The Hint Factory: Automatic Generation of Contextualized Help for Existing Computer Aided Instruction

John Stamper[1,] , Tiffany Barnes[1], Lorrie Lehmann[1], Marvin Croy[2]

[1] Department of Computer Science,
[2]Department of Philosophy,
University of North Carolina at Charlotte,
{jcstampe, tbarnes2, ljlehman, mjcroy}@uncc.edu

Abstract. The Hint Factory is a novel application of Markov decision processes (MDPs), a reinforcement learning technique, to automatically generate contextualized hints from past student data. We describe the process of adding the Hint Factory to an existing, non-adaptive, 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.

Keywords: educational data mining, hint generation, Markov decision process

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. Intelligent tutoring systems can provide adaptive instruction and have been shown to be effective [3] [6], but are not universally seen in CAI because they are difficult to create [12]. 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 [12]. Constraint-based tutors, which look for violations of problem constraints, require less time to construct and can be used for problems that may not be heavily procedural [11]. However, constraint-based tutors can only provide condition violation feedback, not goal-oriented feedback, which has been shown to be more effective [8].

Example-based authoring tools such as CTAT use demonstrated examples to learn ITS production rules [7]. 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) [9]. 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 help predict the amount of time students took to solve arithmetic problems, and to adapt instruction to minimize this time while meeting teacher-set instructional goals [1]. 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 [10].

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 a Markov Decision Process (MDP) that represent all student approaches to a particular problem, and uses the MDP to directly to generate hints. In [2], we used visualization tools to explore how to generate hints based on MDPs extracted from student data and verified that the rules extracted by the MDP conformed to expert-derived rules and generated buggy rules that surprised experts. In [5], we applied the technique to visualize student proof approaches to allow teachers to identify problem areas for students. In [2], we demonstrated the feasibility of this approach by extracting MDPs from four semesters of student solutions in a logic proof tutor, and calculated the probability that hints could be generated at any point in a given problem. Our results indicated that extracted MDPs and our proposed hint-generating functions were able to provide hints over 80% of the time. Our results also indicated that we can provide 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 demonstrate the feasibility of our hint generation approach through simulation experiments on existing student data. 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 two main parts, the MDP generator and the hint provider. The MDP generator is an offline process, but the hint provider must be attached programmatically into the CAI. In this experiment we inserted software hooks into the deductive logic tutor called Deep Thought [4].

3.1 The MDP Generator

The MDP Generator is a process that is run against previous student data in order to generate a MDP of all student states that have been previously seen. In the MDP, each state is given a value based on how "good" the state is based on a given value function.

A Markov decision process (MDP) is defined by its state set S, action set A, transition probabilities P, and a reward function R [13]. 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). For a particular point in a student's proof, 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. Typically, 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 (but we set this at 1/10 the cost of taking an incorrect step). We apply the value iteration reinforcement learning technique using a Bellman backup to assign reward values to all states in the MDP [13]. The equation for calculating values 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.

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

For value iteration, V is calculated for each state until there is little change in the function over the entire state space. Once this is complete, the optimal solution in the MDP corresponds to taking a greedy traversal approach in the MDP [2]. 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.

Each state in the MDP is given a state number, state description, a set of actions from that state with the next state that the action leads, and a value. The state description consists of the current premises and conclusions on the screen. From a MDP generated for a specific problem, a hint file is generated. The hint file consists of all of the problem states generated in the MDP which have hints that can be given to the student. This means all states that are not errors and have a subsequent path to the problem solution.

3.2 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 [4]. The Deep Thought program can be seen in Figure 1. The Deep Thought software has 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 significant amount of data that will allow us to create MDPs for the hint generation. Deep Thought is written in Java and delivered to students via a web link. We have been given access to the complete source code.

In order to add contextualized hints to Deep Thought we first needed to add a hint button. The hint button was added to the main form, but is invisible at start time and will only become available when a student loads a problem that has a hint file associated with it. The hint button can be seen in the lower right corner in figure 1. The button was given a bright yellow color when hints were available to make students aware they could use the button. When a new problem is selected from the options drop down, a hook was added that looks for a hint file associated with that problem. If a hint file is found the hint provider code loads the entire hint file into memory.

3.3 The Hint Provider

Once the student starts working on a problem with hints available, each student step makes a call to the hint provider which checks to see if the current student state exists. If the student clicks the hint button, the hint provider will get read in the current state information and verify that hints are available for that state. Then the possible known actions from the current state are evaluated by comparing the values of the subsequent state. The subsequent state with the highest value will be used to generate a hint sequence.

A hint sequence consists of four types of hints that the hint provider will create. A hint sequence is derived as follows: 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. Using the hint sequence, a student can click the hint button four times at a given state and get the next hint in the sequence.

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

Whenever the hint button is clicked the hint provider also writes out hint information to the student data log file. We track when the hint button was clicked, what hint was given, hint sequence number, and total hints given for the current problem.

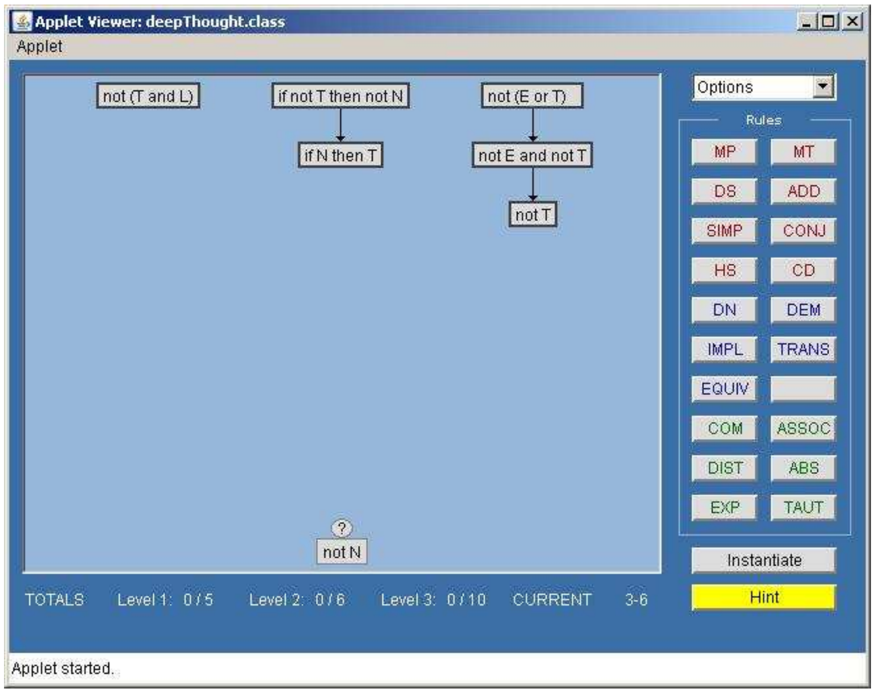

Fig. 1. The Deep Thought Interface, with problem 3-6 partially completed.

4 Pilot Study

Once the Hint Factory had been applied to the Deep Thought software, we proceeded to generate MDPs and hint files for several Deep Thought problems. Since the current semester was already under way, we chose four level 3 problems from Deep Thought and generated hint files. These problems were identified as 3-2, 3-5, 3-6, and 3-8. The student data used consisted of two semesters of data from previous classes that had completed the selected problems in Deep Thought. Table 2 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. The attempts used were hand validated and consisted of only completed proofs.

Table 2: Data Used to Create MDPs

Problem	3-2	3-5	3-6	3-8
Attempts	16	22	26	25
Average Length	11.25	18.82	8.00	11.88
Std Dev Length	4.01	12.39	5.87	3.10
Min Length	7.00	10.00	4.00	7.00
Max Length	18	58	30	19

Our main goal in the pilot study was to see how well the Hint Factory software worked in an actual class situation. We set up the system in such a way that if anything went wrong our software changes the instructor could remove the hints and go back to the original Deep Thought version quickly. Fortunately, the software worked well with no problems with the hint delivery.

The current class spent two weeks working on the Deep Thought level 3 problems and the resulting hint button usage can be seen in Table 3. These attempts include both completed and partial attempts. The Hint Available % shows how often a hint was available when it was asked for by the student. Clearly, our ability to give hints when requested was very good considering the low number of student attempts that were used to create the original MDPs used in the hint files. These results validate our previous results in [2]. Further analysis of the hint data is under way and we plan to explore the overall hint usage among individual students. From a high level view, we can see that some students used the hints frequently to the point of abusing them, while others never used hints.

Table 3: Hint Usage for problems

Problem	3-2	3-5	3-6	3-8
Attempts	69	57	44	46
Hint Button Hits	471	458	157	303
Hint Available	431	432	150	286
Hint Available %	91.51%	94.32%	95.54%	94.39%
Average Hints	6.82	8.03	3.57	6.63
Min Hints	0	0	0	0
Max Hints	22	44	17	37
Std Deviation	2.44	3.68	2.03	1.88

5 Conclusions and Future Work

Although our initial findings are promising, further analysis of the student data with hints needs to be done to draw conclusions about student learning. We plan to compare error rates, proof length, and complete vs. partial attempts data from the hint data and previous semester data. As of this writing this analysis is not complete.

This research represented the implementation and a pilot study for our hint generation method in an actual classroom setting. The main goals of the pilot study were to verify that the software would work in a class setting, to see if students would ask for hints, and verify that hints would be available when asked for by the student. The pilot study has shown that students will use the hints and our hint availability is high. Since the hints were only available for four of thirty problems used in the software, we hope to include more problems over the next several semesters. A survey of the students is planned for the end of the semester to gauge their feelings towards the hints. Further collection and analysis of the student data and hint data will be performed to test the overall effects of giving students hints. We also see the need for some talk alouds with students as they work with the hints. The data raised questions why some students avoided the hints, while others abused them. Our current setup allows the student to ask for unlimited hints. In future experiments limits to the number of hints will be added.

We plan to create and implement different value functions for the MDP generator and perform experiments to see if certain students relate better to a different type of hint. In our next experiment, students will be split in to two groups of higher and lower performing students. These groups will then be given hints generated from different MDPs. We plan to use a optimal strategy, and an error avoidance strategy to see if the different groups perform better with one of the types of hints.

References

1. 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)
2. Barnes, T. & 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,* Marina del Rey, CA (2007)
3. Conati, C. Gertner, A. S. and VanLehn, K. (2002). Using Bayesian Networks to Manage Uncertainty in Student Modeling. In User Model. User-Adapt. Interact, volume 12 (4).
4. Croy, M. Graphic Interface Design and Deductive Proof Construction. (1999). Journal of Computers in Mathematics and Science Teaching, 18 (4):371–386.
5. Croy, M., Barnes, T. & Stamper, J.: Towards an Intelligent Tutoring System for propositional proof construction. In Brey, P., Briggle, A. & Waelbers, K. (eds.), *European Computing and Philosophy Conference*, Amsterdam, Netherlands: IOS Publishers (2007)
6. Heffernan, N.T. and Koedinger, K.R. (2002). An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In Intelligent Tutoring Systems, pages 596–608.
7. Koedinger, K. R., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M.: Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *7th Intelligent Tutoring Systems Conference,* Maceio, Brazil, pp. 162-173. (2004)

8. McKendree, J.: Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction* 5(4), pp. 381—413. Lawrence Earlbaum (1990)
9. 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 (2004)
10. Merceron, A. and Yacef, K.: Educational Data Mining: a Case Study. In *12th Intl. Conf. on Artificial Intelligence in Education*, Amsterdam, The Netherlands,IOS Press (2005)
11. Mitrovic, A., Koedinger, K. & Martin, B.: A comparative analysis of cognitive tutoring and constraint-based modeling. *User Modeling*: 313-322. (2003)
12. Murray, T.: Authoring intelligent tutoring systems: An analysis of the state of the art. *Intl. J. Artificial Intelligence in Education*, 10: 98-129. (1999)
13. Sutton, S. and Barto, A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)