan to investigate the reasons for this surprising result with further data analysis and experiments. It is possible that there are a few key places where many students need help. Another explanation is that, when students are performing actions that have not been taken in the past, they may have high confidence in these steps and need no help.

Table 4: Hint usage and availability by problem, including all solution attempts in Spring 2008

Problem	3.2	3.5	3.6	3.8	Total
Attempts	69	57	44	46	216
Moves	999	885	449	552	2885
Moves w/ Avail. Hints	442	405	230	269	1346
% Moves w/ Avail. Hints	**44.2%**	**45.8%**	**51.2%**	**48.7%**	**47.9%**
Hint1 Requests	236	232	70	154	692
Hint1 Delivered	213	212	66	142	633
% Hint1s Delivered	**90.3%**	**91.4%**	**94.3%**	**92.2%**	**91.5%**

5 Conclusions and Future Work

We have presented the implementation and pilot experimental results on our system to use MDPs to create student models and automatically generated hints. Our approach to creating intelligent support for learning differs from prior work in authoring tutoring systems by mining actual student data to provide goal-oriented contextual hints. We have demonstrated that our method provides hints after just one semester, and can be adapted to add new hints as more data are collected. In our future work, we plan to use machine learning to generate hints for problems with no data and to test our method in varied domains.

References

[1] Barnes, T. and Stamper, J. Toward automatic hint generation for logic proof tutoring using historical student data, To appear in *Proc. Intl. Conf. Intelligent Tutoring Systems (ITS2008),* Montreal, Canada, June 23-27, 2008.

[2] Croy, M. Problem solving, working backwards, and graphic proof representation, *Teaching Philosophy* 23 (2000), 169-187.

[3] Koedinger, K., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. *Intelligent Tutoring Systems*, Maceio, Brazil, 2004, pp. 162-173.

[4] McLaren, B., Koedinger, K., Schneider, M., Harrer, A., & Bollen, L. Bootstrapping Novice Data: Semi-automated tutor authoring using student log files, In Proc. Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes, *Intelligent Tutoring Systems,* Maceió, Brazil, 2004.

[5] McKendree, J.: Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction* 5(4), pp. 381-413. Lawrence Erlbaum, 1990.

[6] Merceron, A. & Yacef, K.: Educational Data Mining: a Case Study. In *12th Intl. Conf. Artificial Intelligence in Education*, Amsterdam, Netherlands, IOS Press, 2005.