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

Many students use computer aided instruction (CAI) to enhance their classroom learning, but most CAI lacks the ability to adapt to each individual student beyond simple answer checking. Intelligent tutoring systems can provide adaptive instruction and have been shown to be effective, but are not universally seen in CAI because they are difficult to create [14]. Our research is an attempt to make intelligent tutors more accessible by simplifying their creation using educational data mining and machine learning techniques. In particular, we seek a path for educators to add intelligent tutoring capabilities to existing CAI without significantly rewriting the existing software. The Hint Factory is a novel technique that uses a Markov decision process, created from past student data, to generate specific contextualized hints for students using CAI. In this paper, we describe the process of adding the Hint Factory to a software program used to teach deductive logic in an undergraduate Philosophy course, and we report the initial results of a pilot study to test the hint generation capabilities in a real class setting.

2 Related Work

The problem of offering individualized help and feedback is not unique to logic proofs. Through individual adaptation, intelligent tutoring systems (ITS) can have significant effects on learning, but take considerable time to construct [14]. Constraint-based tutors, which look for violations of problem constraints, require less time to construct and work well for less procedural problems [13]. However, constraint-based tutors can only provide condition violation feedback, not goal-oriented feedback that has been shown to be more effective [11]. Example-based authoring tools such as CTAT use demonstrated examples to learn ITS production rules [9]. 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) [10]. However,

in both of these approaches, considerable time must still be spent in identifying student approaches and creating appropriate hints.

Machine learning has also been used to improve tutoring systems. In the ADVISOR tutor, machine learning was used to build student models that could predict the time students took to solve arithmetic problems, and to adapt instruction to minimize this time while meeting teacher-set instructional goals [5]. In the Logic-ITA tutor, student data was mined to create hints that warned students when they were likely to make mistakes using their current approach [12]. Another logic tutor called the Carnegie Proof Lab uses an automated proof generator to provide contextual hints [16].

Similar to the goal of BND, we seek to use student data to directly create student models for an ITS. However, instead of feeding student behavior data into CTAT to build a production rule system, our method generates Markov Decision Processes that represent all student approaches to a particular problem, and use these MDPs directly to generate hints. In [3], we used visualization tools to explore how to generate hints based on MDPs extracted from student data. In [8], we applied the technique to visualize student proof approaches to allow teachers to identify problem areas for students. In [4] we performed a feasibility study for hint generation using historical student data, and found that we could have made hints available for 71% of past student steps using one semester of past data. Our results indicated valuable tradeoffs between hint specificity and the amount of data used to create an MDP.

Our method of automatic hint generation using previous student data reduces the expert knowledge needed to generate intelligent, context-dependent hints and feedback. The system is capable of continued refinement as new data is provided. In this work, we discuss how we added the Hint Factory to Deep Thought and the results of our initial pilot study of the system. Although our approach is currently only appropriate for generating hints for specific problems with existing prior data, we believe that machine learning applied to MDPs may be used to create automated rules and hints for new problems in the same domain.

3 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.

3.1 The MDP Generator

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 $P(s' \mid s, a)$ and the expected reward associated with that transition is $R(s' \mid s, a)$. 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, by taking the union of all states and actions, and mapping identical states to one another. Once this graph is constructed, it represents all of the paths students have taken in working a proof. At this step reinforcement learning is used to find an optimal solution to the MDP. For the experiments in this work, we set a large reward for the goal state (100) and penalties for incorrect states (10) and a cost for taking each action (1). Setting a non-zero cost on actions causes the MDP to penalize longer solutions. We apply the value iteration reinforcement learning technique using a Bellman backup to assign reward values to all states in the MDP [16]. Equation 1 shows V(s) for each state s, where R(s) is the reward for the state, γ is the discount factor (set to 1), and $P_a(s,s')$ is the probability that action a will take state s to state s' can be seen in equation 1.

$$V(s) := R(s) + \gamma \max_a \sum_{s'} P_a(s,s')\, V(s') \qquad (1)$$

For value iteration, V is calculated for each state until there is little change in the function over the entire state space. The reward values for each state then indicate how close to the goal a state is, while probabilities of each transition reveal the frequency of taking a certain action in a certain state. Using the MDP, we create a hint file for each problem in the tutor. The hint file consists of all problem states generated in the MDP with available hints. This includes all states that are not errors and have a subsequent path to the problem solution.

3.2 Deep Thought

Deep Thought is a custom CAI tool that allows students to practice solving logic proofs [6][7]. As shown in Figure 1, Deep Thought's graphical interface allows the students to visually connect premises and apply logic rules. 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 a new problem with hints is selected, the hint provider loads the entire hint file into memory.

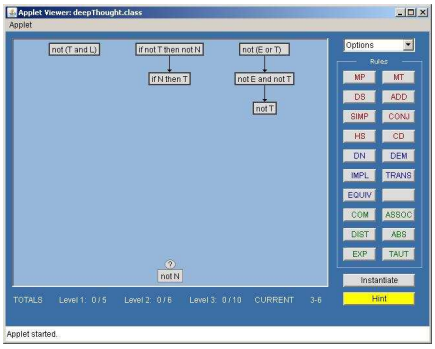

Figure 1: The Deep Thought Interface, with problem 3.6 partially completed

3.3 The Hint Provider

When the hint button is pressed, the hint provider searches for the current state in the MDP and checks that a successor state exists. If it does, the successor state with the highest value is used to generate a hint sequence. When attaching the hint provider to an existing CAI, we work with instructors to determine the wording and order of hints. However, these variables are easily changed, and experiments can verify the appropriateness and effectiveness of the chosen hints.

We worked jointly with logic instructors to construct an appropriate sequence of hints to generate from successor states. Our choices were based on one-on-one tutoring strategies, research on hint strategies, and consistency with existing tutors. In one-on-one tutoring, both instructors prefer hints that help students set intermediate goals, as has been shown to be effective in [11]. Existing tutors use several additional types of hints, including pointing hints and bottom-out hints. Pointing hints help focus user attention, while bottom-out hints essentially tell students the answer [17].

In this experiment, a hint sequence refers to hints that are all derived based on the same current state. A hint sequence consists of four types of hints: 1) indicate a goal expression to derive, 2) indicate the rule to apply next, 3) indicate the premises where the rule can be used, and 4) a bottom-out hint combining 1-3. For the problem state seen in Figure 1, the hint sequence seen in Table 1 would be generated. For each state, four distinct hints are generated. If a student requests a hint, then makes an error, and requests a hint again, the next hint generated is the next one in the current sequence. Once a student performs a correct step, the hint sequence is reset.

Whenever the hint button is pressed the hint provider records the time, the hint sequence number and text, and the total number of hints requested so far in the problem. In our future work we plan to also record the current state and successor state, and the time elapsed in the problem since the last step. These items will enable us to further explore and understand hint usage.

Table 1: Example hint sequence derived from example student solution

Hint #	Hint Text
1	Try to derive not N working forward
2	Highlight if not T then not N and not T to derive it
3	Click on the rule Modus Ponens (MP)
4	Highlight if not T then not N and not T and click on Modus Ponens (MP) to get not N

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. Table 2 shows the problems used and the minimum number of rules needed to solve them.

MDPs were generated using data extracted from Deep Thought solutions from two 2007 Deductive Logic courses taught in the philosophy department: spring (30 students) and summer (20 students). The data were cleaned by removing all incomplete proofs and log files with missing data. Table 3 shows the number of student attempts used to create the MDPs for each problem, the average length of the attempt with minimum and maximum lengths. Based on these and the expert data in Table 2, the problems can be listed in order of difficulty from 3.6, 3.8, 3.2, 3.5.

Forty students in the spring 2008 course were assigned to work these four problems (as many times as desired). We hypothesized that, with hints, a higher percentage of students would complete the given proofs. This can be measured in two ways: by class and by attempts. 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, since the 2008 class was asked to test hints.

Table 2: Deep Thought problems where hints were added (> is implies)

Prob.	Problem Description	Expert length	Rules used; features
3.6	-(T&L), -T>-N, -(E∨T)/-N	3 steps	DEM,SIMP, MT
3.8	Y=P, -Y>-C, -P=-C /Y>C	6 steps	EQUIV(2), SIMP(2), TRANS, HS
3.2	(A>-B)∨C, -C, D∨B/-D>-A	6 steps	DS, TRANS, DN, DN, IMPL, HS
3.5	K>M, Z>R, -(K>R)/M&-Z	8 steps	IMPL, DEM, DN, SIMP, SIMP, MP, MT, CONJ

Table 3: Spring and summer 2007 data used to create MDPs. Attempts include only completed proofs. Length includes correct and incorrect steps.

Problem	**3.6**	**3.8**	**3.2**	**3.5**
# Complete Proofs	26	25	16	22
Average Length	8.0	11.9	11.3	18.8
Std Dev Length	5.9	3.1	4.0	12.4
Avg. Correct Steps	5.5	11.2	9.0	16.7
Average Errors	1.1	1.1	0.9	3.1
Time	3:23	6:14	4:25	9:58

Figure 1 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 2 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. These results suggest that some scaffolding may help identify unhelpful behaviors and be used to train students to more effectively learn and use help as in [1].

Figure 3 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 plan to conduct studies to investigate the particular effects of using hints in a situation and the time taken to solve a similar problem later.

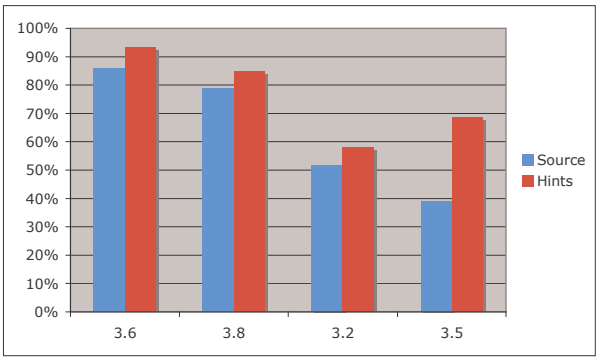

Figure 2: Percent attempt completion between the source and hints groups

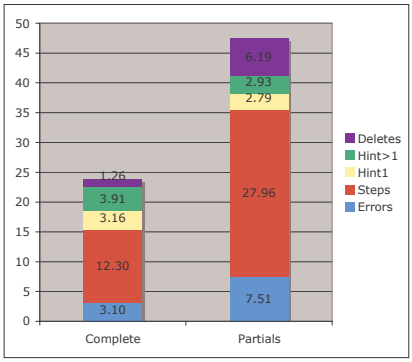

Figure 3: Comparison of behavior between complete and partial solutions

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 4. 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.

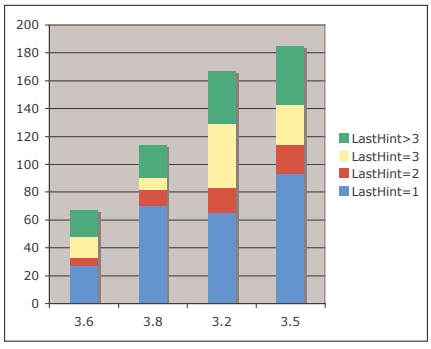

Figure 4: Distribution of hint sequences by sequence length.

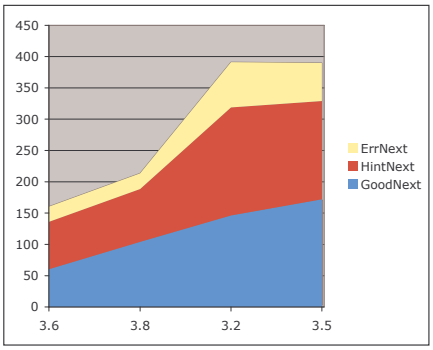

Figure 5: Number of steps after a hint that are correct (good) steps, hints, or errors.

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 built a model to predict the probability that we could provide a hint based on the size of the MDP. In that 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 [4]. In our current 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. In Table 4, Hint1 Requests counts the number of times a first hint was requested (since hints beyond the first in a sequence are all available if the first one is). Hint1 Delivered shows the number of times a first hint was provided. "% Hint1s Delivered" shows that, over 91% of the times a hint was requested, a hint was available. This percentage quite exceeded our expectations. 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 on this dataset and with future 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

Our approach to creating intelligent support for learning differs from prior work in authoring tutoring systems by mining actual student data, rather than relying on teachers to add examples the system can learn from. Our tutor can already classify many errors students make. Adding the MDP to this tutor enables it to provide hints. This MDP can constantly learn from new student data. We note that on cold start for a new problem that has no student data, the system will still act as a problem-solving environment, but after even one semester of data is collected, a significant amount of feedback can be generated. As more data are added, more automated feedback can be generated. This research represents the implementation and a pilot study for our hint generation method in an actual classroom setting. We achieved our main goals, which were to verify that the software would work in a class setting, students would request hints, and hints would be available when requested.

More work is needed to understand the impact of our hint generator on student learning and satisfaction. This semester, we plan to survey the students who used the hints to measure student opinions, perceptions, and usage patterns. In future semesters we plan to conduct talk-alouds to determine how well the hints align with what students need to know in solving proofs, and how well the hints support students in setting sub-goals. These studies will also help us determine why some students

seemed to avoid hints while others abused them, as in [2]. We will also add hints to more problems and conduct additional studies and analyses to better understand the overall effects of giving students automatically generated hints. In our future work, we will continue to explore ways to learn general rules to build intelligent feedback and help with greater coverage and robustness. For instance, we plan to group students according to their proofs behavior and class performance, and create tailored MDPs for each group of students. In addition, we plan to create and implement different value functions for the MDP generator and perform experiments to see if certain students relate better different hint types.

In [3], we have proposed several reward functions that could be used in hint generation, including 1) expert, 2) typical, and 3) least error-prone. The reward function we have described herein reflects an expert reward function, where the value for a state reflects the shortest path to the goal state. Given the current state, when the Hint Button is pressed, we will select a reward function for the current student based on their student profile. If we have identified the student as an at-risk student, we may select the "least error-prone" reward function for generating hints. On the other hand, high-performing students would likely benefit from expert hints, while students between these two extremes may benefit from hints reflecting typical student behavior [3]. We also plan to limit the number of hints a student can use and still receive credit for working the problem. We believe that four hints is a fair number, to be used on a single state in sequence as above or on separate states in the same problem.

References

[1] Aleven, V., McLaren, B., Roll, I., & Koedinger, K. Toward tutoring help seeking: Applying cognitive modeling to meta-cognitive skills. In J. C. Lester, R. M. Vicari, & F. Paraguaçu (Eds.), *Proc. 7th Intl. Conference on Intelligent Tutoring Systems, ITS 2004* (pp. 227-239). Berlin: Springer Verlag, 2004.

[2] Baker, R. Modeling and understanding students' off-task behavior in Intelligent Tutoring Systems. *CHI 2007 Proceedings.* San Jose, CA, USA, April 28-May 3, 2007

[3] Barnes, T. and Stamper, J. Toward the extraction of production rules for solving logic proofs, In *Proc. 13th Intl. Conf. on Artificial Intelligence in Education, Educational Data Mining Workshop (AIED2007),* Marina del Rey, CA, July 9, 2007.

[4] 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.

[5] Beck, J., Woolf, B. P., and Beal, C. R. ADVISOR: A Machine Learning Architecture for Intelligent Tutor Construction. In: *7th National Conference on Artificial Intelligence*, pp. 552—557. AAAI Press / The MIT Press, 2000.

[6] Croy, M. Graphic Interface Design and Deductive Proof Construction, *Journal of Computers in Mathematics and Science Teaching*, 1999, 18(4), 371-386.

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

[8] Croy, M., Barnes, T., and Stamper, J. Towards an Intelligent Tutoring System for propositional proof construction. In P. Brey, A. Briggle & K. Waelbers (eds.), *Proc. 2007 European Computing & Philosophy Conf.*, Amsterdam: IOS Publishers.

[9] Koedinger, K., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. *Proc. 7th Intelligent Tutoring Systems Conference*, Maceio, Brazil, 2004, pp. 162-173.

[10] 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, *7th Intl. Conf. Intelligent Tutoring Systems (ITS-2004),* Maceió, Brazil, August 30, 2004.

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

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

[13] Mitrovic, A., Koedinger, K. & Martin, B. A comparative analysis of cognitive tutoring and constraint-based modeling. *User Modeling* 2003: 313.322.

[14] Murray, Tom. Authoring intelligent tutoring systems: An analysis of the state of the art. *Intl. J. Artificial Intelligence in Education,* 1999, 10: 98-129.

[16] Sieg, Wilfried. The AProS Project: Strategic Thinking & Computational Logic. *Logic Journal of IGPL*, 2007, Oxford University Press, 2007.

[16] Sutton, R. & A. Barto. *Reinforcement Learning: An Introduction*, 1998, The MIT Press, Cambridge, MA.

[17] VanLehn, K. The behavior of tutoring systems, *International Journal of Artificial Intelligence in Education*, 16 (2006), 227-265.